PhD thesis

Máté Hegyháti

DOI: 10.18136/PE.2015.585

Veszprém

2015

PhD thesis

Batch Process Scheduling: Extensions of the S-graph Framework

Máté Hegyháti

Supervisor: Ferenc Friedler, DSc

Doctoral School of Information Science and Technology
University of Pannonia

Veszprém

2015

BATCH PROCESS SCHEDULING:
EXTENSIONS OF THE S-GRAPH FRAMEWORK

Értekezés doktori (PhD) fokozat elnyerése érdekében

Írta:
Hegyháti Máté

Készült a Pannon Egyetem Informatikai Tudományok Doktori Iskolája keretében

Témavezető: **Dr. Friedler Ferenc**

Elfogadásra javaslom (igen / nem)

(aláírás)

A jelölt a doktori szigorlaton ..... % -ot ért el.

Veszprém, ...

a Szigorlati Bizottság elnöke

Az értekezést bírálóként elfogadásra javaslom:

Bíráló neve: ..... (igen /nem)

(aláírás)

Bíráló neve: ..... (igen /nem)

(aláírás)

A jelölt az értekezés nyilvános vitáján ..... % - ot ért el.

Veszprém, ...

a Bíráló Bizottság elnöke

A doktori (PhD) oklevél minősítése .....

.....

Az EDT elnöke

Abstract

In batch processes, a unit can be used for various steps of the production of several different products. This leads to several advantages over continuous systems, e.g., quick adaption to the changing market environment. This flexibility of batch processes, however, requires additional consideration in operational planning, as the equipment units must be scheduled with caution to satisfy all the practical constraints of the problem. Finding the most advantageous schedule is generally a complex problem, nevertheless it is a key component of the profitability of such systems. As a result, the scheduling of batch processes is a widely researched topic, with many different approaches published.

The goal of my PhD work was to extend the capabilities of the S-graph framework to be able to address a wider range of scheduling problems. Unlike other mathematical programming based techniques, the S-graph framework guarantees a feasible and globally optimal solution. There are, however, several practical features, that could not be tackled with the former S-graph based algorithms. In this thesis, extensions of the S-graph framework are presented to address limited-wait storage policies, throughput maximization-, and expected profit maximization problems. The developed algorithms are discussed in detail, and empirically tested on various case studies and literature examples.

Absztrakt

Szakaszos üzemű termelő rendszerek berendezései több különböző termék gyártásának lépéseinél is felhasználhatók, aminek köszönhetően az ilyen rendszerek könnyebben alkalmazkodnak például a változó piaci igényekhez, s további számos előnnyel rendelkeznek a folytonos rendszerek működésével szemben. Ezen szabadsági foknak köszönhetően ugyanakkor különös odafigyelés szükséges a tervezés során, hogy a berendezések ütemezése minden gyakorlati korlátozásnak eleget tegyen. A legelőnyösebb ütemezés megtalálása általában egy összetett feladat, mely kulcsfontosságú a hasonló rendszerek nyereséges működéséhez, így a téma széles körben kutatott, s számos módszer látott napvilágot a kapcsolódó szakirodalomban.

Doktori munkám célja az S-gráf módszertan kiterjesztése volt, megteremtve annak szélesebb körű alkalmazási lehetőségeit szakaszos üzemű rendszerek ütemezéséhez. A matematikai programozáson alapuló módszerekkel ellentétben az S-gráf módszertan korábban kidolgozott algoritmusai garantálják az optimális megvalósítható megoldást, azonban nincsenek felkészítve néhány fontos ipari korlátozás figyelembe vételére. Dolgozatomban az S-gráf módszertan több kiegészítése kerül bemutatásra, melyek alkalmazásával kezelhetők például az időkorlátos tárolások, vagy a profit valamint várható profit maximalizálását megcélzó feladatok. A kifejlesztett algoritmusok részletes ismertetését követően azok összehasonlítása kerül bemutatásra esettanulmányokon és irodalmi példákon keresztül.

Contents

List of Figures

List of Tables

List of Algorithms

Achnowledgements

First of all, I would like to thank to my supervisor and all members of the Scheduling Research Group at the Department of Computer Science and Systems Technology, Faculty of Information Technology, University of Pannonia, for helping me making my own contribution to the field of batch process scheduling. Prof. Ferenc Friedler, my supervisor aided my research with many suggestions, corrections, and continuous guidance thourough our evening meetings despite all the strains He beard as the rector of our University. Dr. Tibor Holczinger, my senior colleague at the research group provided me with many great insights and suggestions, and showed always infinite patience towards me, though I often meant a hindrance in our joint works. I am exceptionally thankful to all the students I had the luck to work with: Ákos Orosz, András Szoldatics, András Éles, Balázs Kovács, Benjámin Tóth, Olivér Ősz, and Zsolt Nemes, as they were very active, resourceful, creative, and stimulative colleagues during this last 5 years.

For providing a supportive aura and workplace for my research I am thankful to my friends, family, and all the staff (including lecturers, researchers, administrative and cleaning personnel, etc.) of the Department and the University, as well as the city of Veszprem itself. Special thanks is in order for Gergely Vakulya, who shared the same office with me thorough this last couple of years, and proved to be a supporting roommate, with whom I shared many scientifically inspiring discussions.

The Paul Erdos Talent Care Program and Dr. Ferenc Pinter played a key role shepherding me towards the academic career, for which I am now utmost grateful. Over my 21 years of being a student, many people contributed in developing skills necessary for graduating as a PhD student, including all my teachers, students, orchestra members, powerhiking fellows, and rivals in competitions.

This research has been supported by the European Union and Hungary and co-financed by the European Social Fund through the project TÁMOP-4.2.2.C-11/1/KONV-2012-0004 - National Research Center for Development and Market Introduction of Advanced Information and Communication Technologies.

Preface

Batch processes are becoming more and more important in the production industry due to their flexibility. This advantage, however, also comes as a shortcoming, as the units need to be scheduled, which results in an additional complexity for the planning of operations. The industry generates a wide range of batch scheduling problems, where the goal in general is to allocate the tasks of the process to the available equipment units in the most favorable way. Thus, scheduling is an important and unavoidable problem of batch processes, for which many approaches has been published in the literature over the past two decades.

Chapter 1 first presents the theoretical foundations of machine scheduling, which provides the basis for the practical, industrial problems described in the second part of the chapter. The approaches developed and published in the literature for batch scheduling problems are summarized in Chapter 2. The MILP programming based approaches presenting the majority of publications are discussed in detail. The model and algorithms of the S-graph framework and its past developments are detailed thoroughly in the second part of the Chapter, as it provides the basis for the new developments presented in this thesis.

In the next five chapters, the new results of the thesis are detailed. First, some modeling issues are addressed in Chapter 3, that appear in the literature approaches. The next three chapters provide extensions of the S-graph framework for throughput maximization (Chapter 4, limited-wait storage policy (Chapter 5), and stochastic profit maximization (Chapter 6), respectively. In Chapter 7, a generalized modeling framework is presented in order to extend the expressiveness of the S-graph framework.

Chapter 1

Scheduling problems

Scheduling problems appear in almost every part of life, e.g., finding the shortest path with a shopping list in a mall, selecting the best order of airplanes to land during a rush hour, or providing a robust execution plan for a construction project. The problems that arise at different parts of life often differ in many aspects, however, the underlying basics are the same: ***the goal is to assign tasks to some kind of available resources and time intervals in the most favorable way for a certain objective, while satisfying the constraints of the problem definition.*** Just like the aforementioned constraints and objectives vary, the terminology for the basic elements of the problem are also different at different fields of science. Tasks to be performed are often called jobs, activities, etc. The available resources in many cases are some kind of machines that are often called equipment or units as well. The main field of the present work is the scheduling of batch processes of chemical industries, thus the terms ***task*** and ***unit*** are used almost everywhere, which are the most accepted in this field.

The classification of scheduling problems is different for each field. However, there are some basic aspects that can be the basis of the categorization in almost any case. Regardless of the field specific attributes of a scheduling problem, it can be classified as either an ***online***, ***offline***, or ***semi-offline*** problem. In the case of offline problems, all the necessary input data is available at the time of the optimization, when the scheduling decisions are made. In contrast, for an online scheduling problem, the decisions have to be made before some of the problem parameters are revealed. In the semi-offline case, some but not all information about the problem parameters is available in advance before making the decisions. In general, it is impossible to provide an approach generating the optimal solution if the problem is not offline. In the online and semi-offline case the proposed methods in the literature can only guarantee a competitive ratio, that is the maximum deviation of the provided solution from the optimal one. For offline problems, the optimal solution is usually theoretically findable. Nevertheless, most of the offline problems are NP-hard, thus finding the optimal solution in a reasonable time is in many cases not possible or at least challenging.

An other dimension for classification is the uncertainty. A scheduling problem is called ***stochastic*** if some of the problem parameters take values only at the time of the execution

of the provided schedule, and the problem is called ***deterministic*** otherwise. For stochastic problems the decisions to be made can be categorized into two groups: the ones that must take values before the stochastic parameters take values, and the ones that can alter their value after that.

The two mentioned categorization aspects are only loosely defined, and thus they also have kind of an overlap. As an example, stochastic problems can be considered semi-offline, as some of the decisions has to be made before knowing the exact values of some problem parameters, and usually some additional information is available, e.g., the probability distribution function of them. Nevertheless, both of the aspects are addressed independently, as the most common attributes of a stochastic and a semi-offline problem are different, as well as the objectives and the developed approaches for solving them.

In this work, mostly deterministic and offline problems are considered if not stated otherwise. Chapter 6 focuses on a class of stochastic scheduling problems. The domain of the problems and the developed approaches is the scheduling of the batch chemical processes although many of the algorithms may be applied for similar problems of other fields. Section 1.1 provides a brief review of the basic scheduling problems in the literature of combinatorial optimization, as some of these problems are the roots for the scheduling problems that appear in the chemical industries. The introduction of the special features of batch scheduling problems is given in Section 1.2.

1.1 Basic scheduling problems

In this section some of the most simple scheduling problems are discussed. Simple refers here to the description of the problem, not the mathematical complexity of solving it. As it will be presented, some of these problems are already very difficult to tackle to begin with.

Scheduling problems in chemical industries usually involve more problem parameters and constraints than the problems presented in this section. These problems, however, are the basis for the more detailed practical problems, and they provide a lower bound for their complexity.

1.1.1 Problem classes

In the literature[101] of these problems, the terms ***job***, ***operation***, and ***machine*** are used instead of products, tasks, and units, respectively. The problem classes are formulated with a triple, $\alpha|\beta|\gamma$, where the

α **field** describes the jobs and the available infrastructure

β **field** provides additional parameters of the problem if any

γ **field** defines the objective

The α field describes the number of machines, whether they are identical or not, and also the type of the jobs to be completed on those units. Without attempting to be comprehensive, some of the possible entries of the α field are given here:

Single stage problems

1 - **Single machine** A single machine is available, each job consists of a single step to be performed on that machine. The products may (and usually do) have different processing times.

Pm - **Identical parallel machines** m identical machines are available, with single step jobs, that can be completed on any of the machines.

Qm - **Parallel machines with different speed** are similar to the Pm case with the difference that each machine has an assigned speed factor.

Rm - **Unrelated machines in parallel** Generalization of the single stage case, when the processing time of a job on a machine is defined as an input parameter.

Shop problems

Fm - **Flow shop** The jobs have m steps that have to be performed on the m machines in the same order for each job.

FFc - **Flexible flow shop** Generalization of the Fm and Pm cases, where each job has to go through c stages in the same order, where several parallel machines are available.

Jm - **Job shop** Generalization of the Fm case, where the jobs can have different order for the machines, moreover, it is not mandatory for a job to visit all of the machines.

FJc - **Flexible job shop** Generalization of the Jm and FFc cases, when the jobs have to go through some of the stages in a job-dependent order, and at each stage, some identical units are available.

Om - **Open shop** Each job has to go through all of the machines. However, their order is not given by the problem definition.

As it is already highlighted in the description, some of the cases are special cases of others, e.g., Pm is a special case of FFc when $c = 1$. These dependencies between the different α field values are shown in Figure 1.1, where an arc leading from A to B represents that B is a special case of A.

Unlike the α and γ fields, the β field that provides additional constraints, may contain several or no entries at all. Some of the most common entries are:

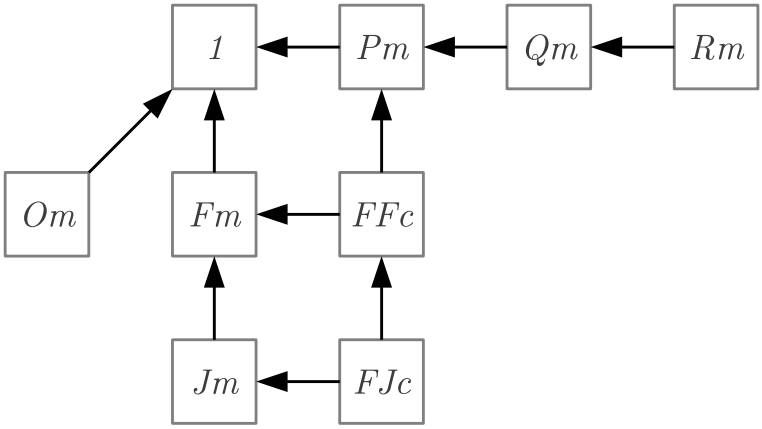


Figure 1.1: Dependency between the possible entries in the α field

r_j **- release dates** For each job, its processing cannot start earlier than its release date.

prmp **- preemptions** The processing of a job on a machine can be interrupted, and continued later on any suitable machine.

s_{jk} **- sequence dependent setup times** After completing job k on a machine, it needs s_{jk} time to be adjusted for the job j.

batch(b) **- batch processing** The machines can perform at most b jobs simultaneously, however, the jobs has to wait for each other when starting or finishing the task.

brkdwn **- breakdowns** The machines are available only in given time intervals.

M_j **- machine eligibility restrictions** In the case of parallel machines, each job has a subset of machines that can be used to process it.

prmu **- permutation** In case of flow shop problems, the order of the jobs at each stage must be the same.

nwt **- no wait** The jobs can not wait between the different machines in case of flow-, job-, or open shop problems.

In the case of the α field, all of the entries were generalizations of the 1 case. Similarly, the empty β field is a special case of many other. From the above list r_j, s_{jk}, *brkdwn*, M_j, and their combination are such entries, however, the preemptions, for example, change the problem in its core.

The γ field defines the objective of the optimization. Similarly to the α field, exactly one entry is allowed (and required) here. The most common entries are:

C_{max} **- makespan** Minimization of the maximum completion time.

$\sum w_jC_j$ **- weighted completion time** Minimization of the total weighted completion time of jobs.

L_{max} **- maximum lateness** Minimizing the maximal violation of due dates. Lateness can take a negative value, when a job is finished earlier than its due date.

$\sum w_jT_j$ **- total weighted tardiness** Minimization of the weighted sum of non-negative lateness values of the jobs.

$\sum w_jU_j$ **- total weighted number of tardy jobs** Minimization of the weighted number of jobs that are not completed within the given deadline.

Similarly to the other two fields, C_{max} is a special case of L_{max} when all the due dates are 0, and $\sum w_jC_j$ is the special case of $\sum w_jT_j$.

Obviously, if a problem class is the same or special case of an other in all of the three fields, its solution can be reduced to the solution of the less restricted one. Since the algorithms for the latter are applicable for the former one, the former can not be more complex than the latter one.

1.1.2 Classification of solutions of scheduling problems

The term ***schedule*** refers to an assignment of each task to units and time intervals in most of the cases. There exist, of course, infinitely many assignment functions for a given scheduling problem, and the term ***solution*** is also often used when referring to a schedule. The schedules that do not satisfy at least one of the constraints of the problem description are called ***infeasible***, and the others are termed ***feasible***. Among the feasible schedules, one is called ***non-delay*** if no unit is kept idle while a task is waiting for processing. It is obvious that any feasible schedule can easily be converted to a non-delay schedule, by shifting each task as early as possible in the solution.

A schedule for a problem is called ***active*** if there exists no other feasible schedule in which some of the tasks start earlier, and none of them starts later. Obviously, all of the active schedules must be non-delay schedules as well.

Active and non-delay schedules play an important role in the case of makespan minimization objective, where the optimal solution or solutions must be non-delay, and at least one of the optimal solutions can be found among the active schedules.

1.1.3 Approaches and complexity

Some classes of the aforementioned problems can be solved to guaranteed optimality by simple, polynomial algorithms. Most of these problems, however, are proven to be NP-hard, thus efficient algorithms are not expected for them.[1]

[1]Note, that there are polynomial algorithms developed for some NP-hard problems, which can provide optimal solution for the *majority* of the instances of the problem class, and a suboptimal solution for the rest.

A simple strategy could be to assign the jobs to the machines based on the increasing order of processing times. This, so-called SPT (Shortest Processing Time) algorithm provides optimal solution for a couple of problem classes if the objective is $\sum C_j$. If the objective is C_{max}, the opposite strategy, i.e., prioritizing jobs with the longest processing time, often provides good competitive ratio for FF problems.

Among the easy to solve problems, the 2-stage flow- and job-shop scheduling problems are probably the closest ones to industrial scheduling problems discussed later. In this case there are two machines for either one or two stage jobs, and the objective is C_{max}, i.e., the minimization of the makespan. Johnson's[61] and Jackson's[58] algorithm provide an optimal solution for the scheduling problems in this classes. However, increasing the number of machines and stages per job makes the problem much more complex, and efficient algorithms are not expected.

Also, in many cases, the problem definition of a real industrial case study involves additional parameters that need to be addressed. In such cases, even offline problems with parallel identical units become NP complete[110].

1.2 Scheduling problems of chemical batch processes

Scheduling problems that arise in the chemical industry are more complex in their description, though they show many similarities to the basic problems described in the previous section.

Batch chemical scheduling problems are mostly given by their

1. recipe
2. storage policy
3. objective, and related parameters
4. additional parameters

These parameters are described in the following subsections in detail, however, some words must be addressed to the assumptions and conventions of this field.

Assumptions

If not stated otherwise, the following assumptions are considered:

Unique allocation A task is assigned to a single unit in the schedule, even if several applicable ones are available.

Non-preemptivity The execution of a task must not be interrupted

Batch processes Each task behaves in a batch-like favor. If a task is continuous, it has sufficiently large dedicated storages to consider it as a batch process.

Fixed batch sizes The amount of material processed in each task is fixed by the production recipe.

Conventions

The notation used among scientist is not standardized. However, several conventions are applied, which may change based on the actual example. To avoid further confusion, the most common notations are summarized here briefly, as they will be used alternately throughout the whole document in problem descriptions, diagrams, etc.

In case studies, products, tasks, units are usually assigned a real name. However, if this is not the case (frequent for literature examples), or the notations need to be shorter, the following conventions are used to label the elements of the system.

Products are often denoted by the first capital letters of the alphabet, i. e., $A, B, C, \ldots$, or by $P1, P2, P3, \ldots$.

Tasks are often labeled after their product, especially in sequential recipes, like $A1$, $A2$, $A3$, ... In some papers, labels like $T1$ or T_1 are applied. In mathematical formulations the dummy index i is used for tasks, thus sometimes labels like $i1$, i_1, or i_{A1} are used.

Units are similarly denoted by $U1, U2, \ldots$, and sometimes labeled as $Eq.1$, $E1$, or E_1 to refer to equipment units. In case of the mathematical formulations, the applied dummy index is j, thus sometimes $j1$ or j_1 is used as well.

1.2.1 Recipes and example problems

The word "recipe" itself is an ambiguous term, as the ISA SP88 standard defines four levels of recipes: general recipe, site recipe, master recipe, and control recipe[52], however, none of these are entirely satisfying our requirements towards a recipe. Thus, thorough the document the term recipe will refer to the collection of the following parameters:

- list of products
- list of tasks, that are to be performed in order to produce the products
- productional precedences among tasks[2]
- available units
- processing times for suitable task-unit pairs

[2]This is often indirectly given by providing the inputs and outputs of tasks.

The recipe of a product refers to the information in the recipe related to that specific product.

Based on the precedence structure among tasks, the recipes are usually categorized into the following groups starting from the simplest one to the most general one. Each class is a specific case of the next one.

Single stage Each product is produced via a single step. Similar to the Pm case, but the available units are not necessarily identical or uniformly applicable.

Simple Multiproduct Similar to an FFc layout, i.e, each product is produced linearly through a fixed number of stages. The difference is the same as in the previous case: the units at a stage are usually not identical and can not perform the same subset of tasks.

General Multiproduct Often referred to as ***Multiproduct recipe***, the generalization to the simple case allows for a production plan to skip several stages.

Multipurpose Unlike in the Multiproduct case, the stages cannot be ordered in such a way along a line that the production of each product goes from left to right. The number of stages and their order is arbitrary. Moreover, a stage may reoccur several times in the same production. This is the most general sequential recipe.

Precedential The tasks in the production of a product are not assigned to stages, and the precedence between them is given apiece. The key difference between the multipurpose and precedential recipes is that the production of a product is not necessarily linear, i.e., there can be junctions in it, but in the case of several prerequisites for a task, all of them must be completed before starting it.[3]

General network This is the most general recipe class, where the tasks are given by their inputs and outputs, which indirectly define their precedences. Here, unlike the Precedential case, the same material may be produced by several alternative tasks making them optional and not mandatory prerequisites. Moreover, cycles may occur in the dependencies.

It is important to note that although some papers have attempted to provide a classification of scheduling problems[32, 93], the terminology is not standardized. Terms like "multiproduct" may refer to (slightly or significantly) different recipe classes in different papers.[4]

The above classification is the proposition of the author, that satisfies the following important criteria: most of the published approaches can unambiguously assigned to one of the defined problem classes.

[3] Obviously, the dependencies must not create a cycle.

[4] These definitions are mostly indirect by the problem classes covered by the presented approach. Many articles state that the proposed approach or formulation solves e.g., multiproduct scheduling problems. However, the problem class covered by these approaches vary for each paper, making the indirect definition of the multipurpuse problem class inconsistent.

Recipe representations and example problems

Single stage recipes are usually simply given by a table, where the rows correspond to jobs, the columns to units, and each cell contains the processing time if the unit is applicable for that product. Kopanos et al.[67] presented such a case study, whose data is given in Table 1.1.[5]

Processing times (h)	$U1$	$U2$	$U3$	$U4$
$P1$	1.538			1.194
$P2$	1.500			0.789
$P3$	1.607			0.818
$P4$			1.564	2.143
$P5$			0.736	1.107
$P6$	5.263			3.200
$P7$	4.865		3.025	3.214
$P8$			1.500	1.440
$P9$			1.869	2.459
$P10$		1.282		
$P11$		3.750		3.000
$P12$		6.796	7.000	5.600
$P13$	11.25			6.716
$P14$	2.632			1.527
$P15$	5.000			2.985

Table 1.1: Example data for a single stage problem

Multiproduct recipes can still be represented by tables. However, in many cases block diagrams are simpler to describe the problem. The problem from Voudouris et al.[126] is presented in Table 1.2 and as a block diagram in Figure 1.2. Note, that the block diagram is only suitable if each task can be performed by exactly one unit at a stage.

Processing times (h)	Stage 1		Stage 2		Stage 3
	$U1$	$U2$	$U3$	$U4$	$U5$
A	7		3		4
B	8			5	3
C		4	6		4
D		6	9		3

Table 1.2: Multiproduct example represented in a table

Multipurpose recipes are difficult to represent in a table, as the order of the stages are arbitrary. If each task can be performed only by a single unit, the block diagram is a suitable choice, as represented in Figure 1.3 for an example by Ferrer-Nadal et al.[31].

[5]The problem presented in the paper contained additional data for changeovers as well.

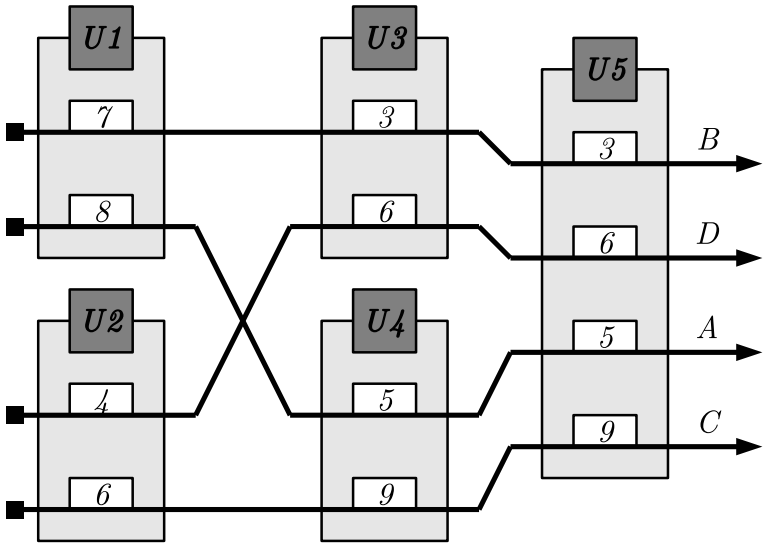


Figure 1.2: Multiproduct example represented in a block diagram

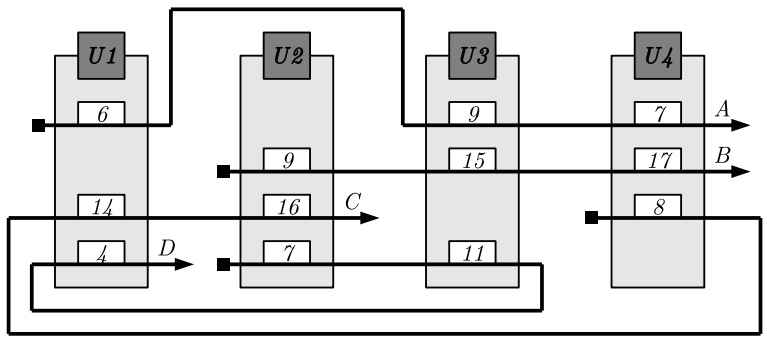


Figure 1.3: Multipurpose example represented in a block diagram

Precedential recipes are usually represented by some kind of graphs, where the tasks are assigned to the vertices, and the arcs represent the productional dependencies. An example by Holczinger[52] is illustrated in Figure 1.4. The processing time and the applicable units are indicated at each vertex, and it is assumed that the processing time is the same if several units are suitable.

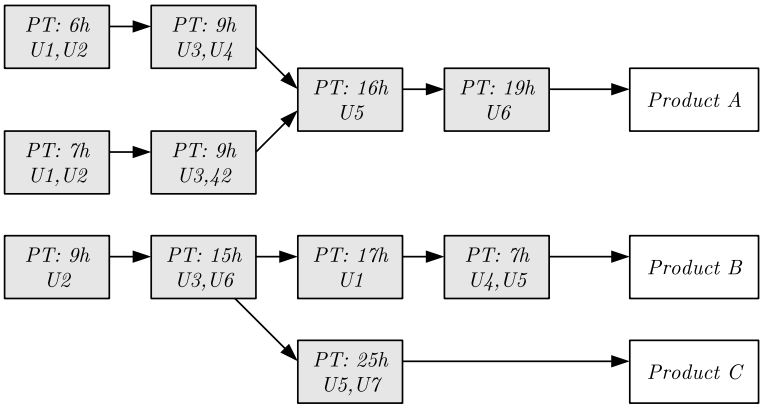


Figure 1.4: Precedential example represented in a graph

General network recipes are most often represented by State-Task-Networks (STN [66]) or Resource-Task-Networks (RTN [100]). The original example for introducing the STN representation from Kondili et al.[66] is given in Figure 1.5. Circle shaped vertices represent materials, while rectangles represents tasks. This graphical representation sometimes does not contain processing times, it is provided in an additional table. The representation is identical to that of the P-graph mathematical model introduced by Friedler et al.[33], that was developed for the optimal design of continuous processes.[34, 36, 35] The key difference in an RTN representation is that units are also represented as resources and tasks are duplicated if several units can perform them. Some papers use the so-called State-Sequence-Network representation[84], where dedicated states (mostly materials) give the vertices, and arcs represent tasks.

1.2.2 Storage policies

The term, ***storage policy*** refers to the constraints for the storage of intermediate products between consecutive tasks of a recipe. For each material there are two factors that can induce constraints on its storage:

1. Chemical and physical properties of the material
2. Infrastructural opportunities for storage in the given plant

Thus, storage policy is a two dimensional property that can differ for all of the intermediates in the recipe. In the majority of the cases, however, the policy is uniform for all of the intermediate materials.

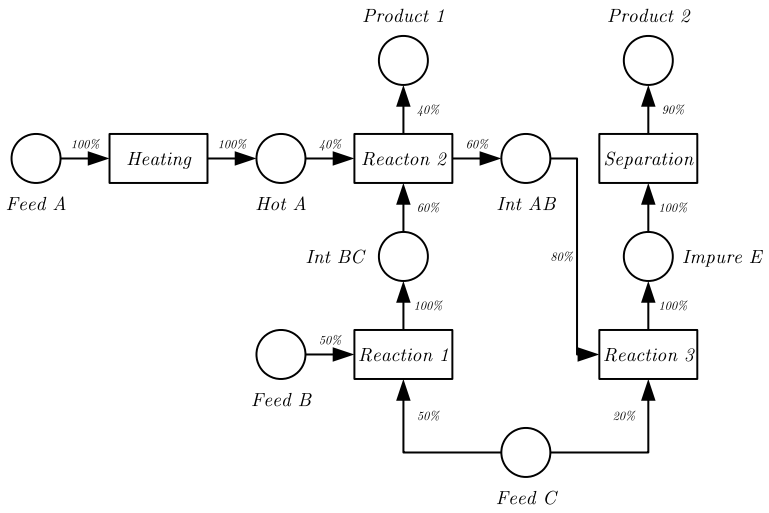


Figure 1.5: STN representation of a general network recipe

The chemical and physical properties of the material provide bounds on the time of the storage, based on which the following categories are usually identified:

UW - Unlimited Wait is the simplest and most common policy, when the material does not lose any of its important chemical or physical properties by time, thus it can wait any amount of time before going to the upcoming task of the production.

LW - Limited Wait storage policy is applied to the intermediate materials that must not wait longer than a certain amount of time before the next task in order not to lose a certain property, e.g., cooling out.

ZW - Zero Wait policy is strictly speaking the special case of the LW policy, when the limit on the storage time equals to 0.

The infrastructural capabilities of the given plant impose limits on the amount of material that can be stored, and on the way of storing it:

UIS - Unlimited Intermediate Storage policy is applied when there is enough storage place available to store any amount of intermediates.

FIS - Finite Intermediate Storage policy is considered when there are storage units available to store the intermediate, but it is limited.

NIS - No Intermediate Storage policy refers to the case when dedicated storage units are not available for the storage of an intermediate material, but it still can wait in the processing unit of the previous task.

In many papers, the storage policy is not defined by a pair, as UW policy is assumed for UIS, FIS, and NIS if not stated otherwise. In the case of ZW policy, the limit for the

storage space is irrelevant, thus it is often used in itself as it is, and this is the reason, why it is not tackled as a special case of LW.

Although, the combination of the above mentioned policies covers most of the cases, there are always practical problems which require special definition. One of the most common examples is the ***Common Intermediate Storage*** policy (CIS), where there is (typically finite) storage available that is shared among several intermediates, and at most one of them can use the storage at a time.

Storage policy is an important parameter of the problem definition, changing the storage policy of the problem can change the optimal solution, the set of feasible solutions, and the applicable approaches as well.

1.2.3 Objective functions

Industry provides a wide range of objectives resulting from practical considerations. However, the two most common objectives are the ***minimization of makespan*** and the ***maximization of throughput***.

Makespan minimization

Makespan minimization is equivalent to the definition of C_{max} in the previous section. In general, the minimal overall processing time is to be found for a given number of batches of each product. In many cases however, the number of batches is not specified in the problem definition, only the amount of products to be produced, thus, the number of batches can change if the recipe allows variable batch sizes. Larger number of batches may lead to better solution, when the smaller units are less loaded, or the processing time depends on the quantity.

Throughput maximization

In case of throughput or profit maximization, the problem description entails a time horizon as well as a certain benefit value for each product that is usually based on mass, profit, and revenue. The goal is to maximize the cumulative benefit of all of the produced products while keeping the production time below the time horizon. The number of batches or amount of products is usually unbounded, it is even allowed to leave out some of the products completely.

Other objectives

Although these two are the basic and most common objectives that provide the basis for comparison of the scheduling approaches, there are plenty of other objectives that arise in real life chemical production. Without attempting to be comprehensive, some of them are:

- minimizing the cost of total earliness-tardiness

- minimizing the clean water use, and in parallel the wastewater effluents (See e.g., [20, 28, 40, 44, 83])
- minimizing heat utility energy or cost (See e.g., [19, 54, 73, 74, 81, 130])
- optimization for a combined cost function that may include utility cost, clean water costs, etc. (See e.g., [43, 38, 80])
- maximizing the overall expected profit
- maximizing cyclic profit (See e.g., [105, 19, 73])
- minimizing the over- or underconsumption of electricity (See, e.g., [94])
- minimizing transportation cost (See e.g., [106])
- etc.

1.2.4 Common additional parameters

Scheduling problems arising in industry often entail further problem specific parameters. Some of them may not play important role in the schedule, thus it is neglected or approximated, e.g., the temperature and cooling of a material, control parameters of the process. In many cases, however, these parameters must be included in the problem definition. Without attempting to be comprehensive, some of the most common additional parameters are:

Cleaning time Some tasks leave a unit contaminated, making it unsuitable for any upcoming task, thus it needs cleaning, which is often comparable with processing times. See e.g., the paint production example by Adonyi et al.[1]. This parameter usually depends on the task-unit pair.

Changeover time This parameter is similar to the cleaning time, which can be considered as a special case. The difference is that a changeover time also relies on the subsequent task that is scheduled for the unit. This happens for example when a unit must be adjusted before undergoing a certain task.[6] See e.g., the examples in the papers by Kopanos et al.[67, 68].

Transfer time Often, the time required for the transfer of intermediates from one unit to an other is not negligible. See e.g., the examples by Grau et al.[41]. This parameter is usually defined for the triple of source unit, material, and destination unit. If the transfer is continuous, both units must be available during the material tansfer. If the transfer is discrete like in the case of the

[6]Setup times are in this sense also a special case of changeover times, as they do not depend on the previous task of a unit.

scheduling of wet-etch stations [5, 7, 6, 13, 12, 18, 37, 62, 95, 129, 128], the units may be used for other purposes during the transfer.[7]

Unit piping If the intermediates are liquid, thus their transport between the units is mostly done via pipes and compressors. Although several applicable units may be available for two subsequent stages of a production, the piping (or better its absence between two units) limits the possible choices. See e.g., Kopanos and Puigjaner [69].

Cleaning the units, transferring materials, etc. obviously entails costs, energy consumption, clean water consumption, wastewater generation, etc., which are often considered as constraints for the schedule as well.

Some papers consider scheduling integrated with process planning[41, 70, 116] or control level decisions[108, 15], and include relevant parameters from that level and a combined objective.

1.2.5 Representation of a schedule

A schedule is basically a set of quadruplets in the form of (i, j, t^s, t^f), where:

i is the task to be performed

j is the unit to perform task i

t^s,t^f are the starting and finishing times respectively

The prevalent graphical visualization of this data is the Gantt chart, see Figure 1.6 as an example for 8 sequential products. The x axis represents time, while the units are listed on the y axis. Each task is represented by a rectangle in the row of the unit performing it, ranging from its starting time to its finishing time horizontally.

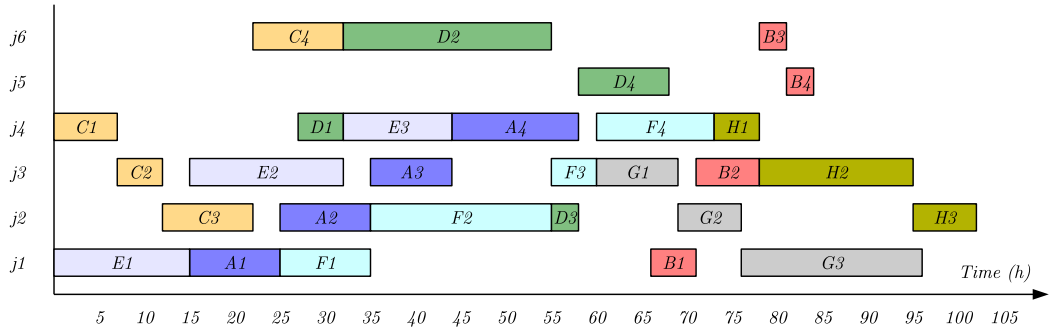


Figure 1.6: Example Gantt chart for 8 products

[7]Note, that in case of wet-etch stations, the robot arm performing the transfer must be scheduled as the other units (baths in this case).

To enhance visibility, the rectangles of the tasks belonging to the same product are usually highlighted with an identical color. In many cases, especially if colors are not applied to differentiate between products, only the name of the product is indicated at the rectangles.

If the problem includes changeover, cleaning, or transfer times, they are represented analogously with rectangles. In Figure 1.7, a schedule for producing PHBA (para-hydroxy benzoic acid) is illustrated. Here, four batches of the same product are produced, thus each batch has its designated color. The recipe is not sequential, while the phenolate reaction is executed in the first or second phenolate reactor, marlotherm is filled into and heated up in two parallel carboxylation reactors. When the phenolate reaction finishes, the intermediate is transferred to the two selected carboxylation reactors, represented by darkened rectangles at all of the three reactors. After the transfer finishes, the phenolate reactors need to be cleaned, which is visualized as a fading rectangle.

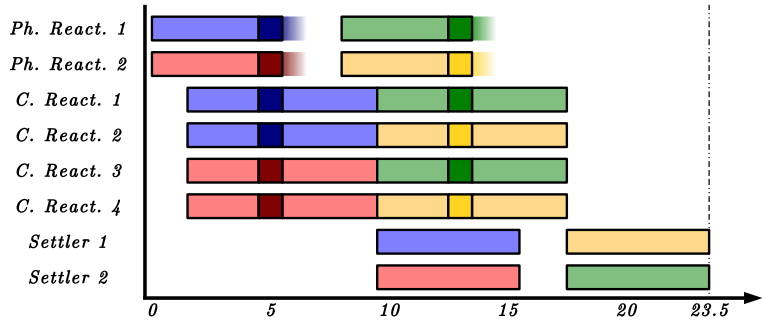


Figure 1.7: Example Gantt chart for non-sequential recipe with transfer and cleaning times

Chapter 2

Mathematical tools for the scheduling of batch chemical processes

In the last two decades, many different approaches have been published in the literature to tackle the problems presented in Section 1.2. They differ in both their domain of solvable problems, and the applied mathematical tools. Over the years, these approaches have gone through vast development, whose goal was dual:

1. broadening the range of solvable problems
2. accelerating the solution procedures in order to bring larger problems to a manageable level

The advancement in both directions is remarkable. The first, heuristics based approaches have considered only simple multiproduct examples, and could not guarantee optimal solutions for even the smaller instances. The first optimization based methods were extended to a wide range of problems. In terms of speed, computational times were reduced by magnitudes, allowing not only the solution of larger problems, but also integration with other control or design aspects.

In order to provide a review of the state-of-the art tools of the field in the next sections, these approaches must be categorized before further discussion. This categorization can be based on various aspects. The most common one is to consider the applied mathematical tool as the major attribute[32, 93, 48].

The majority of the published approaches rely on mathematical programming, namely on Mixed-Integer Linear Programming (MILP) formulations, or Mixed-Integer Non-linear Programming (MINLP) models in rare cases[32, 93]. These models are usually solved by an implementation of general purpose MILP algorithms[127]. In case of MINLP problems, the authors have also developed general purpose MINLP solution procedures.[42] Another branch of research focuses on applying graph theoretical and combinatorial tools and problem specific solution procedures. Among these the S-graph framework[112], the Alternative Graph model[98], and the Timed Priced Automata[99] or Timed Place Petri Net[39] approaches are notable. A small number of papers considers mathematical logic [103] or

Constraint Programming (CP)[128] to tackle scheduling problems. More often, these tools are used combined with other approaches, like MILP formulations[87, 107].

These lastly introduced class of approaches will not be discussed in more detail. The classification of combinatorial approaches based on the exact tool is enough for further discussion. In the case of MILP models, however, further categorization is needed before investigating them closely.

The further classification of MILP formulations can be best explained by the binary variables used to address scheduling decisions. Though the variables may differ in some details for different approaches in the same subclass, these minor differences will be detailed in the subsequent sections.

Precedence based formulations binary variable $X_{i,j,i'}$ represents whether task i' is preceded by task i in unit j.

Time point/slot based formulations has a binary variable $y_{i,j,n}$ denoting whether unit j perform task i at time point n, or in the nth slot.

Start-Stop formulations [1] use variables $Start_{i,j,n}$ and $Stop_{i,jn}$ to represent the starting and finishing of task i in unit j at time point n

Some formulations apply several different techniques redundantly in the hope of a better performance, and thus cannot be categorized unambiguously [64].

Before discussing these approaches in detail, one other classification has to be noted by Hegyhati and Friedler [47], that considers the underlying idea of addressing the problem as the main categorization angle. This aspect has various benefits, the approaches in the same main category

- can usually address the same or very similar set of scheduling problems;
- have similar performance
- can benefit from the development of others, or implement their idea of improvement;
- suffer from the same difficulties and shortcomings

The proposed main categories are:

Time discretization based techniques consist of the time slot and time point based approaches, and the Start-Stop models, which rely on the discretization of the time horizon that can lead to suboptimal or even practically infeasible solutions. On the other hand, they can address a wider range of scheduling problems, though the implementation of sequence dependent attributes are complicated.

[1] There is no commonly used terminology for this set of formulations, the Start-Stop term is used only here.

Precedence based techniques consider the order of tasks assigned to the same unit as the key decision, rather then their exact timing. Precedence based MILP models, the S-graph Framework and the Alternative Graph methods belong here. These approaches often outperform the previously mentioned ones on the class of scheduling problems they can tackle, though this domain is considerably smaller. An additional advantage is the absence of suboptimal or infeasible solutions due modeling errors.

State space based techniques are performing the optimization via a sophisticated exploration of the state space of the system. Although, similarly to the precedence based methods, their model building ensures global optimality, in terms of performance and modeling power they drop behind the aforementioned approaches. The significant advantage of these methods resides in the possibility of integration with the control level, and their straightforward extension to online problems.

2.1 MILP formulations

As it was already discussed, the MILP models play a dominant role in solving batch scheduling problems. In this section the different branches of MILP models are discussed. The related modeling issues are not discussed here, only briefly mentioned, as they will be presented in detail in Chapter 3.

Before the detailed description of different MILP formulation branches, there is an other classification aspect that needs to be mentioned. There exist so-called STN and RTN models for all of the formulation types that will be discussed later. Though STN and RTN are only representation tools, this terminology is widely used to indicate how processing units are tackled in he model. In STN formulations, units are dedicated elements of the model, thus the binary variables usually have an index for units as well, indicating whether a tasks starts, finishes, or precedes an other in a certain unit. On the other hand, RTN formulations consider units as any other resources, e.g., materials, and the tasks "consume" and "release" these resources when their execution starts and finishes. As a result, units do not explicitly appear in the models, only a set of material balance constraints refer to them. Note that in RTN models, a task should be duplicated if it can be performed by several different units.[2] In the following subsections the examples are given for STN models, from whose their RTN counterpart can be derived easily).

[2]Note that even among RTN models, strict and lenient models can be differentiated, which can have a notable impact on the performance as indicated by Eles [30]. If several identical units are available, the lenient RTN models consider them separate resources, implying a sort of redundancy, while strict RTN models consider them as a single resource with higher availability.

2.1.1 Time discretization based formulations

Chronologically these types of formulations were the first ones to appear in the literature [66]. Their underlying idea is to identify several ***time points*** or ***time slots*** over the time horizon. Time slot and time point based approaches [124] show a lot of resemblance, as an interval from a time point to the next one can be considered as a time slot, and vice versa the starting time of a time slot can be considered as a time point. Although time slot based approaches were mainly developed for sequential processes and the time point approaches tackle general network problems, the two classes of approaches are addressed simultaneously in this section. If not indicated otherwise, statements for time point approaches hold for time slot approaches as well.

At each time point, binary variables are assigned to tasks denoting whether the execution of the task is scheduled to start at that time point or not. As a result, the number of binary variables is roughly proportional to the number of time points, i.e., the computational time strongly depends on the number of time points. Thus, it has always been the researchers intention to develop models that can find the optimal solution with minimal number of time points. However, it is not evident, how the sufficient number of time points can be determined for a model and a problem instance. The most commonly applied methodology is to consider a small number of time points first, and perform the optimization. Then the number of time points is increased by one, and the optimization is carried out again. This last step is repeated until the same objective value is found for two consecutive steps. This technique, however, does not guarantee the optimal solution, which is discussed in detail in Section 3.1. Nevertheless, the vast amount of development focused on these approaches achieved a significant reduction in the necessary number of time points. However, the advanced models often became less transparent, the constraints became more complicated and modeling errors occurred.

In general, time discretization tools can address the widest range of scheduling problem with general network recipes, recycling, flexible batch sizes, load dependent processing times, etc. An other advantage of these models is that there is no need to define the number of batches a-priori the optimization, as at each time point the decision is made independently on each task. This feature also allows these models to address several units performing the same tasks in parallel, without any modification.

The following subsections present the key properties and elements of models belonging to a branch of formulations.

Fixed time point models

In the early time discretization models, the time points was equidistantly selected prior to the optimization process, resulting in the so-called ***Discrete-time models***[66]. The used terminology in the literature is misleading in this case, as the later, more advanced approaches also discretize the time, the only difference is whether the placing of the time

points is fixed or not. Thus, the sections here use the more adequate terms ***Fixed time point model*** and ***Variable time point model***.

The unit distance between consecutive time points could be the largest divider among the plausible processing times. The typical binary variable in such a model is W_{ijt} representing that unit j starts performing task i at time point j. Since the processing time is known for i in j, it is known exactly at which time point j will be free again for other tasks. The constraint below is a typical assignment constraint expressing this feature:

$$\sum_{i' \in I_j} \sum_{t'=t}^{t+t_{ij}^{pr}-1} W_{i'jt'} - 1 \leq M \cdot (1 - W_{ijt})$$

Material balance constraints are appropriately stated at each time point. The advantage of these models is, that the above constraint has tight LP relaxation[3] , and there is no need for big M constraints[4] in the model. The regular distribution of time points also makes it simple for example to address FIS-LW storage policy[63]. On the downside, the number of time points and thus the number of binary variables, and the computational need is high. As a result, these models cannot be applied for medium size problems.

Variable time point models

In order to reduce the number of binary variables, the number of time points needed to be decreased. The next step in this development was to make the placing of the time points variable[102]. In the developed models, a continuous variable is assigned to each time point, defining its exact position. The material balance and assignment constraints are similar to the previous models, the key difference lies in the timing of the time points. In order to appropriately constrain the timing difference between the time points, several big M constraints needed to be inserted into the model. Although these constraints have worse LP relaxations, the reduction in the number of binary variables has much higher impact on the CPU needs.

The variable time point based approaches can be categorized based on several aspects:

- if the placing of the time points are the same for all of the units, the approach is called a "global time point", otherwise a "unit-specific time point" based model
- Some of the approaches do not allow tasks to overlap several time points, while others do.

As opposed to unit specific time point models[57, 56], the global time point models[85] may require a larger number of time points to cover the same set of schedules, thus they

[3]By replacing the yet undecided binary variables with $[0, 1]$ continuous ones, the optimal objective value of the resultant LP model is close to that of the source MILP.

[4]Inequalities that become non-constraining for certain values of one or several binary variables, which is done by the product of a sufficiently big number (usually denoted as M, hence the name) and the linear expression of those binary variables. This type of constraints usually have poor LP relaxations.

are often slower. On the other hand, in the case of unit specific time point models, the synchronization between the material flows becomes rather difficult, as the time points of the units are independent of each other. This also gives rise to the possibility of modeling errors, see section 3.3 for more detail.

If the processing time of a task is considerably larger than some of the other tasks, several schedules are not covered by those models, that do not allow time point overlaps for tasks, regardless the number of time points. This issue does not appear for the Start-Stop models. They provide, however, very poor performance results in general. For the more efficient time point based models, the models had to be generalized, as discussed in a little bit more detail in Section 3.3 An other mentionable attempt to tackle this issue used the SSN formulation and introduced additional variables for the storage availability, and usages[115].

2.1.2 Precedence based formulations

The first precedence based MILP formulations appeared around the same time, as the introduction of the S-graph framework, for multiproduct and multipurpose problems. Unlike the previously discussed time discretization based approaches, the precedence based models do not need to discretize the time horizon, and thus they do not use any unknown parameter in their model. Generally, they provide better computational results for the problems they can address. However, this set is much smaller than that of the time discretization based approaches. Although most of the models were introduced for multiproduct or multipurpose recipes, they can be extended to address more general precedential recipes in a straightforward way. Throughput maximization is usually not addressed, as the number of batches is an input parameter of the model.

The key foundation of these formalizations are the two sets of binary variables: $Y_{i,j}$ denoting, whether task i is assigned to unit j, and the sequencing variable $X_{i,j,i'}$ which takes the value of 1, if both tasks i and i' are performed in j, and i is enlisted earlier in the production sequence of j. There is a number of different versions of precedence based formulations based on the exact binary variables and constraints used, but the two main categories are the ***Immediate precedence*** and the ***General precedence*** models. In the former case, the sequencing variable $X_{i,j,i'}$ takes the value of 1 if only if i and i' are consecutive tasks in the production sequence. In general, General precedence models need half as many binary variables (as $X_{i,j,i'}$ and $X_{i',j,i}$ are each others complement if assigned to the same unit), and usually outperform the immediate precedence models, but some features are easier to be expressed by immediate precedence variables. Also, some of the models use both variables redundantly, resulting in hybrid models[67]; many models leave out the index j from the sequencing variable[90]; some formulations introduce additional binary variables to address other features, e.g., additional resources[91].

This results in a wide range of very similar yet different models, with different computational needs.

2.2 Analysis based tools

Petri nets and automata are widely used for the modeling of discrete event systems[16]. There have been several attempts to extend the modeling power of these tools and apply for the scheduling of batch processes. In order to do so, the basic models had to be extended with timing, Timed Place Petri Nets (TPPN) and Timed Priced Automata (TPA) are expressive enough to address most aspects of these scheduling problems. The approaches usually use a B&B algorithm to explore the state space of the system in order to find the most advantageous solution candidate. Due to proper model building, modeling errors are avoided: cross transfer (see Section 3.2) for example, is eliminated as a deadlock situation. Although, these approaches bear the advantage of straight-forward modeling, opportunity to integrate control level decisions, and simple extension to reactive scheduling, the efficiency of these techniques is still behind that of the state of the art MILP models or S-graph algorithms.

Timed Priced Automata

There are several ways to extend the automata with timing. In a so-called time guarded automaton, some additional clocks are responsible for timing considerations[9, 8]. At each transition a timing condition has to be satisfied in order for the transition to happen. After that, some of the clocks may be reseted. Time guards can also appear on states as well. A further extension of this model is the Timed Priced Automaton[11], that has been applied by Panek et al.[99] and Subbiah et al.[122] for batch process scheduling. In these approaches, the recipes and units are usually modeled separately, and the model of the system is generated by applying parallel composition of them. Although the resultant model is usually huge, and difficult to present, its soundness is guaranteed by the mathematical proven model building operation. A general complexity of this approach is that the state of clocks is uncountably infinitely large, and thus the state space of the system also. In order to tackle this issue, the states of the clocks are clustered into so-called clock-regions, and thus, infinitely many states can be described by a single region. The modeling of these regions can be done efficiently by Difference Bound Matrices[27].

Timed Place Petri Net

In a TPPN, the tokens of a transition are generated by a delay, that can present processing times, etc. Ghaeli et al.[39] presented such an approach for the scheduling of batch processes. Some extensions for the modeling expressiveness of this approach were later explored[49]. Soares et al.[120] presented a timed Petri net based approach for the real time scheduling of batch systems.

2.3 S-graph

The S-graph framework was the first published graph theoretic approach[112] to address scheduling problems of batch processes. The framework consists of a directed graph based mathematical model, the S-graph, and the corresponding algorithms[113].

In this section the framework and the basic algorithm is presented in detail, as they provide the fundamental basis for the later chapters. In the end of the section, further developments are briefly introduced.

2.3.1 S-graph representation

The mathematical model of the framework, called the S-graph is a special directed graph for scheduling problems. Note, that unlike the formerly introduced recipe representations, the S-graph is not only a visualization of the recipe, but a mathematical model. In the framework both recipes, partial and complete schedules are represented by S-graphs. In all of these graphs the products and the tasks are represented by vertices, which are usually termed as nodes. Also, if an arc between two tasks is said, it is to be interpreted as the arc between the nodes representing these tasks.

The S-graph without any scheduling decisions is called the ***Recipe graph***, as it describes the recipe itself. An example is shown in Figure 2.1.

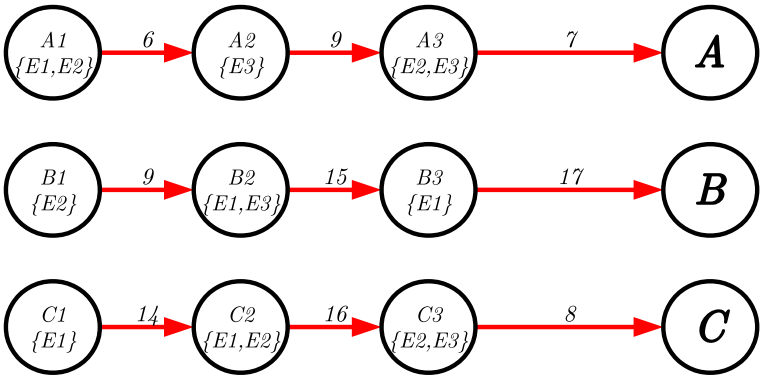


Figure 2.1: Example recipe graph

The three nodes on the right correspond the products, the other nine to the tasks which need to performed in order to produce them.

The arcs, called ***recipe arcs*** between the nodes represent the dependency between either:

- two tasks that depend on each other, i.e., one of them generates the input for the other
- a product and the task producing it

In this example, each product is produced through 3 consecutive steps. In general, the model (and the algorithm from the next chapter) can tackle the set of Precedential recipes, i.e., junctions are allowed. The sets indicated at each task are the sets of plausible units,

and the weight of the recipe arcs are the processing time of the tasks where they start. If a task can be performed with several equipment units, the weight of its recipe-arc (or recipe arcs) is the smallest processing time among all the units suitable to perform it.

All of the S-graph algorithms extend this graph with so-called ***schedule arcs*** that represent the scheduling decisions made by the algorithm. Whether there are still some decisions left or not, the S-graph is called as a ***Schedule graph***. An example is shown in Figure 2.2, where all the decisions are already made and represented by blue schedule arcs.

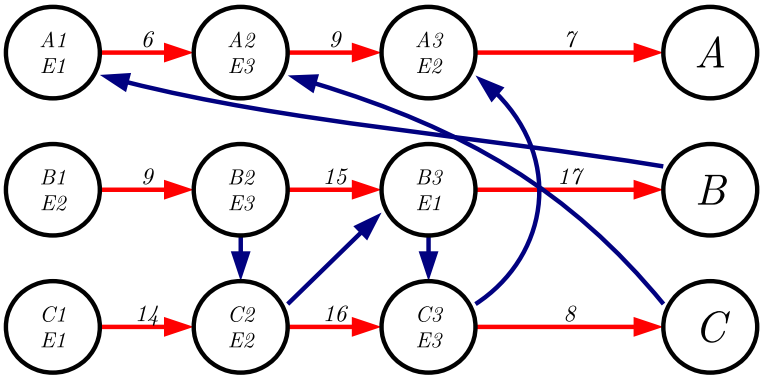


Figure 2.2: Example schedule graph for the recipe graph in Figure 2.1

Note that at each task node, the set is replaced by the selected unit, as this decision has already been made. Also, the weight of schedule arcs is 0 by default, when no changeover-, transfer-, or cleaning times are included in the problem. Modeling of these parameters is simple. It is further discussed in Chapter 7.1. The sequence of tasks assigned to the same unit can easily be exploited from the graph. As an example, the sequence for unit $E2$ is $B1 \rightarrow C2 \rightarrow A3$, as illustrated in Figure 2.3.

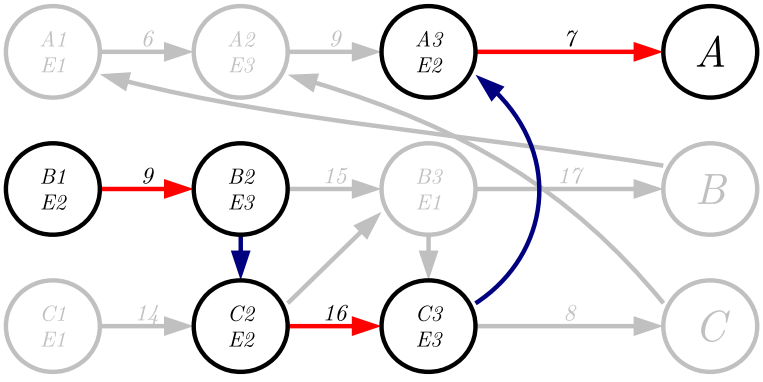


Figure 2.3: Sequence of tasks assigned to unit $E2$ in the schedule represented in Figure 2.2

Note, that the schedule arc corresponding to the decision that $E2$ first performs $B1$ and then $C2$ is expressed by a schedule arc between $B2$ and $C2$, i.e., the schedule arc of the decision does not start from the previous task, but from its subsequent task or tasks. This way, the schedule arc expresses that the unit must not only finish a task before going under

the next one, but also the subsequent task of this task (performed in an other unit) must also take the intermediates.

From the schedule graph, the Gantt chart can easily and unambiguously be generated. The Gantt chart for the schedule in Figure 2.2 is shown in Figure 2.4.

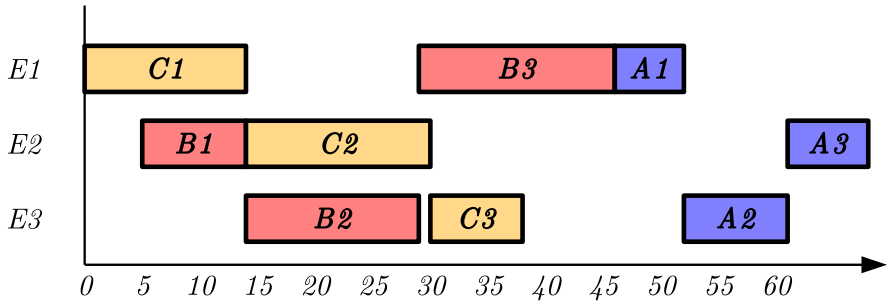


Figure 2.4: Gantt diagram generated from the schedule in Figure 2.2

2.3.2 Algorithm for makespan minimization

The algorithm described here were published by Sanmarti et al.[112] for the minimization of makespan. The pseudo code presented in the algorithm blocks 2.1 is not identical to the one in the original article, though the key aspects are the same. The main differences are:

- The original code addressed problems where each task had only a single plausible unit, i.e., only sequencing decisions were needed no allocational ones. The algorithm presented here extends the original pseudo code to a wider range of scheduling problems, where tasks may have several plausible units.
- The notation is simplified and adjusted to the general conventions.
- The algorithm is no longer divided into a main and branching part.

The algorithm first initializes the value of $makespan^{cb}$ to infinity, and the set $\mathcal{S}$, that will be the set of open subproblems during the optimization. Initially, $\mathcal{S}$ contains only the root problem, i.e., the recipe graph without any assignments made so far. The simple function **recipe** returns the recipe graph for the problem denoted by $G(N, A_1, A_2, w)$, such that:

$N := I \cup P$, the set of nodes

$A_1 := \{(i, i') | i \in I \quad i' \in I_i^+\}$, the set of recipe arcs

$A_2 := \emptyset$, the set of schedule arcs

$w_{i,i'} := \min_{j \in I_j} t_{i,j}^{pr}$, the weights for all recipe arc $(i, i') \in A_1$: the minimal processing time for i

The elements of the set are quadruplets $(G(N, A_1, A_2, w), I', J', \mathcal{A})$ such that

Algorithm 2.1 Makespan minimization with the S-graph framework

$makespan^{cb} := \infty$
$\mathcal{S} := \{(\textbf{recipe}(), I, J, \emptyset)\}$
while $\mathcal{S} \neq \emptyset$ **do**
  $(G(N, A_1, A_2, w), I', J', \mathcal{A}) :=$**select_remove**$(\mathcal{S})$
  if bound$(G) <$ $makespan^{cb}$ **then**
    if $I' = \emptyset$ **then**
      $makespan^{cb} :=$**bound**(G)
      $G^{cb} := G$
      $\mathcal{A}^{cb} := \mathcal{A}$
    else
      $j :=$**select**(J')
      for all $i \in I_j \cap I'$ **do**
        $G^i(N, A_1, A_2^i, w^i) := G(N, A_1, A_2, w)$
        for all $i' \in \bigcup_{(i',j)\in\mathcal{A}} I_{i'}^+ \setminus \{i\}$ **do**
          $A_2^i := A_2^i \cup \{(i', i)\}$
        end for
        for all $i' \in I_i^+$ **do**
          $w_{i,i'}^i := t_{i,j}^{pr}$
        end for
        $\mathcal{S} := \mathcal{S} \cup (G^i(N, A_1, A_2^i, w^i), I' \setminus \{i\}, J', \mathcal{A} \cup \{(i, j)\})$
      end for
      if $I' \subseteq \bigcup_{j'\in J', j\neq j'} I_{j'}$ **then**
        $\mathcal{S} := \mathcal{S} \cup (G(N, A_1, A_2), I', J' \setminus \{j\}, \mathcal{A})$
      end if
    end if
  end if
end while
if $makespan^{cb} \neq \infty$ **then**
  return $(G^{cb}, \mathcal{A}^{cb})$
end if

$G(N, A_1, A_2, w)$ is a schedule graph

I' is the set of unscheduled tasks

J' is the set of units to which the algorithm can still assign tasks

$\mathcal{A}$ set of task-unit assignments in the form of (i, j) pairs

In each iteration a subproblem is arbitrary selected and removed from $\mathcal{S}$ by the function **select_remove**. The exact behavior of this function may be different for different implementations, resulting in various search strategies.

At the beginning of the iteration, it is evaluated, whether the subproblem has the potential to provide an optimal solution ot not. This is done by the **bound** function, towards which the following requirements hold:

- it should provide a lower bound for the solutions that can be derived from the subproblem
- it should provide the exact makespan of leaf problems, i.e., for completely scheduled graphs
- it should return infinity if the graph contains a cycle, indicating that it is unfeasible

The mostly used bound function is the longest path in the graph, but LP based models can also be used, see Holczinger [52] for details. If the bound of the subproblem is not smaller than the best solution found so far, the iteration ends, and an other subproblem is selected (if exists).

If the bound is smaller than the value of $makespan^{cb}$, the algorithm first checks, whether all of the tasks are already scheduled, i.e., whether the subproblem is completely scheduled. If this is the case, the values of G^{cb}, $\mathcal{A}$, and $makespan^{cb}$ are updated to the S-graph of the best solution, the corresponding assignments, and the value of its makespan, respectively.

In the case of a partially scheduled subproblem, the algorithm selects an available unit (i.e., one from J') using the **select** function. Similarly to **select_remove**, the implementation of this function may also be different to achieve various search strategies.

For the selected unit j, the algorithm assigns all the possible tasks ($i \in I_j \cap I'$ to the end of its processing queue.[5] For each assigned task a copy is made of the current S-graph, or more precisely about the set of schedule arcs and the weights, as the set of nodes and recipe arcs do not change during the optimization. This copy is first extended with the schedule arcs induced by the new assignment, i.e., arcs from all the subsequent tasks of previously assigned tasks to j are directed to i.[6] Then, the weight of all of the recipe arcs from i are

[5]Note, that if no such task exist, the algorithm simply skips this loop. Avoiding this situation is not necessary because of the reduction of the J'.

[6]i itself is excluded to avoid loops in case of assigning two subsequent tasks to the same unit. Moreover, in the original algorithm, the arcs were directed only from the last assignment if it existed. Here all the assignments are stored in $\mathcal{A}$, which makes the description of the algorithm simpler. Although the additional

updated to $t^{pr}_{i,j}$. Finally, a new subproblem is added to $\mathcal{S}$ with the modified graph, a reduced I' set, and extended $\mathcal{A}$ set.

If all of the unscheduled tasks can be performed by other available units, a new subproblem is created where j becomes unavailable for further assignments. This part is needed to allow other units to perform the same tasks that j can, and to tackle the situation, when j has no more compatible tasks among the unscheduled ones.

After the set $\mathcal{S}$ becomes empty, the graph G^{db} and the assignments in $\mathcal{A}^{cb}$ describe the optimal solution, and are returned by the function if at least one feasible solution has been found. Otherwise the algorithm does not return with any solution.

2.3.3 Extensions and developments of the S-graph framework

During the years, many extensions and developments of the S-graph framework has been done and published [46]. The algorithm presented in the previous subsection is often referred to as the ***Equipment based algorithm***, as it selects a unit, and branches based on which task should be the next in its production queue. A Branch and Bound algorithm based on a different aspect was presented by Adonyi [3], where a task is selected at each subproblem, and the branching is based on finding an appropriate unit, and an appropriate place in its queue. This method is often referred to as the ***Task based algorithm***. The performance of the two algorithms were compared via an extensive empirical analysis. As a result, it was stated that there are problem instances for both of them where they outperform the other one. However, it is not evident based on the problem description, which is the favorable one, although, problems with a "bottleneck" usually prefer the equipment based approach.

In many cases, the problem instances include the repetitive production of several batches of the same product. This is usually addressed by copying the recipe of the product multiple times. With this approach, however, the same solution can be found multiple times with different order of the identical products. To avoid this, and reduce the computational need, Holczinger et al.[51] introduced auxiliary arcs in the S-graph ensuring that each solutions is found at most only once. With this modification, the authors achieved a tremendous reduction in the CPU time. Hegyhati and Friedler[49] has shown that the same effect can be achieved for precedence based formulations by adding constraints equivalent to these auxiliary arcs.

Next to these algorithmic developments, the framework has been extended to various fields of application, where some modifications of the original framework was also needed:

Paint production Adonyi et al.[1] has applied the framework for the scheduling of a large scale paint production plant, where cleaning times had to be addressed as well.

arcs (compared the the original algorithm) are redundant, they express valid relations. Moreover, the current implementation also includes these arcs (although in an earlier stage) in order to sharpen the bound function.

Heat integration One of the most developed field of continuous plant design is the design of the heat exchanger network, to minimize utility cost. Although, the Pinch technology[65, 77, 78] can provide a decent targets for continuous systems, in case of batch processes the timing of streams must be considered as well. Adonyi et al.[2] presented an extension of the S-graph framework, where heat integration specific constraints were addressed with an LP model that is maintained thorough the optimization process in sync with the schedule graph. Holczinger et al.[54] further extended this approach to address the scheduling of heat exchangers as well.

Train scheduling Adonyiet al.[4] has extended the framework to address the scheduling problem of the supply trains of tunnel boring machines, and developed a graphical interface to convert these problems for the S-graph solver.

An often exploited advantage of the specialized solution algorithm is the possibility to generate several different solution candidates, which can be really useful in practice, if some parameters of the original problem can not be included in the mathematical model. In case of a single objective, the n best solution can be easily generated, and with multiple objectives, the Pareto border can also simply be maintained (assuming a discrete search space). This framework also gives chance for an accelerated bi-objective B&B solver, that has been developed for other discrete problems[121, 26].

Implementational techniques

The outstanding computational performance of the S-graph framework can only partially be credited to the structure of the branch and bound algorithms, and the algorithmic accelerations. The other half belongs to the implementational techniques and accelerations of the solver written in C++, that are - with few exceptions - not presented in the scientific literature.

As the mathematical model of the S-graph framework is not a general model, like the linear programming models, the solution algorithms are developed by the researchers. This carries some benefits and shortcomings as well:

speed The implementation is adjusted to the model, and optimized, thus the CPU requirements are reduced, and thus the S-graph approaches are competitors of not only the free MILP solvers, but the commercial ones as well.

flexibility At each extension, the researchers has the chance to implement their ideas in a low level of the algorithm, not in the model level.

learning curve These changes, however, require an extensive knowledge not only about the S-graph framework, but the software implementation as well.

The only paper about the S-graph solver implementation of was published by Smidla and Heckl[119], where the parallel Branch and Bound implementation is briefly introduced. The authors provided a scalable implementation, which in some cases could achieve super-linearity.

At many points of the algorithm the longest path between two vertices is needed (by the bound function, and later extensions). Although there exists a linear time algorithm to the evaluation of the longest path between two vertices [23], the memory requirement of the S-graph solvers is negligible (compared to commercial MILP solvers), thus it is beneficial to store a matrix of longest paths for each subproblem, and keep it updated.

If a task can be performed only by a single unit, or by several units, but only one of them is in the available unit set at a certain subproblem, a schedule arc can already be inserted from the subsequent tasks of the already assigned tasks of that unit to this task. It is not sure that this task will be the next task in the production queue of the corresponding unit, but it will definitely be assigned to it later. This small technique can drastically sharpen the bounds for problems with tasks that can be performed only by a dedicated unit.

Similarly to the problem investigated by Holczinger et al.[51], the same sitution can occur, when identical units are available, i.e., the same schedule can be generated multiple times. To avoid this, a simple condition is to be inserted to the algorithm which ensures that if two units are identical, the one with a smaller id has a first task on its queue with the smaller id.

In many cases, the difference between a subproblem, and its children problems is small compared to the size of the subproblem. To save time on copying these objects, containers using implicit sharing are applied.

Memory handling has also a crucial impact on the CPU requirements. In his diploma work, Kovacs [71] details and analyses the effect of different malloc implementations. This work details some other enhancements of the solver as well.

Relation to Precedence bases MILP models

Without going into formal details, the aim of this subsection is to reveal the strong connection between the S-graph framework and the Precedence based MILP formulations. This type of investigation is not unprecedented in the literature, Uma et al.[125] investigated the relation between the relaxation of linear formulations and graph models of scheduling problems, while Maraveilas[86] investigated the combinatorial structure of fixed time point MILP formulations. As it has already been mentioned, in case of the classification from Hegyháti and Friedler [49], the precedence based formulations and the S-graph framework belong to the same category. Both approaches consider the sequence of tasks assigned to the same unit as the key question during the optimization. As a result, the capabilities, performance, and search space are quite similar for them.

In order to illustrate the strong connection between the two approaches, a partial schedule of the example in Figure 2.1 is given in Figure 2.5. The figure represents the partial schedule

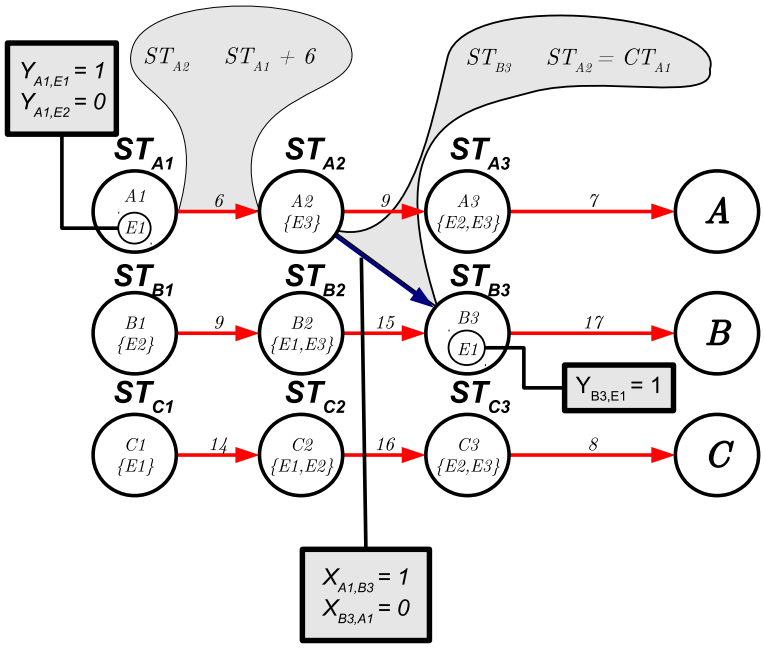


Figure 2.5: Illustration of the connection between precedence based models and the S-graph framework on a partial schedule of the example given in 2.1

after two steps in the S-graph based algorithm. First, the task $A1$ is assigned to the unit $E1$, then the task $B3$ is assigned as the second task for the same unit. The sequencing between them is expressed by the arc between $A2$ and $B3$.

As illustrated in the figure, the same partial schedule in a precedence based model would mean, that some of the binary variables are fixed to certain values:

- $Y_{A1,E1}$ takes the value of one, because of the first assignment, and as a consequence, the value of $Y_{A1,E2}$ is set to zero.
- Similarly, at the second step, $Y_{B3,E2}$ takes the value of one.
- The second step, however, also decides the sequencing between $A1$ and $B3$, thus $X_{A1,B3}$ is set to one, and $X_{B3,A1}$ is set to zero.

If the values of these variables are substituted into some of the constraints, they result in the following form:

- $ST_{A2} \geq ST_{A1} + 6$ is a result of the recipe sequencing constraint with $Y_{A1,E1} = 1$ and $Y_{A1,E2} = 0$. This constraint is expressed directly by the updated recipe arc between the nodes of $A1$ and $A2$.
- $ST_{B3} \geq ST_{A2} = CT_{A1}$ is the result of a sequencing constraint with $X_{A1,B3} = 1$ and $X_{B3,A1} = 0$.

In general, the following objects of the S-graph framework and the precedence based models relate to each other:

Precedence based MILP model		S-graph framework
Continuous timing variables	-	Nodes
Binary allocation variables	-	Assignment at nodes
Binary sequencing variables and constraints	-	Schedule arcs
Recipe constraints	-	Recipe arcs
Model infeasibility	-	Cycle in the S-graph

The connections above explain why these methodologies have similar features. Though the search space is the same, the way how the S-graph explores it is rather different. In each branching step the S-graph based algorithm decides all of the assignment variables at a node, and several precedence variables as well. Thus, the branching tree is much smaller. On the other hand, at each subproblem, the S-graph algorithm uses the Longest path as a bound, which is weaker than the optimal solutions of the corresponding relaxed LP model. These observations give rise to many opportunities for the integration of these two type of approaches, e.g., a precedence based model can be maintained in the S-graph algorithm for providing sharper bounds, or addressing continuous decisions, that are difficult to implement with graphs.

Chapter 3

Critical modeling issues

As it has already been mentioned, most of the approaches, especially the MILP formulations do not go under mathematical validation. The common practice is that the model is developed based on a new idea, and the validation is only empirical, i.e., the implementation of the model is compared on several examples with previous models from the literature. If the result is the same on all of the tested examples, that approach is considered to be accurate.

For the earlier, in a way simpler models, the equations were mostly straight-forward, and even though, no theoretical proof was attached. Readers could accept with confidence that the result will be correct. As the formulations developed, however, they became more and more complicated. In order to reduce the computational time, the scheduling problems were tackled from different and unusual "angles", which brought great success on one side, the side of performance. On the other hand, the implementation of the same constraints become more complicated and "tricky". The constraints are no longer straight forward, which is not a problem itself, however, the validity of the model became "less convincing". The situation is even worse, when the problem definition for an approach is incomplete, i.e., it is not unambiguously defined, what kind of problems are addressable with the approach, what kind of assumptions are made towards the parameters or the behavior of the system[1].

This fear is not unsubstantiated, as several modeling issues were already unveiled in the literature, and there is nothing, which would suggest, that there are no undiscovered ones. Before introducing some of these issues in detail, the nature of modeling mistakes must be investigated first. All of the approaches discussed in Chapter 2 are based on the examination of a search space, and finding the best candidate among them. From the mathematical point of view, these approaches can have two deficiencies, assuming that the ranking of solution candidates is correct[2]: under- and over-constraining. To give the accurate definition of these, some additional terms should be introduced first, which will be used throughout the whole document:

[1] Part of this roots back to the problem of the lack of standardized definitions for problem classes, see Section 1.2.

[2] In most of the cases, this holds. There are, however, examples, when the objective value of a candidate is not evaluated correctly[6]

solution is a term used for a schedule, and in the same time for its representation in a mathematical model if it exist.

practically feasible solution is a solution that can be executed in real life.

practically infeasible solution is a solution that violates some of the constraints of real life, thus it cannot be executed.

practically optimal solution is the best solution among the practically feasible ones.

model-feasible solution is a term used with respect to a model or approach for describing a solution that is plausible for that approach, i.e., it is in its feasible region.

model-infeasible solution is also defined for a model for those solutions that are not in its feasible region.

model-optimal solution is the best model-feasible solution

If not stated otherwise, the terms feasible and infeasible will refer to practically feasible and practically infeasible solutions, respectively.

One would assume, that in case of a proper model, the set of the model-feasible solutions is exactly the same as the set of the (practically) feasible ones. Moreover, if an approach or model has model-feasible / model-infeasible solutions which are (practically) infeasible / feasible, then it is fundamentally wrong. These approaches, however, are not used for generating all of the feasible solutions, only to provide at least one optimal, thus the requirements from the previous sentence are unnecessarily limiting.

Having infeasible solutions in the feasible region of a model is a common practice in optimization to enhance performance, and it does not result in improper results as long as it is guaranteed that the objective value of the practically infeasible solutions will not get better than the optimal value. As an example, the integer variables of MILP problems, whose matrices satisfy the requirements of total unimodularity (assignment problem for example) can be relaxed to continuous variables, and the optimal solution is ensured to be integer[127]. This relaxation significantly reduces the computational need obviously.

Similarly, reducing the search space, and excluding many practically feasible solutions is acceptable if it is ensured that at least one optimal solution remains in the feasible region of the model. Holczinger et al.[51] has improved the efficiency of equipment-based branch-and-bound algorithm of the S-graph framework by magnitudes for problems with high batch numbers using exactly this idea.

Thus, in order for a model to malfunction, it has to fail in at least one of the following ways for some problem instances (not necessarily for all of them):

Under-constraining: At least one practically infeasible solution is model-feasible and it has better objective value than the optimal, i.e., the model-optimal solution is practically infeasible.

Over-constraining: All of the practically feasible and optimal solutions are model-infeasible, i.e., the model-optimal solution is practically suboptimal.

Unfortunately, both type of mistakes has appeared in the literature of batch process scheduling, and in many cases the model bared both type of issues.[3] The following sections will introduce and investigate this kind of issues.

3.1 Minimal sufficient number of time points

As it was briefly discussed in Section 2.1, variable time point based approaches suffer from a common problem[4], that is finding a number of time points that is sufficiently large for the model to contain the optimal solution. For this section, it is assumed that there is a number of time points, where the model has at least one optimal solution in its feasible region.[5]

It is clear, that this is an over-constraining issue, as the practically optimal solution is not in the feasible region of the model. To investigate this issue, first the problem of Voudouris et al.[126] - represented in Figure 1.2 - is examined with batch numbers $2-1-1-1$.

Three models are compared:

M&G a global variable time point formulation by Maravelias and Grossmann[85]

I&F the first unit specific variable time point formulation by Ierapetritou and Floudas[57]

S&F further development of the *I&F* model by Shaik and Floudas[118][6]

For all of the three models the iterative method were applied, i.e., the number of time points is increased until the same objective value repeats. Table 3.1 contains the best model-optimal solutions for each investigated case.

In this particular case, the makespan of 31 h is optimal, as it has been confirmed by other approaches that do not discretize the time. However, the number of time points needed to find this solution is varying for the different models, so it is clear to see, that finding the optimal number of time points (i.e., the minimal number of time points, with which the model results in the optimal solution) is not trivial.

[3] Although there are models which provide infeasible solutions for some instances and suboptimal ones for others, there has not been an instance published, where a model would provide a suboptimal solution which is infeasible at the same time. Constructing an example like that, however, would not be a challenging task.

[4] Fixed time point formulations do not have this issue with throughput maximization problems, as the number of time points is given by definition in that case. For makespan minimization, the incremental increase of time points will obviously end up at the optimal solution, as the number of time points correlates unambiguously to the length of the production.

[5] It is not always the case as it will be discussed in Subsection 3.3.1

[6] The Δ parameter is set to 0.

n	$M\&G$	$I\&F$	$S\&F$
6	×	×	×
7	×	×	31
8	32	33	31
9	31	32	
10	31	31	
11		31	

Table 3.1: Illustration of the iterative approach for the time point MILP formulations (n stands for the number of time points, × indicates that the solver has not found any model feasible solution)

Moreover, the common iterative approach can not ensure that the reported solution is the globally optimal one. This issue is illustrated via a simple single stage example of three products and three available units. The problem data is given in Table 3.2

	u1	u2	u3
$P1$	15 h	7 h	7 h
$P2$	11 h	14 h	9 h
$P3$	14 h	5 h	

Table 3.2: Processing times for the single stage example

The instance of producing 1, 4, and 5 batches of products $P1$, $P2$, and $P3$ is solved by the time slot model of Sundaramoorthy and Karimi [123]. The number of slots has been increased from 1 to 8, however, the model has no feasible solutions for less then 4 time points. Table 3.3 contains the objective value of the model optimal solutions for 4 to 8 time slots .

Number of slots	Optimal makespan
4	34 h
5	27 h
6	26 h
7	26 h
8	25 h

Table 3.3: Illustration of the time point issue

As it is shown in the table, the iterative approach would stop at 7 time slots with the objective value of 26 hours, although by further increasing the number of time points a solutions with 25 hours could have been found. This counterexample proves that the iterative approach can not guarantee the optimal solution.

Several papers tried to address this issue in the literature[114, 75]. However, the approaches published in these papers has the same flaw, as the original iterative approach:

their soundness is not proved, and counterexamples can actually be constructed in a similar fashion. They may provide better solutions than the simple iterative approach. However, the original problem remains: when applying flexible time point based methods, the optimality of the provided solution can not be guaranteed.

3.2 Cross transfer

The root for this issue lies in the way the time needed for the transfer of intermediate materials is addressed in case of an NIS policy. In many cases, the transfer time is negligible compared to processing times, thus it is completely left out from the problem data, and the model considers it instantaneous. Even if the processing times are comparable, they are often lumped to the processing time of the previous task, and the same model is used.

In this section the first case is investigated, i.e., when small transfer times are not considered in the model, though the second approach can result in a similar error. Without doubt, the exact solution provided by the approach will not be applicable in practice, as the transfer of the intermediate will take time, and this would definitely shift the execution of tasks a little. This in itself is, however, the most natural thing when optimizing real life systems that are too complicated to be modeled with 100% accuracy. In general, the biggest task of modeling is to identify the parameters and rules of the system that are important to consider in the model, and the ones that would just make the approach more complicated, without resulting in a significant change for the provided solution. Thus, the exact solution of the approach is not expected to be implementable in practice as it is but it is expected to remain feasible when the additional parameters are put back to the solution.

Unfortunately, all of the MILP based approaches published in literature can provide solutions for some problem instances that are impossible to implement in practice. This under-constraining issue has been investigated independently by Hegyhati et al.[45] and Ferrer-Nadal et al.[31].

The malfunction of the MILP approaches can be illustrated on the simplest example of two products with two stage sequential recipes, as illustrated in Figure 3.1.

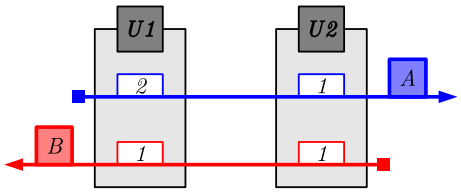


Figure 3.1: Simple example recipe for illustrating the Cross Transfer issue

The problem can bee solved for 1 batch of each product with different approaches. The reported solutions would differ based on the selected approach. Figure 3.2 shows the Gantt chart of the solutions provided by *a*) the S-graph framework and the state space based techniques, and *b*) any of the MILP formulations.

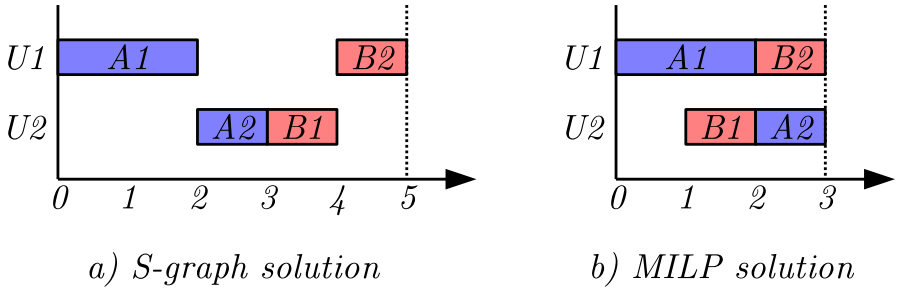


Figure 3.2: Provided Gantt charts for the example in Figure 3.1

As the solutions differ, either solution a) must be suboptimal or solution b) must be infeasible (or both). It is easy to verify that the solution provided by the combinatorial approaches is in fact the optimal solution, and the one provided by the MILP formulations is practically infeasible. In the latter solution, 2 hours after starting the production, units $U1$ and $U2$ should simultaneously exchange the intermediate materials of the production of products A and B. This is obviously not implementable without a temporary storage unit, even if the transfer of the materials can be carried out in a negligible amount of time.

This phenomenon can appear between any number of units, and also occurs on real life examples as well. As discussed by Hegyhati et al.[45], the same infeasible solution has repeatedly been published by Kim et al.[64] and Mendez and Cerda[90] for the problem shown in Figure 3.3.

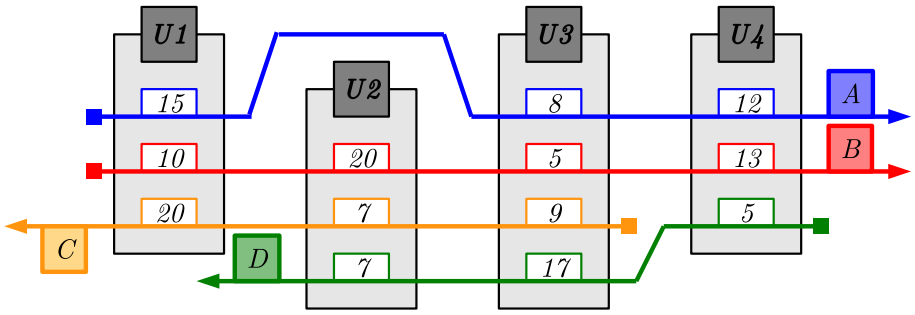
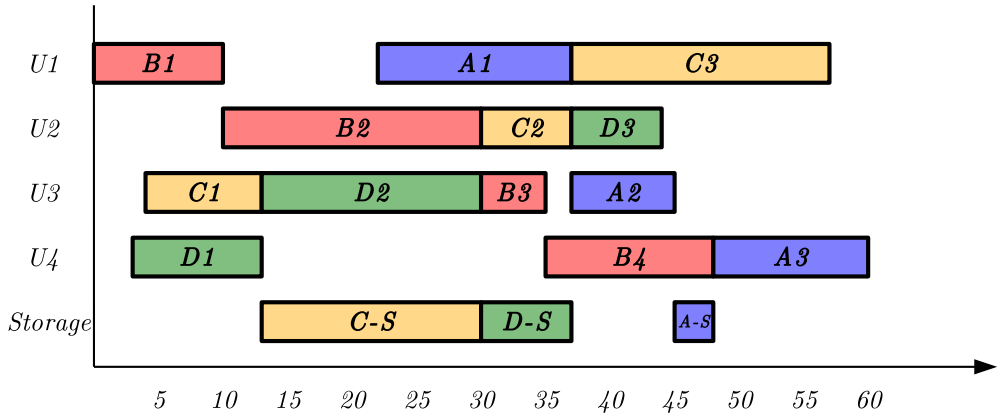


Figure 3.3: Literature example for cross-transfer illustration

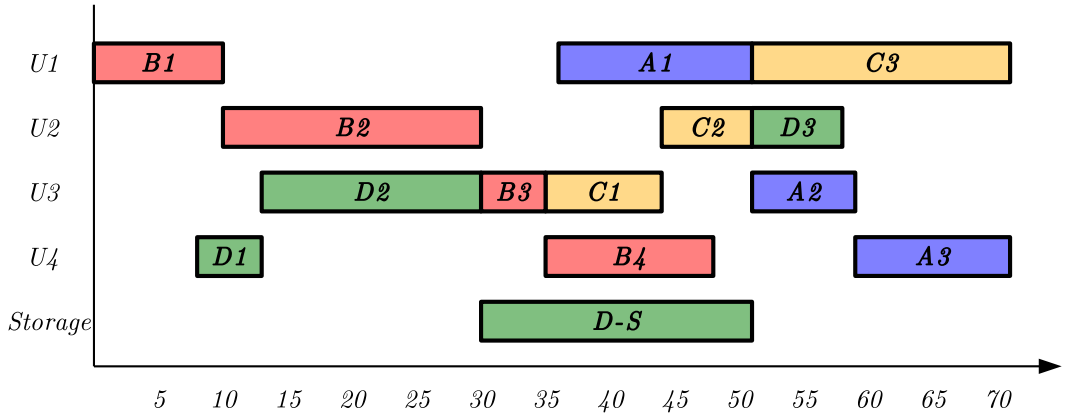
The Gantt chart of the solution provided by the MILP formulations is shown in Figure 3.4 - a). At 30 hours of the production, three units, $U2, U3$ and the S should exchange materials simultaneously. The real practically optimal solution provided by the S-graph framework is shown in Figure 3.4 - b).

In this example, $U4$ is free at the time of the cross transfer. In the unlikely case that this unit can store at least one of the intermediates of B, C, or D, the schedule can be executed. Gouws and Majozi[40] have investigated the benefits of using inherent storage, however, their model also do not tackle this issue.

Further examples from Ferrer-Nadal et al.[31], or other examples could be mentioned to further demonstrate that this issue affects not only theoretical problems, but real life



a) MILP solution



b) S-graph solution

Figure 3.4: Solutions provided for the literature example of cross-transfer illustration

examples as well. Solutions containing cross transfer usually have a denser schedule, thus shorter makespan, making them favorable for the objective. As a result, solutions with cross transfer are likely to be reported for any multipurpose or more complicated example.

Without doubt, the solution provided by the MILP formulations is improper. However it is not obvious where the mistake was made. In defense of the MILP formulations, they would have provided the correct solution if the transfer time were not have been neglected. The reason behind this can be illustrated via some of the mathematical constraints for the infeasible schedule of the above example:[7]

$$\begin{aligned} T^s_{B2} &= T^f_{B1} + t^{tr}_{Int_B} \\ T^s_{A2} &= T^f_{A1} + t^{tr}_{Int_A} \\ T^s_{B2} - t^{tr}_{Int_B} &\geq t^f_{A1} + T^{tr}_{Int_A} \\ T^s_{A2} - t^{tr}_{Int_A} &\geq t^f_{B1} + T^{tr}_{Int_B} \end{aligned}$$

The first two equations refer to the recipes, the second task of each product will start exactly the transfer time later than the finishing of the first.[8] The inequalities describe that a unit cannot start the transfer of the intermediate for the upcoming task until the transfer of the intermediate product of the previous task is finished. It is easy to see that there is no solution to this system, if the t^{tr} parameters are positive, thus the MILP solver would purge this candidate from the B&B tree and find the truly optimal solution.[9] If, however, the t^{tr} values are 0, the model finds the trivial solution, where all of the mentioned variables are equal, leading to the infeasible solution.

The result of the previous examination suggests that the mistake is not done by the MILP formulations, but it is rather the fault of the modeling process when transfer times were neglected. On the other hand, it has been stated earlier that such simplification is acceptable during a modeling process, and other approaches did not fail in finding the real optimal solution.

From the practical point of view, it is irrelevant where the mistake was made during the modeling process. The only important thing is that using the MILP formulations with the common modeling routines may end up with infeasible solutions.

From the theoretical point of view, to make the decision about where the mistake was taken, it has to be noted that the published papers have never restricted their approach for positive transfer times. Even the problems on which the formulations were illustrated had 0 transfer times, indicating that the formulation was supposed to tackle this problems appropriately as well.

[7] i.e., these are some of the constraints that remain in the model after substituting values for the binary variables according to the given schedule.

[8] Equations are needed, as NIS policy is considered. The T^f variables here refer for the ending of the storage of the intermediates in the previous unit after its execution.

[9] Other MILP solution techniques would obviously also avoid this solution.

What happens is that the transfers of materials induce logical constraints, namely: a transfer to a unit cannot start when it still has the intermediate product of the previous task. When the transfer times are positive, this logical constraint is "less constraining" then the timing constraints of the model, thus it has been neglected. I.e., some constraints became redundant for the seemingly more difficult problem, thus - for the reason of simplification and performance - they were removed. The mistake was made when it has not been realized that these logical constraints are no longer covered by the timing constraints in the "simpler" cases.

These logical constraints could be implemented via binary variables, which represents the order of material transfers. This would, however, include a huge number of binary variables in addition (and thus increase the computational need enormously), if the variables are not present somehow already in the model. Ferrer-Nadal et al.[31] proposed an algorithm for generating additional constraints for their precedence based model to avoid cross-transfer, as discussed a bit later.

This logical constraint could maybe also be expressed something like $T^f_{A1} > T^f_{B1} + t^{tr}_{Int_B}$ as well. However, MILP models cannot accept strict inequalities. One way to overcome this issue is introducing small values, as Ferrer-Nadal et al.[31] suggest in their paper: if no transfer times are included in the problem description, a small value should be introduced instead of 0, and later removed when the "optimal" schedule is obtained. Although this approach can overcome this issue in many occurrences, it has its shortcomings:

- it may not be applied for problems with ZW or LW storage policies, as it can render otherwise feasible schedules infeasible
- if the introduced value is too small, it may cause numerical errors for the MILP solver
- if the introduced value is too big, the approach may end up at a suboptimal solution

Before ending this section, a few words must be said about the approaches that avoid this issue. State-space techniques do not find this solution, as the cross-transfer appears as a deadlock while exploring the search space, and the optimization continues in a different direction.

In the S-graph framework, precedences between tasks are represented by directed arcs. In case of the NIS policy, the schedule arcs for a unit are directed to the next task from the recipe-subsequent task(s) of the previous task (See Chapter 2.3). These arcs represent exactly the aforementioned logical constraints independently of the value of transfer times. Cross transfer appears as a directed, 0 weighted cycle in the S-graph as shown in Figure 3.5 for the wrong schedule in Figure 3.4 - *a*).

In case of positive transfer times, this cycle would have a positive weight and the longest path could not be determined, i.e., this procedure would cover the logical constraint of the transfer. In case of 0 transfer times, the longest path procedure may still succeed (as the equations were feasible in case of the MILP models). However, the logical information is still kept, and the cycle detection algorithm recognizes the infeasibility.

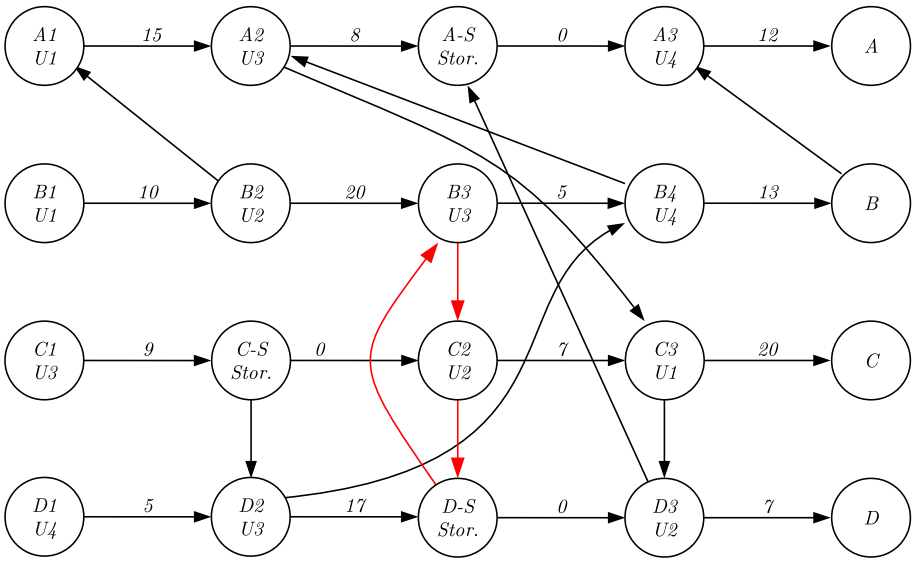


Figure 3.5: Cycle in the S-graph representation of the schedule delivered by the MILP formulations

Since the precedence based models and the S-graph framework has a lot in common, this approach can be "translated" to precedence based MILP models as well, and it is exactly what Ferrer-Nadal et al.[31] has published. As the algorithm is not well formulated in the paper[10], a reformulated, corrected, and clarified pseudo code is presented in Algorithm blocks 3.1 and 3.2.

The main routine goes through all of the plausible unit-task pairs, and starts a recursion. To enumerate each cycle only once, it is assumed without the loss of generality that the cycle starts at the unit with the smallest index.

For precedential recipes, the algorithm remains the same, with the only modification that the **search** subroutine is called recursively for not only i^+, but for all of the subsequent tasks of i.

At each iteration it is checked, whether the subsequent task of the last assignment can be performed by the unit that started the cycle. If yes, the cycle is closed and a new constraint is generated with the function **generate**. This function basically generates the following constraint:

$$\sum_{X \in \mathcal{X}} X + \sum_{Y \in \mathcal{Y}} Y \leq |\mathcal{X}| + |\mathcal{Y}| - 1$$

The constraint ensures that the collected assignment and allocation variables in $\mathcal{Y}$ and $\mathcal{X}$ cannot all take the value of 1, which would cause this cycle.[11]

After this part, the algorithm enumerates all the possible units for the subsequent task of the last assignment, and all the suitable tasks from those whose product has not yet been included to the cycle. The cycle is extended with the assignment, and the function calls

[10] The original publication contains a flow diagram with undocumented notations, and some errors.

[11] Collecting the assignment and precedence variables separately is not necessary. However, this enhances understandability.

Algorithm 3.1 Algorithm to generate constraints avoiding cross transfer in Precedence based MILP models

Inputs

P set of products

n_P number of stages for each product $p \in P$

J set of units

J_i Set of units that can perform task i

Notations

I' set of all tasks, except the last ones in the production, i.e., $\bigcup_{p \in P} \{i_{p,1}, i_{p,2}, \ldots, i_{p,n_k-1}\}$

J_j^+ set of units that are later in an arbitrary ordering of the units.

$X_{i,i'}$ is a general precedence binary variable denoting whether task i precedes task i' if performed in the same unit

$Y_{i,j}$ is the assignment binary variable denoting whether task i is assigned to unit j or not

for all $j \in J$ **do**
  for all $i \in I_j \cap I'$ **do**
    search(j,i,$\{Y_{j,i}\}$,$\emptyset$,J_j^+,$I' \setminus I_{p_i}$,i^+)
  end for
end for

itself recursively.

Algorithm 3.2 Recursive subroutine for Algorithm 3.1

search(j^0, i^0,$\mathcal{Y}$,$\mathcal{X}$,J^R,I^R,i^l)

j^0, i^0 starting unit-task pair of the cycle

$\mathcal{Y}$ set of already fixed assignment variables

$\mathcal{X}$ set of already fixed precedence variables

J^R set of units not yet in the cycle

I^R set of non-final tasks of products not yet in the cycle

i^l last task in the cycle, i.e., the subsequent task of the last assignment

if $\mathcal{X} \neq \emptyset \wedge i^l \in I_j^0$ **then**
  generate($\mathcal{Y} \cup \{Y_{j^0,i^l}\}$,$\mathcal{X} \cup \{X_{i^0,i^l}\}$)
end if
for all $j \in J^R \cap J_i^l$ **do**
  for all $i \in I^R \cap I_j$ **do**
    search(j^0,i^0,$\mathcal{Y} \cup \{Y_{j,i^l}, Y_{j,i}\}$,$\mathcal{X} \cup \{X_{i,i^l}\}$,$J^R \setminus \{j\}$,$I^R \setminus I_{p_i}$,$i^+$)
  end for
end for

The shortcoming of this approach is that the model can no longer be implemented in a single primitive model description language. A higher level language supporting loops and generating equations is needed.

If the analogy between the precedence based models and the S-graph is used, the behavior of the algorithm can loosely[12] be described as follows:

1. Take an S-graph where all of the possible schedule arcs are inserted, and the recipe arcs are removed. See Figure 3.6 for the example given in 2.1.

2. Find all the zero-weighted cycles, and insert an equation that forbids them in the model.[13]

3.3 Other issues

In this section, some other issues are introduced briefly without detailed explanation.

[12] In certain cases the same schedule arc can belong to different units as well, for which additional care must be taken.

[13] There are efficient algorithms published in the literature for the enumeration of all of the cycles in a graph.[60, 89, 79, 111]

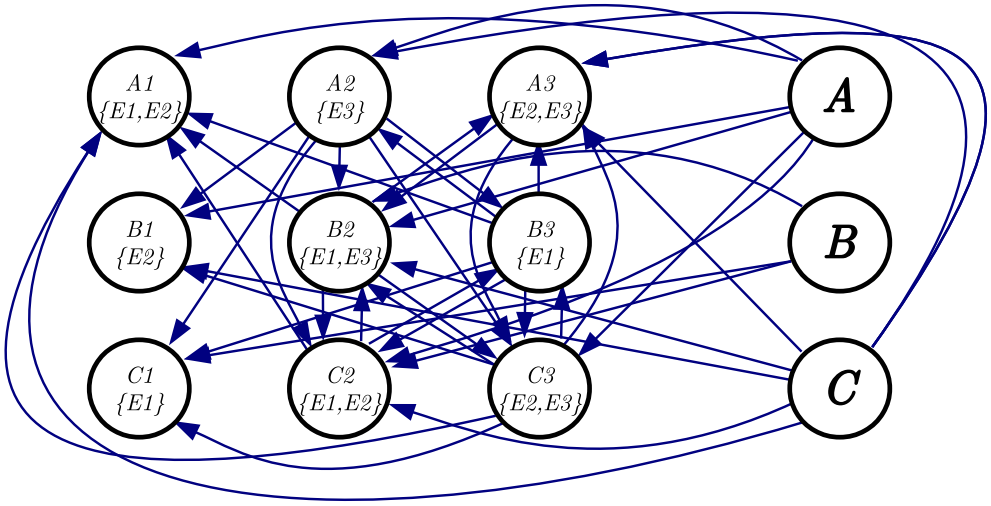


Figure 3.6: The S-graph containing all the possible schedule arcs for the Example given in Figure 2.1

3.3.1 Long tasks

By investigating the same multiproduct example from Section 3.1, and choosing the number of batches to be $3-2-2-2$, the optimal solution reported by any of the mentioned models is suboptimal. The reason is not to be found in the iterative approach, as the model is not capable of finding the practically optimal solution with larger number of time points either. As briefly mentioned in Section 2.1, the problem lies with the typical constraint that if a task starts at a time point, it will finish by the subsequent one. This assumption will usually not disregard the optimal solutions when the processing times are very similar in range. However, in case of a long and several shorter tasks, it is feasible in practice that the short ones are performed after each other in parallell with the long task. Many models, however, cannot find this solution, as tasks are not allowed to overlap several time points. This issue has been already reported by several papers, and addressed partially. In both the global time point[17] and the unit specific[117] models, the binary variables were extended with an additional index: the finishing time point. Thus, theoretically, if there are n time points, and k plausible unit-task pairs, the number of binary variables increased from $n \times k$ to $\binom{n}{2} \times k = n \times k \times \frac{n-1}{2}$. As a result, the number of time points became even more crucial, and much smaller problems got impossible to be solved in a reasonable time. In order to avoid this computational disadvantage, these models were extended with an additional parameter, Δ[14] that denotes the maximal number of time points that the execution of a task can overlap. The special case, $\Delta = 0$ brings the model back to the original case. This way, the number of binary variables is reduced to $(n + (n-1) + \cdots + (n-\Delta)) \times k$ that is roughly $(\Delta + 1) \times n \times k$. Although the optimal solutions can usually be found by using 1 or 2 for Δ, it has to be noted that these cases also roughly double and triple the number of

[14] In the original papers it is Δn for the unit specific, and Δt for the global time point models.

binary variables, resulting in huge increase for the computational need. Also, Δ is an other model parameter, that has to be identified a-priori the optimization, similar to the number of time points, and thus the optimality of the solution cannot be guaranteed. Although there were attempts to identify the ideal value for Δ, counterexamples can easily be created to undermine the soundness of those approaches.

3.3.2 Time point synchronization

Unit specific models are advantageous from the computational point of view. However, the synchronization between the different time point sets can be rather nontransparent. Investigating this issue in detail would require many pages, thus it is omitted here, and only a simple example is given as an illustration: The so called "Different task - different unit" constraints in these models state, that if task i is finishing in time point n, and produces some intermediate, that *can* be consumed by a task i' which will start to be performed by an other unit at time point n, than i' must start later than i finishes. The problem with this assumption is that i' may gets its inputs from a storage or an other unit that also performed task i, but finished earlier, thus the model is over-constraining.

Summary and concluding remarks

This chapter presented some of the modeling issues, that arise in case of the MILP formulations published in the literature. These issues may lead the optimization process to suboptimal or even practically infeasible solutions. Some of these issues were presented in detail and analyzed, others just mentioned and highlighted. Some of the issues are easy to address, others are not. However, even in the former case, the fix could increase the CPU needs drastiqually.

The overall conclusion of the chapter is, that ***empirical tests can never validate the soundness of a presented approach***, which is a common practice in the literature. Moreover, authors should investigate in more detail, what kind of assumptions are hardwired into their model, and highlight them in their papers.

Related publications

- Hegyhati, M., T. Majozi, T. Holczinger, F. Friedler, Practical infeasibility of cross-transfer in batch plants with complex recipes: S-graph vs MILP methods, Chemical Engineering Science, 64, 605-610 (2009). [IF = 2.136]

- Hegyhati, M., F. Friedler, Overview of Industrial Batch Process Scheduling, Chemical Engineering Transactions, 21, 895-900 (2010).

Related conference presentations

- Hegyháti, M., A. Éles, F. Friedler, Modeling issues of mathematical programming based approaches in batch process scheduling, presented at: PRES 2012, Prague, Czech Republic, Aug 25-29, 2012.
- Hegyháti, M., T. Holczinger, F. Friedler, In-Depth Study and Comparison of S-Graph Framework and Precedence Based MILP Formulations for Batch Process Scheduling, presented at : AIChE Annual Meeting 2011, Minneapolis, USA, October 16-21, 2011.
- Hegyhati, M. and F. Friedler, Overview of industrial batch process scheduling, presented at: PRES 2010, Prague, Czech Republic, August 29 - September 1, 2010.
- Hegyhati, M., T. Majozi, T. Holczinger, F. Friedler, Practical feasibility of mathematical models in scheduling, presented at: VOCAL 2008, Veszprem, Hungary, December 15-17, 2008.

Related conference posters

- Éles A., B. Kovács, B. Tóth, M. Hegyháti, Empirical Analysis of Methods and Software Tools for Batch Process Scheduling, presented at: VOCAL 2012, Veszprém, Hungary, Dec 11-14, 2012.
- Hegyháti, M., A. Éles, F. Friedler, Modeling issues of mathematical programming based approaches in batch process scheduling, presented at: PRES 2012, Prague, Czech Republic, Aug 25-29, 2012.

Chapter 4

Throughput or profit maximization with the S-graph framework

The S-graph framework was originally introduced to address makespan minimization problems with NIS or UIS policies. The extension to throughput-, revenue-, or profit maximization is not as trivial as in the case of the time discretization based MILP approaches, as the S-graph algorithms (like the precedence based formulations) consider the number of batches as an input parameter, which is unknown a-priori to optimization in these cases. In this chapter, an algorithm is presented to extend the S-graph framework to address revenue maximization problems. Throughput or profit maximization can be addressed analogically. The basic idea that has been presented by Majozi and Friedler[82] and by Holczinger et al.[55], is to have a top-level branching for the number of batches for each product, that will be called as a ***configuration*** in this chapter. The approach described in Section 4.1 and 4.2 relies, however, on fixed batch sizes, that is not always the case for throughput maximization problems. Section 4.3 introduces an approach to discretize the batch sizes without the loss of generality in order to provide the recipes with fixed batch sizes for the previously described algorithm.

4.1 Main algorithm for revenue maximization

In this section, it is assumed, that for all products the batch size is fixed, i.e., the revenue of one batch of a product is known, and it is denoted by R_p for all $p \in P$.

As it has been mentioned before, it is not known in advance at what number of batches the revenue will be maximal. Thus, the basic concept of the approach is to introduce a top level search space for all the possible batch numbers. The pseudo code of the algorithm can be found in the algorithm box 4.1

Essentially, the algorithm first initializes the set $\mathcal{S}$ with all the possible batch numbers for the products. ($\mathbb{Z}^*$ denotes the set of non-negative integers.) Then, in each iteration a batch number configuration is selected and removed from the set by the **select_remove** function,

Algorithm 4.1 Throughput minimization with the S-graph framework

```
revenue^cb := 0
S := (Z*)^|P|
while S ≠ ∅ do
  x :=select_remove(S)
  if feasible(recipe(x),t^H) then
    if revenue(x)> revenue^cb then
      revenue^cb :=revenue(x)
      x^cb := x
      update(S, revenue^cb)
    end if
  else
    S := {x' ∈ S | x' ≱ x}
  end if
end while
if revenue^cb ≠ 0 then
  return (x^cb, revenue^cb)
end if
```

and tested for feasibility, i.e.whether it can be produced within the given time horizon, t^H. If the configuration is feasible, and has a higher revenue than the best found so far, the currently best solution is updated, as well as the set $\mathcal{S}$. If the configuration is infeasible all the configurations larger[1] than the current one are removed from the set $\mathcal{S}$, due to the fact, that if there would be a feasible schedule for a larger configuration, a feasible schedule for x could be created as well by removing all the superfluous batches from it. When $\mathcal{S}$ gets empty, the algorithm returns with the best configuration and the corresponding revenue if there was a feasible solution.

Though the algorithm is simple, several aspects need to be discussed:

- The algorithm uses several subroutines. Some of them is discussed in the next section in more detail. The function **recipe** generates the recipe graph of a problem where x_i batches of the ith product is to produced for each i, and all the other necessary information needed by the feasibility tester subroutine, e.g., set of tasks, units. The **revenue** function simply returns the revenue for the configuration, which in this case is the scalar product $x \cdot R$.

- The set $\mathcal{S}$ is infinitely large, and seemingly only one element can be guaranteed to be removed from it in each iteration. This suggests, that the algorithm is not finite. Note, however, that this is not the case, the algorithm always finishes in a finite number of steps if the selection of configurations is appropriate. Let t_p^{min} be the smallest processing time for any of the tasks of product $p \in P$. Obviously, $[\frac{t^H}{t_p^{min}}]$ is an upper limit (though probably not very tight) for the number of batches that can

[1] A configuration is considered to be larger or equal than an other one, if it entails at least as many batches from each product than the other configuration.

be produced from p. As an upper bound exists for each product, the algorithm ends in a finite number of steps if it chooses configurations, such that $x_p \leq [\frac{t^H}{t_p^{min}}] + 1$. The configurations where $x_p = \lfloor \frac{t^H}{t_p^{min}} \rfloor + 1$ for at least one product will definitely be infeasible, and thus they will remove all the other configurations from $\mathcal{S}$. In practice, i.e., in the implementation of this algorithm the set $\mathcal{S}$ is not initialized like this, see more about this at the description of the **select_remove** function.

- In the presented version, the algorithm does not return the schedule of the optimal solution. The implementation of course saves the feasible solution found and returns it at the end.

Assuming that the subroutines work properly, it is easy to see, that the algorithm is sound, as it evaluates all the possible configurations.

4.2 Subroutines for the algorithm

In this section several subroutines of the revenue maximization algorithm is discussed.

4.2.1 The select_remove method

The only requirement towards the **select_remove** function to make the algorithm sound is that it should select and arbitrary element of $\mathcal{S}$ and remove it. For the finiteness of the algorithm however, the function should satisfy some additional condition. A simple, yet sufficient condition is that the function should never select an element x such that none of its "preceding configurations" has been selected before. A configuration x' is preceding x if for some $k \in \{1, \ldots, |P|\}$, $x'_k = x_k - 1$ and $x'_i = x_i$ for all $i \neq k$. It is easy to see that if the function follows this rule, the algorithm will always have a finite number of iterations, as the feasible region is finite (as discussed in the previous section) and its border is finite as well.

In the implementation, however, the function behaves a bit differently. First, the function goes through the configurations where only one product is produced, and finds the largest feasible batch number, b_p^{max} for each $p \in P$. Based on these values, a finite region of interest can be defined, that contains all the feasible configurations (and some infeasibles as well). For two products, the result of this initial step is illustrated in Figure 4.1.

The two products are denoted by A and B, and the two axises by N_A and N_B referring to the number of batches for A and B, respectively.

After identifying this feasible region, the function can have different strategies to select the configuration, which could have a major effect on the CPU requirements. As shown in Figure 4.2, the number of examined configurations can be significantly different for different selection strategies.

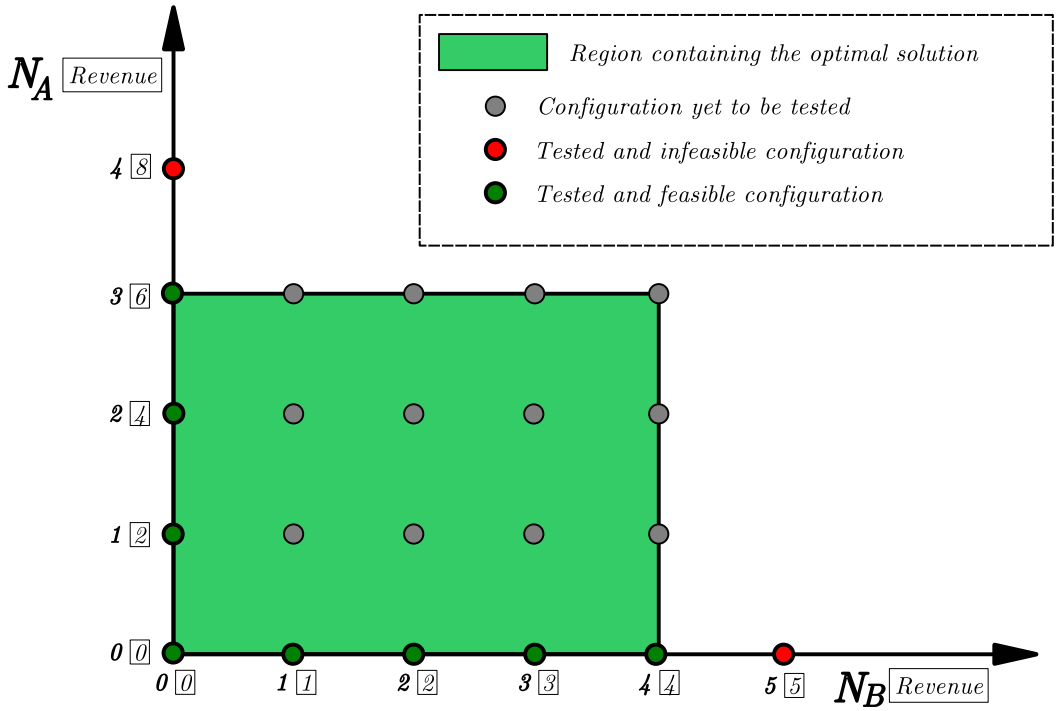


Figure 4.1: Finding the initial region containing the optimal solution for revenue maximization

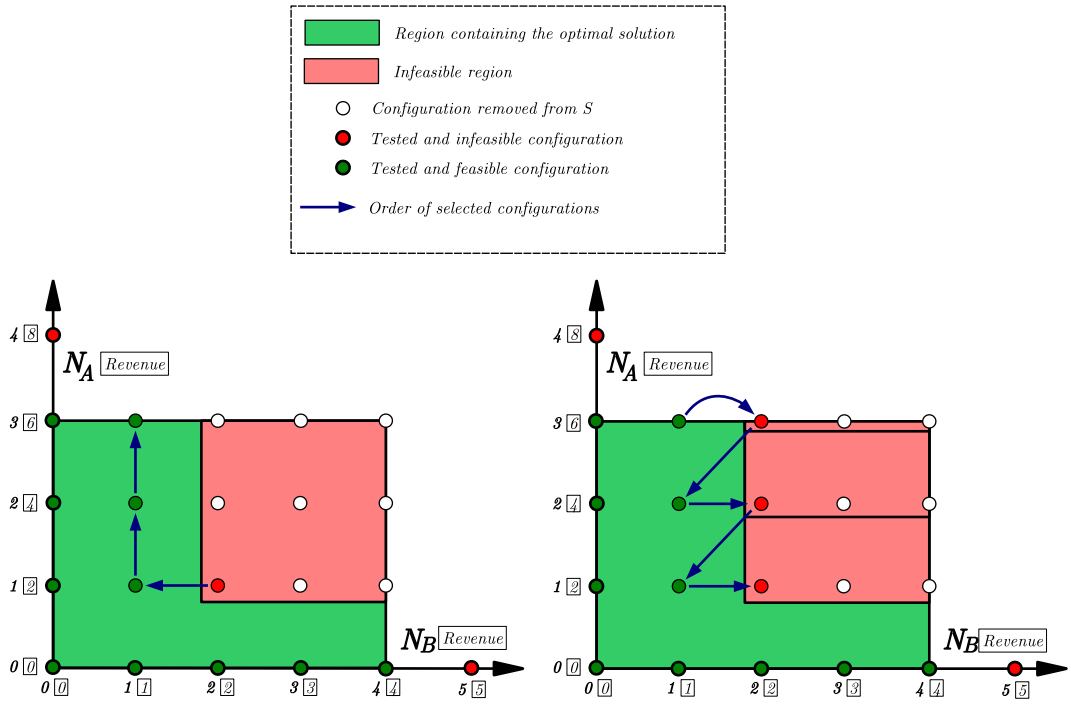


Figure 4.2: Illustration of the significance of configuration selection on the number of iterations

As it can be seen from the figure, in one of the cases the algorithm had 4 iterations after the initial phase, in the other there was 6 iterations. The former one could save 2 iterations by testing the configuration $(2, 1)$, and finding it infeasible first. Empirical test has shown, that the most of the time needed by the algorithm belongs to infeasible configurations. Thus, testing the "minimal" infeasible configurations first is crucial for reducing the CPU time. Orosz [96] has investigated the effects of different selection strategies in his work.

4.2.2 The update method

Essentially, there are two tests that has to be carried out at each configuration:

- Is the configuration feasible?
- Does the configuration provide higher revenue than the current best?

In many practical examples, the evaluation of the first question takes much more time than that of the second. According to experimental experiences, this is especially true, if the configuration is infeasible, as it takes more time for the feasibility tester to examine all the infeasible branches, than finding a single feasible schedule. In these cases it may be more beneficial to evaluate first the second question, and if the answer is no, then purge the case, even if it could be used for reducing the search space if it turns out to be infeasible.

The purpose of the **update** function is to make this type of behavior possible. Although there are many possibilities, there are two implementations present:

1. The function does nothing
2. The function removes those untested configurations from the search space that do not have higher revenue than the current best solutions, i.e., $\mathcal{S} := \{x' \in \mathcal{S} \,|\mathbf{revenue}(x') \leq revenue^{cb}\}$

The first case is evident; in the second case, the region is reduced each time when a feasible configuration is found. Note that in this case, the if statement testing whether the revenue of x is higher than the current best is unnecessary. In the implementations this second option is only used after finding the initial region of feasibility (see the previous section). When this region is identified, one of the tested configurations provide the highest profit found so far. This solution can immediately be used to reduce the search space, as shown in Figure 4.3.

In this case, the revenue of producing 3 batches of A was 6, and all the configurations indicated by white dots could be removed from the search space, as they had the revenue of at most 6.

Using the second version of this function, the search space $\mathcal{S}$ gets reduced at each iteration, regardless, whether the configuration x turns out to be feasible or infeasible. Using the two selection strategies shown in Figure 4.2, and the second version of the updating function, the evaluation of the search space is illustrated in Figure 4.4.

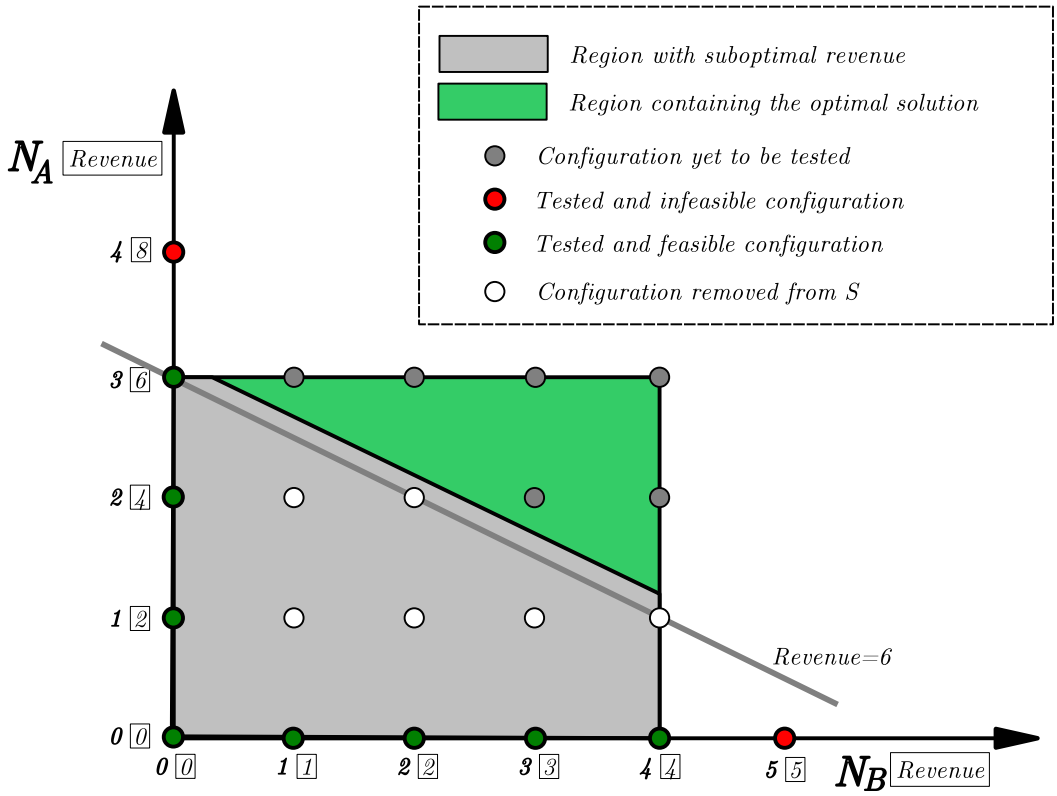


Figure 4.3: Reducing the search space based on the revenue of the best feasible solution

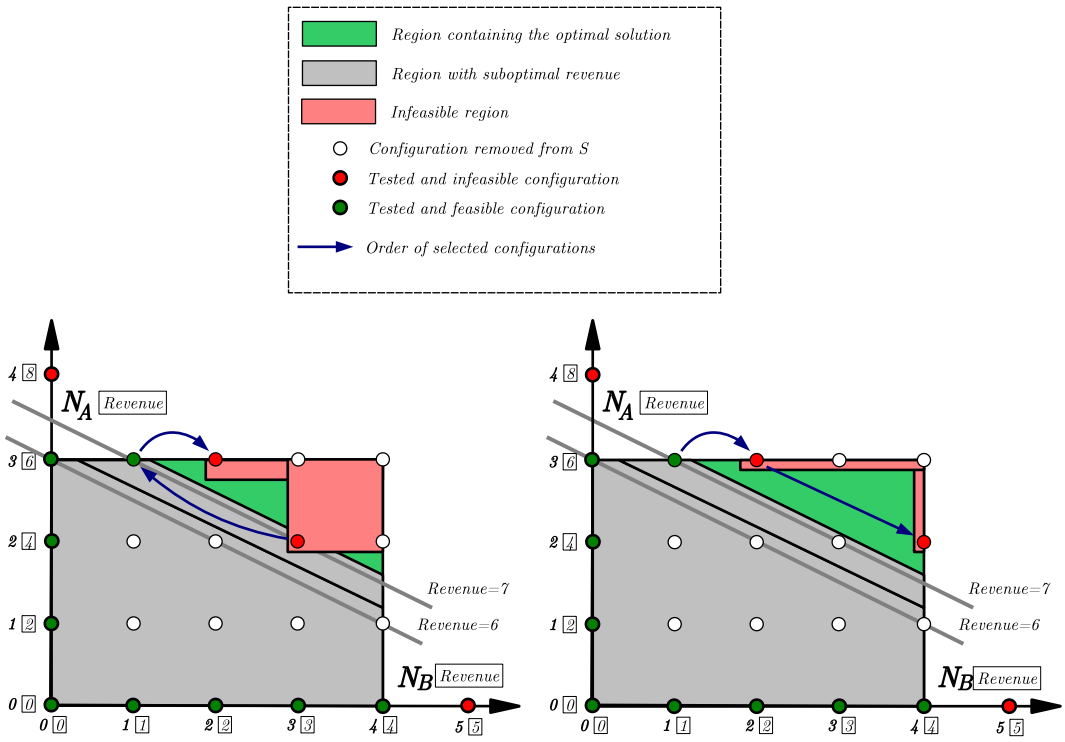


Figure 4.4: Evaluation of configurations with revenue updates on $\mathcal{S}$

Although, this reduction can be beneficial in many cases, it has drawbacks as well, which can be seen by comparing Figures 4.2 and 4.4 as well. In the latter case, regardless of the selections strategy, two infeasible configurations needed to be tested, which was not the case for the first strategy in the former one. As testing infeasible configurations usually take most of the computational time, in most of the cases it is advised not to reduce $\mathcal{S}$ based on the current best revenue.

4.2.3 The feasible method

The **feasible** method plays a key role in the maximization of the revenue, as this function is responsible for the evaluation of each configuration. The function must return true if there is a schedule with at most the time horizon for the given number of batches, and false otherwise.

The simplest implementation is by using the makespan minimization function described in Subsection 2.3.2. At each configuration the number of batches for each product are available, thus the minimal makespan for that amount of products can be found. Then, the result is compared with the time horizon: if it is not larger, the configuration is feasible, and infeasible otherwise.

This approach however does numerous unnecessary calculations:

- Even if a solution has been found within the time horizon, the function continues to find the solution with the optimal makespan, that is, in this case, out of the interest of the main algorithm.
- Even if all the subproblems has higher lower bounds then the time horizon, i.e., all of the feasible solutions has longer makespan than the time horizon, the algorithm still continues to find the optimal solution, although the configuration will be evaluated as infeasible anyway.

In order to avoid these unnecessary calculations the makespan minimization algorithm must slightly be modified the following way:

- The variable $makespan^{cb}$ must be replaced by t^H.
- When a feasible solution is found the algorithm should return with true immediately.

The pseudo code for this feasibility tester function is presented in Algorithm block 4.2.[2]

This simple modification can significantly reduce the time needed for the optimization. However, there are still redundant calculations, since if x is a configuration larger than x', then part of the the search tree for the feasibility test of x appears in the tree of the test for x' as well. Orosz [96] has investigated this opportunity, and provided a global search tree

[2]Note that the presented algorithm simply returns true, and does not return the schedule. Obviously, the algorithm can be easily modified to return the schedule or save it in a global variable, etc.

Algorithm 4.2 Feasibility tester subroutine for revenue maximization

feasible$(G(N, A_1, A_2, w), I, J, t^H)$

$G(N, A_1, A_2, w)$ recipe graph

I, J set of tasks and units, respectively

t^H time horizon

$\mathcal{S} := \{(G(N, A_1, A_2, w), I, J, \emptyset)\}$
while $\mathcal{S} \neq \emptyset$ **do**
  $(G(N, A_1, A_2, w), I', J', \mathcal{A})$:=**select_remove**$(\mathcal{S})$
  if bound$(G) < t^H$ **then**
    if $I' = \emptyset$ **then**
      return TRUE
    else
      j :=**select**(J')
      for all $i \in I_j \cap I'$ **do**
        $G^i(N, A_1, A_2^i, w^i) := G(N, A_1, A_2, w)$
        for all $i' \in \bigcup_{(i',j)\in\mathcal{A}} I_{i'}^+ \setminus \{i\}$ **do**
          $A_2^i := A_2^i \cup \{(i', i)\}$
        end for
        for all $i' \in I_i^+$ **do**
          $w_{i,i'}^i := t_{i,j}^{pr}$
        end for
        $\mathcal{S} := \mathcal{S} \cup (G^i(N, A_1, A_2^i, w^i), I' \setminus \{i\}, J', \mathcal{A} \cup \{(i, j)\})$
      end for
      if $I' \subseteq \bigcup_{j'\in J', j\neq j'} I_{j'}$ **then**
        $\mathcal{S} := \mathcal{S} \cup (G(N, A_1, A_2), I', J' \setminus \{j\}, \mathcal{A})$
      end if
    end if
  end if
end while
return FALSE

based method where the feasibility testers use the results of the preceding configurations to accelerate the search. The approach has been implemented with a task based feasibility tester, and despite its development phase, it has shown promising results.

4.3 Flexible batch sizes

For many case studies and literature examples the batch sizes are not fixed, and if several units are capable of performing a task, they are allowed to do it in parallel. This type of problem definition is especially often for throughput/revenue maximization problem, but appears for makespan minimization as well. Time discretization based approaches can address this issue as they are, the S-graph framework, however, needs some adjustments.

The previously described algorithm requires that the recipe is fixed, and the revenue is known for one batch of a product. With the aforementioned problems, however, neither of these two is guaranteed. As an illustration for the proposed approach, the example from Kondili et al.[66] is taken, that was shown in Figure 1.5.

The process consists of 5 tasks: heating, 3 reactions and a separation, the material flows are clearly represented in the figure.[3] For these tasks, 4 units are available, dedicated Heater and Separator, both with 100kg of capacity for the Heating and Separation, respectively. For the three reactions, two different reactors $R1$ and $R2$ with identical processing times, and capacities of 80 kg and 50 kg are available, which may be used in parallel. It is assumed that all of the units can operate with any load smaller than their capacity, i.e., there is no lower bound for their load.

The process produces two products with identical revenues, and an additional constraint is given: no intermediate material may be left at the end of the production, i.e., it is not allowed to produce only Product 1 via the first part of the process. Moreover, there are no storages available, i.e., NIS policy is assumed.

Each reaction can be performed by either one of the reactors or with both of them parallel. For the 3 reactions, this results in $3^3 = 27$ different "fixed recipes", that can have different batch size intervals, as represented in Table 4.1.

To apply the previously described S-graph algorithm for revenue maximization, a separate S-graph recipe for each of these cases must be created, thus the top level search region would become a 27 dimensional space. This would result in an enormous CPU need for the optimization, thus, the reduction on the number of cases is essential.

When looking at the table, it can easily be seen that only several different values repeat. The reason behind this comes from the material balances, for example, even if we assign both $R1$ and $R2$ instead of just $R1$ to Reaction 3, the output will not be higher, if the supply from the previous reactions does not reach a certain level. If two different cases c and c' has the same maximal revenue, but c uses only a (not necessarily real) subset of units for each

[3] In the original example, 10% of the output of separation is $IntAB$ that is recycled. This is neglected, as the presented algorithm is not capable of addressing problems with loops in their recipe.

Case	Reaction 1	Reaction 2	Reaction 3	Max revenue
1	$R1$	$R1$	$R1$	86.00
2	$R1$	$R1$	$R2$	71.67
3	$R1$	$R1$	$R1\&R2$	86.00
4	$R1$	$R2$	$R1$	53.75
5	$R1$	$R2$	$R2$	53.75
6	$R1$	$R2$	$R1\&R2$	53.75
7	$R1$	$R1\&R2$	$R1$	114.67
8	$R1$	$R1\&R2$	$R2$	71.67
9	$R1$	$R1\&R2$	$R1\&R2$	139.75
10	$R2$	$R1$	$R1$	86.00
11	$R2$	$R1$	$R2$	71.67
12	$R2$	$R1$	$R1\&R2$	86.00
13	$R2$	$R2$	$R1$	53.75
14	$R2$	$R2$	$R2$	53.75
15	$R2$	$R2$	$R1\&R2$	53.75
16	$R2$	$R1\&R2$	$R1$	89.58
17	$R2$	$R1\&R2$	$R2$	71.67
18	$R2$	$R1\&R2$	$R1\&R2$	89.58
19	$R1\&R2$	$R1$	$R1$	86.00
20	$R1\&R2$	$R1$	$R2$	71.67
21	$R1\&R2$	$R1$	$R1\&R2$	86.00
22	$R1\&R2$	$R2$	$R1$	53.75
23	$R1\&R2$	$R2$	$R2$	53.75
24	$R1\&R2$	$R2$	$R1\&R2$	53.75
25	$R1\&R2$	$R1\&R2$	$R1$	114.67
26	$R1\&R2$	$R1\&R2$	$R2$	71.67
27	$R1\&R2$	$R1\&R2$	$R1\&R2$	139.75

Table 4.1: 27 different "fixed recipes" for the example by Kondili et al.[66]

reaction of those used by c', than c dominates c'. As an example, case 9 dominates case 27, and case 24 is dominated by all of the cases 4,5,6,13,14,15,22,23. Obviously, if a case is dominated by an other, it can be excluded from the investigation without using loosing the guarantee for the global optimality. For the example above, the cases, which are not dominated by an other are shown in Table 4.2 ordered by the maximal revenue.

Case	Reaction 1	Reaction 2	Reaction 3	Max revenue
4	$R1$	$R2$	$R1$	53.75
5	$R1$	$R2$	$R2$	53.75
13	$R2$	$R2$	$R1$	53.75
14	$R2$	$R2$	$R2$	53.75
2	$R1$	$R1$	$R2$	71.67
11	$R2$	$R1$	$R2$	71.67
1	$R1$	$R1$	$R1$	86.00
10	$R2$	$R1$	$R1$	86.00
16	$R2$	$R1\&R2$	$R1$	89.58
7	$R1$	$R1\&R2$	$R1$	114.67
9	$R1$	$R1\&R2$	$R1\&R2$	139.75

Table 4.2: Non-dominated cases for the example by Kondili et al.[66]

After this reduction, still, 11 different cases should be given as input to the S-graph algorithm. In order to further reduce this number, several cases can be merged together. As an example cases 4 and 5 are identical except for the selection for the third Reaction, thus these two cases can be merged togerther by the assignments $R1$, $R2$, $R1 \vee R2$ for Reactions 1,2, and 3, respectively. Applying the same idea, the 6 cases shown in Table 4.3 remain.

Case	Reaction 1	Reaction 2	Reaction 3	Max revenue
4,5,13,14	$R1 \vee R2$	$R2$	$R1 \vee R2$	53.75
2,11	$R1 \vee R2$	$R1$	$R2$	71.67
1,10	$R1 \vee R2$	$R1$	$R1$	86.00
16	$R2$	$R1\&R2$	$R1$	89.58
7	$R1$	$R1\&R2$	$R1$	114.67
9	$R1$	$R1\&R2$	$R1\&R2$	139.75

Table 4.3: Merged non-dominated cases for the example by Kondili et al.[66]

To all of these merged cases, the recipe graph can be generated, as shown in Figure 4.5.

4.4 Empirical tests

In this section, the results of the implemented algorithm are presented via three examples. For each example, the problem has been scaled and solved with different time horizons. At each case altogether 18 variants of the throughput maximization algorithm have been

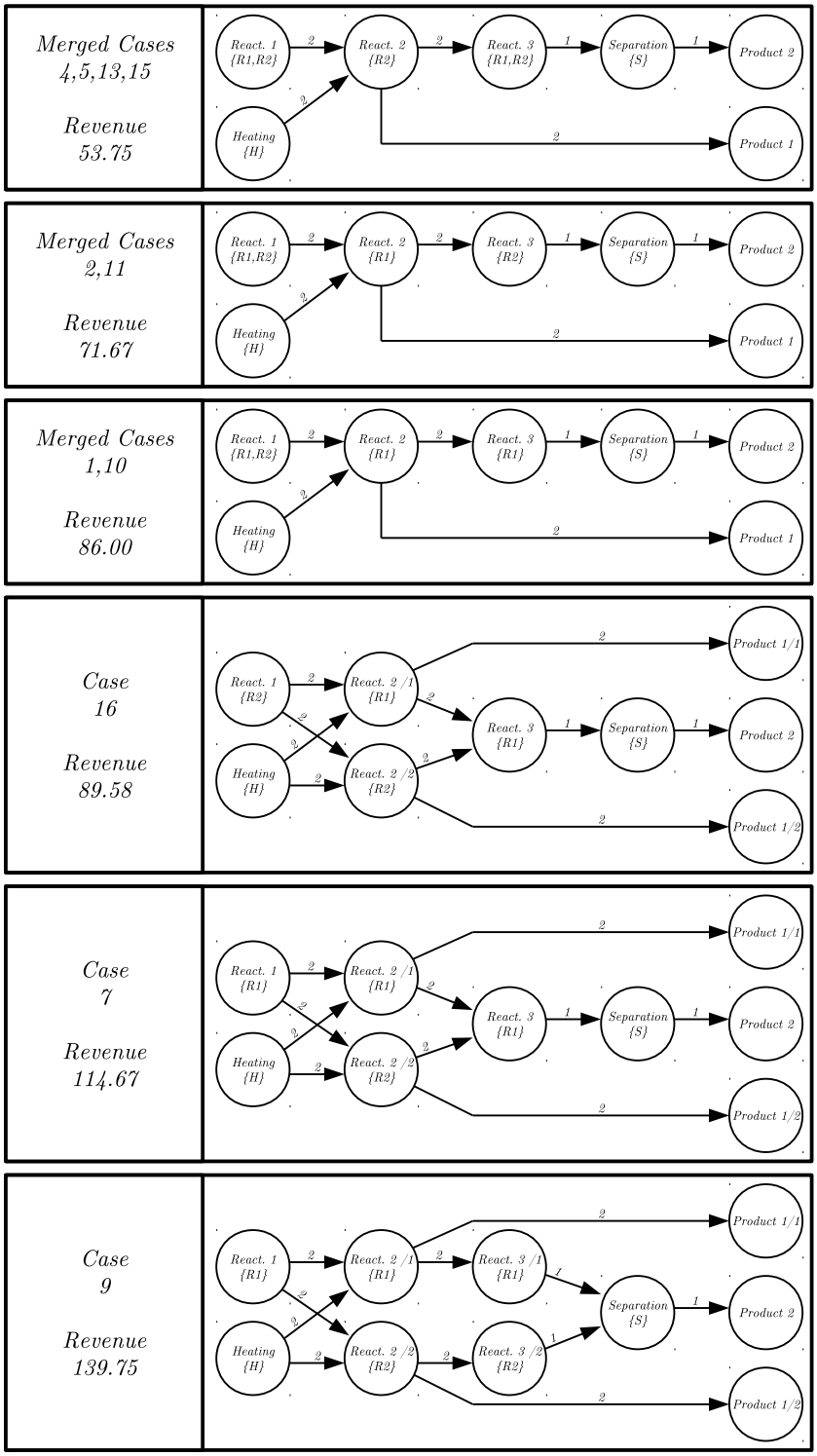


Figure 4.5: The corresponding recipe graphs for the 6 cases in Table 4.3

tested and compared. These variants add up by the possible combinations of 3 different subproblem selection rules, 3 different update and 2 different feasibility subroutines.

Each of the subproblem selection strategies first explores how many batches of a single product can be produced within the given time horizon, then they explore the bounded region

LEX in a lexicographical order,

BFS via breath-first search,

DFS via depth-first search.

The update subroutines:

E empty, no configuration is removed via the bound of a feasible solution

F only the best axial configuration is used for removing some of the configurations after the initialization phase

U the update function removes all the configurations that has lower revenue than the currently best feasible solution

Last, the makespan minimization (**MM**) and the described feasibility tester (**FT**) is used for the evaluation of the configurations. At each run, the time limit of 1 hour was used for the solver algorithm.

The subsections present only some highlights of the results due to space limitations. The table of all of the results can be found in Section C.1. When the algorithms have not reached the 1 hour time limit, they delivered the same globally optimal solution.

A general experience is that the type of the feasibility tester have a major effect on the CPU times, as expected. Usually, most of the CPU time is taken for the testing of infeasible configurations, where the **FT** does not gain any advantage of stopping after the first feasible solution. The initial bound of the **FT** subroutine however, cause a significant reduction on the search space.[4] Thus, in the following subsections the variants having **MM** subroutine are not investigated.

For all of the examples, NIS storage policy were considered for all intermediates.

4.4.1 Pharmaceutical case study

This example is taken from a multinational pharmaceutical company. There are 5 hair and skin care products produced, all of them via two consecutive steps: mixing and packing. For the packing, there are 3 identical packing lines available, and the packing time is uniformly 12 hours for all of the products. There are 4 mixing vessels available, however, they differ in applicability and processing times due to the different stirrer designs. The detailed processing times are given in table 4.4

[4]Some additional tests were run using the makespan minimization subroutine with the initial bound, and the results were close to that of the **FT** approach.

Product	Revenue (cu)	Processing time (h)				
		Mixing vessels				Packing lines
		V1	V2	V3	V4	
Cream 1	2	10	5		5	12
Cream 2	3	12	10	7		12
Conditioner	1			12		12
Shampoo	3.5		8	13		12
Lotion	1.5	10	6		9	12

Table 4.4: Processing times and revenue data for the pharmaceutical case study

A summary of the test results are shown on a logarithmic scale in Figure 4.6, where the colors correspond to configuration selection rules, and the shapes for the different update functions.

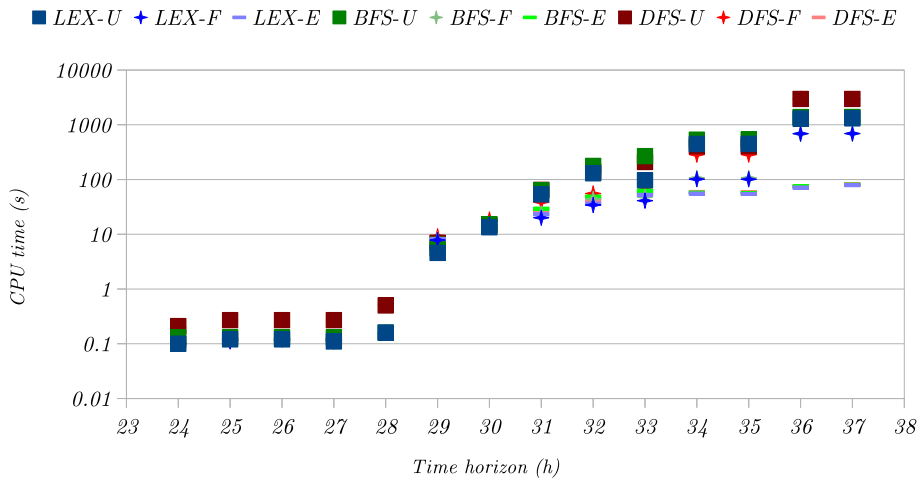


Figure 4.6: Summary of results for the pharmaceutical case study

Based on these results the following conclusions can be drawn for this example:

- For most of the cases, the alternative solution approaches could not outperform each other more than a magnitude.
- For small problems (with time horizon less than 29) the CPU times do not increase drastically.
- After 29 hours of time horizon, the CPU times have a drastical increase.
- For larger problems the alternatives with the **E** update function dominate the others, and there is only a negligible difference between them for the different configuration selection strategies.

4.4.2 Agrochemical example

In this example herbicide is produced through three reactions, a separation, and an evaporation. The flowsheet of the problem is given in Figure 4.7.

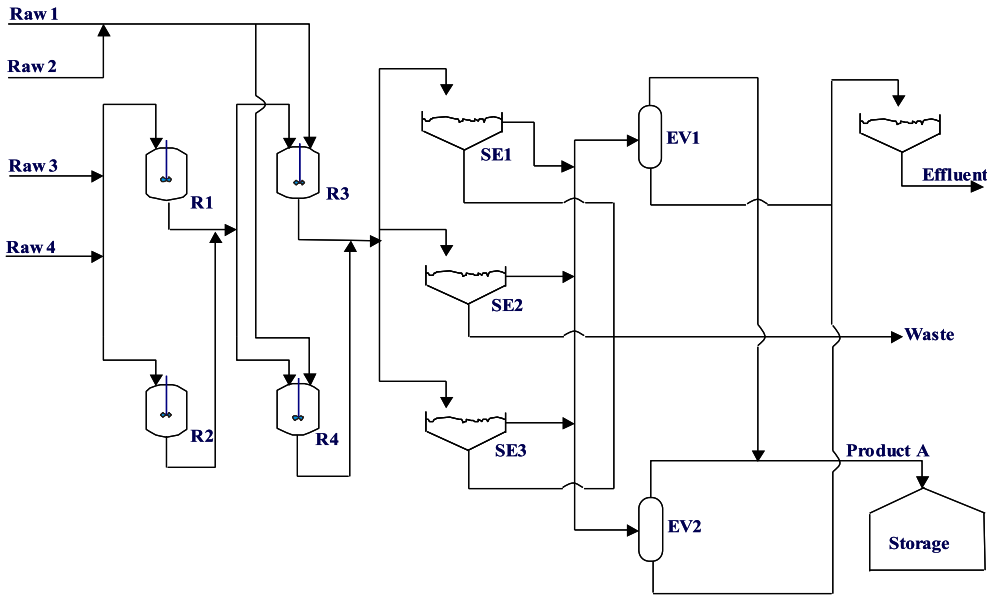


Figure 4.7: Flowsheet of the agrochemical process for herbicide production

The processing times, capacities of the units, output ratios, and other details of the problem can be found in the paper by Majozi and Friedler [82]. Although there is only one product, the batch sizes are not fixed, and two units may work parallel on the same batch. Thus the algorithm of Section 4.3 need to be applied to create fixed recipes. After this preprocessing steps, there are two different fixed recipes as shown in Figure 4.8, where the revenues of $F1$ and $F2$ are 3.7 and 4.5 cost units, respectively.

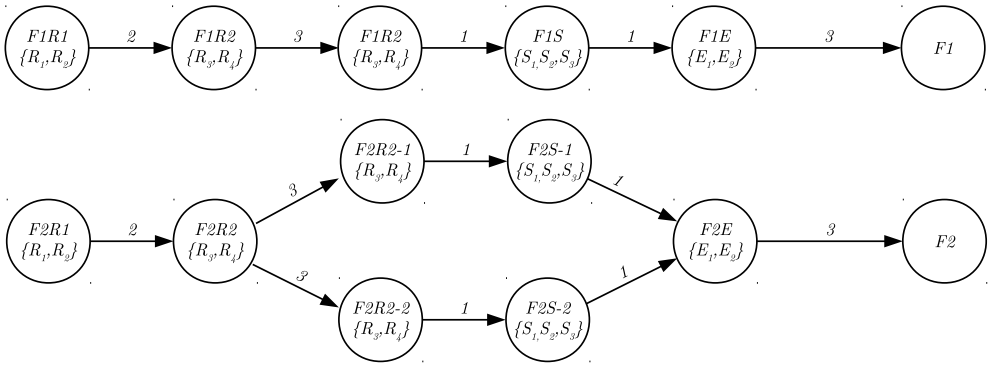


Figure 4.8: S-graph of the two fixed recipes for the agrochemical process

The results of the tests are shown in Table 4.5, and the best CPU times are indicated in each row.

Time horizon (h)	CPU time (s)								
	LEX			BFS			DFS		
	U	F	E	U	F	E	U	F	E
13	3.06	3.07	3.05	3.08	3.04	3.05	**3.03**	3.08	3.07
14	339	340	**205**	405	405	414	405	402	269
15	453	**451**	453	583	582	591	589	583	581
16	3600	3600	3600	3600	3600	3600	3600	3600	3600

Table 4.5: Test results for the agrochemical example

It is easy to see, that in case of this problem, the increase in the time horizon has a drastic effect on the cpu times. Each additional hour in the time horizon resulted in at least one magnitude growth in the CPU time. Based on this few results, it can be observed, that the **E** update function dominates here the other two as well. Moreover, in general, the **LEX** configuration selection strategy proved to be the most efficient.

4.4.3 Literature example

For the last comparisons, the example from Section 4.3 is taken. As described in that section, there are 6 fixed recipes for this problem with two products. The results of the test are shown in Table 4.6, and again, the best CPU times are indicated in each row.

Time horizon (h)	CPU time (s)								
	LEX			BFS			DFS		
	U	F	E	U	F	E	U	F	E
14	16.88	15.84	15.21	**14.95**	18.52	21.07	22.63	22.53	21.01
15	179	134	**113**	208	144	133	437	279	225
16	1260	1159	984	**964**	1496	1512	1594	1629	1403
17	3600	3600	3600	3600	3600	3600	3600	3600	3600

Table 4.6: Test results for the example of Section 4.3

As in the case of the previous examples, the **LEX-E** alternative provides good results. However, interestingly, the **BFS-U** strategy often have good results.

Summary and concluding remarks

In this chapter, the S-graph framework has been extended to throughput or revenue maximization problems. The algorithm is based on the enumeration and maximum search of the feasible set of configurations, i.e., batch numbers, which are producable in the given time horizon. The possibilities for different variations of the general algorithm were presented and thouroughly compared after implementation. The results showed that the approach is

capable to solve the considered set of problems, and provide the optimal solution efficiently when the correct variation is applied.

Related publication

- Holczinger, T., T. Majozi, M. Hegyhati, F. Friedler, An automated algorithm for throughput maximization under fixed time horizon in multipurpose batch plants: S-Graph approach, 17th European Symposium on Computer Aided Process Engineering Elsevier, 24, 649 - 654 (2007).

Related conference presentations

- Majozi, T., F. Friedler, T. Holczinger, M. Hegyhati, S-graph based continuous-time approach for throughput maximization in multipurpose batch plants, presented at: IFORS 2008, Johannesburg, South Africa, July 13-18, 2008
- Hegyhati M., T. Holczinger, T. Majozi, F. Friedler, Transforming STN based scheduling problems to S-graph representation, presented at: XIX Polish Conference of Chemical and Process Engineering, Rzeszów, Poland, September 3-7, 2007.
- Holczinger, T.,T. Majozi, M. Hegyhati, F. Friedler, Throughput Maximization in Multipurpose Batch Plants: S-graph vs Time Point Based Methods, presented at: PRES 2007, Ischia Island, Italy, June 24-27, 2007.
- Holczinger, T.,T. Majozi, M. Hegyhati, F. Friedler, An Automated Algorithm for Throughput Maximization Under Fixed Time Horizon in Multipurpose Batch Plants: S-Graph Approach, presented at: ESCAPE 17, Bucharest, Romania, May 27-30, 2007.
- Holczinger, T.,T. Majozi, M. Hegyhati, F. Friedler, Using S-graph for Throughput Maximization in Multipurpose Batch Plants, presented at: VOCAL 2006, Veszprem, Hungary, December 12-15, 2006.

Chapter 5

Limited- and Zero-wait storage policies in the S-graph framework

The S-graph framework was originally developed for NIS-UW and UIS-UW storage policies, and the only published storage policy extension of the S-graph framework[109] focused on CIS-UW policy. However, in many industrial application LW or ZW policies are required, as certain intermediates lose some physical or chemical properties over time, that would be important for the upcoming task. In this chapter, several new approaches are introduced to tackle these storage policies. As it has already been mentioned in Section 1.2, in case of ZW policy, the restriction on the infrastructure is irrelevant. However, LW itself does not define the storage policy. In this chapter LW will refer to NIS-LW policy, and the approaches are described for that, although they could easily be modified to address UIS-LW policy as well. Moreover, if an approach is presented for LW policy, it can be automatically applied for ZW cases, as it is a special case of LW. Finally, it is not assumed anywhere, that all of the intermediates share the same restriction for waiting time, i.e., some of them are ZW, LW, or even UW.

Note that UW is a relaxation of ZW/LW, thus, all of the schedules with ZW or LW policies on some intermediates remain feasible if all of the LW intermediates are set to UW policy. Later on, the terminology UW-relaxation will be used if the LW restrictions of a problem are disregarded this way.

Most of the constraints in scheduling has a "greater or equal" nature,e.g., a task must start later than the finishing of the previous task in the recipe, or the previous task assigned to the same unit. LW constraints, however, define a "not later then" thus "smaller or equal" type of constraint, which cannot directly be expressed by the ordinary S-graph arcs. There are, however, several ways to address this policy, which are detailed in the following subsections.

In section 5.1 a hybrid approach is introduced, where the S-graph based branching procedure is extended with an LP based bounding function that also includes the "smaller or equal" type of LW constraints.

The approach introduced in section 5.2 relies on the fact that a typical "less or equal"

type of LW constrains, such as $T_1^{start} \leq T_2^{start} + t^{proc}$ can easily be converted to a "greater or equal form": $T_2^{start} \geq T_1^{start} + (-t^{proc})$, which can be modeled by regular S-graph arcs. The weight of the edge, however, becomes negative (non-positive to be precise), which needs some additional care.

In section 5.3 two approaches are introduced for problems with only UW and ZW intermediates without introducing negative weighted arcs in the S-graph. Addressing LW stages is possible through a modeling conversion.

Last, section 5.4 compares the performance of these approaches through several examples.

5.1 Auxilary LP model

The smaller-or-equal type of constraints of LW policy can be addressed by an LP model that can be formulated for each subproblem. The mathematical model can simply built based on the S-graph (N, A_1, A_2):

- A non-negative continuous variable is assigned to each node, which represents its starting time: S_i, for all $i \in N$.
- Schedule arcs represent simple ordering in time, thus a constraint in the form of $S_{i'} \geq S_i$ is added for all $(i, i') \in A_2$.
- For all recipe arc, a similar constraint is added: $S_{i'} \geq S_i + w_{i,i'}$ for all recipe arc $(i, i') \in A_1$, where $w_{i,i'}$ is the weight of it.
- Last, for each task i with LW policy, the constraint $S_{i'} \leq S_i + w_{i,i'} + maxwait_i$ is added, where i' is an upcoming task of i, i.e., $(i, i') \in A_1$, and $maxwait_i$ the maximal allowed waiting time.

The solution of this mathematical model will not provide sharper bounds than the longest path algorithm. However, it will detect if a certain subproblem is not feasible due to wait restrictions. In such a case, the corresponding branch of the search tree is pruned obviously. Note that in case of a partial schedule that is not even feasible for the UW-relaxation, the S-graph approach will detect the infeasibility in the usual manner by finding a directed cycle, thus the solution of the LP model is not necessary.

Advanced LP approach

The most serious drawback of the previously mentioned approach is that the mathematical model needs to be formulated at each subproblem, taking a lot of CPU time, while the LP itself does not provide a sharper bound than the longest path approach. There is, however, a more sophisticated way of using an LP for LW and ZW policies, as suggested in subsection 2.3.3.

Instead of formulating a separate LP model for each subproblem, the LP relaxation of the whole precedence based model can be built for the root subproblem immediately. Then, at each decision, this model is copied, and some of the binary variables are fixed by changing their upper or lower bounds.

Though, these models are much bigger than the formerly introduced ones, their bitwise copying and modifying some variable bounds takes less time than building up the same model from scratch. Moreover, as they have the same number of variables and constraints, the solution of the parent LP model can be used as a starting basis for the dual simplex algorithm. Finally, the solution of these models provide a tighter bound than the longest path algorithm itself.

Note that similarly to the original approach, there is no need to copy the LP relaxation, modify and solve it if the schedule graph contains a cycle or the longest path is higher than the current upper bound.

Moreover, as the models at the subproblems are the same except for the bounds on variables, it may be more beneficial not to copy the model, but use it as a global variable, and adjust the bounds at each subproblem.

5.2 Combinatorial approach with negative weighted arcs

As it has already been mentioned, the less-or-equal type constraints in the form of $T_1^{start} \leq T_2^{start} + t^{proc}$ can easily be converted to a greater-or-equal form: $T_2^{start} \geq T_1^{start} + (-t^{pt})$. This transformation, however, introduces non-positive weights on the arcs. This idea has already been used in for the Alternative graph model to model similar situations [88, 98, 24, 25, 22, 21]

The limited waiting times can easily be modeled by negative weighted arcs. If there is a task i with processing time t_i^{pt}, and maximal waiting time $t_i^{maxwait}$, then two recipe arcs should be inserted into the S-graph as illustrated in Figure 5.1:

- An arc with weight t_i^{pt} from i to its subsequent task(s).
- An arc with weight $-t_i^{pt} - t_i^{maxwait}$ from the subsequent task(s) of i to i.

It is obvious to see that these arcs express exactly the desired constraints. In the figure the interval in which task i' can start (from $T_i^{start} + t_i^{pt}$ to $T_i^{start} + t_i^{pt} + t_i^{maxwait}$) is indicated by blue color on the time axis.

There are, however, some aspects of the algorithm that must be taken care of with the introduction of these arcs.

- The longest path algorithm must be adjusted accordingly. As the current implementation maintains a longest path matrix throughout the algorithm, there is no need to change anything with this.

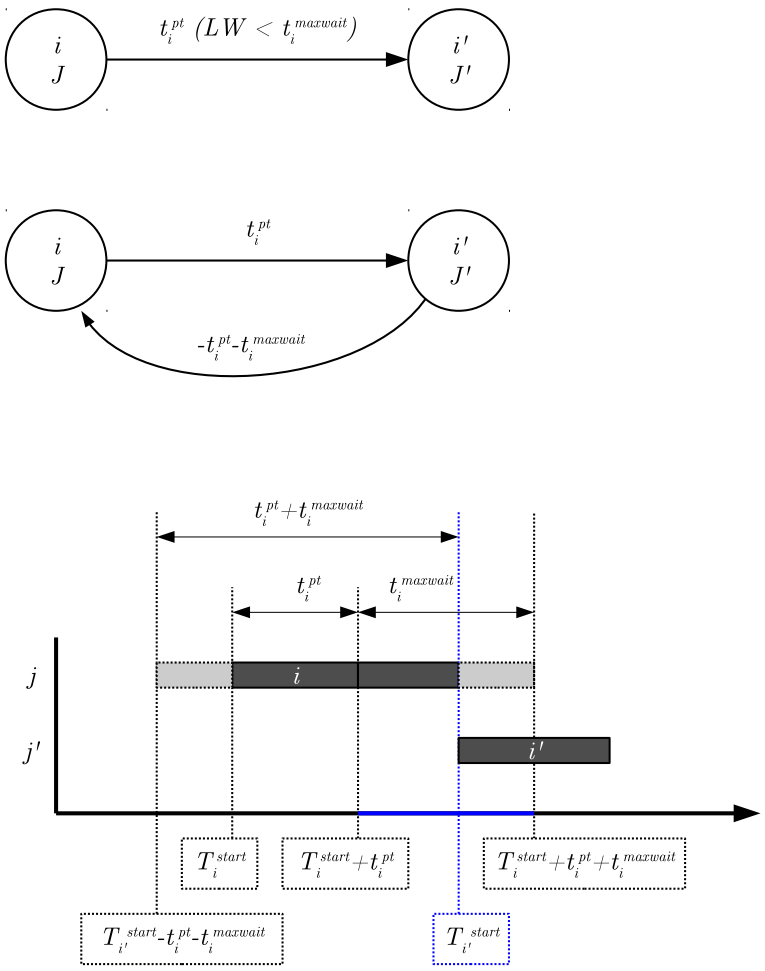


Figure 5.1: Modeling LW policy with negative weighted arcs

- Even the recipe graph will immediately contain several cycles. Their weight is negative, or in the case of ZW arcs, 0. Naturally, negative weighted cycles are not of great interest. They can simply be disregarded. However, the 0 weighted cycles need extra care, as some of them only represent a ZW connection, while the others model a cross transfer (See section 3.2). The algorithm must be modified in order to report only those zero-weighted cycles that do not have recipe arcs in them.

5.3 Combinatorial approach without negative weights

In this section two approaches are described to tackle ZW policy. LW stages are not considered, but they can be addressed via a modeling transformation described in the last subsection.

As briefly discussed before, a schedule can belong to one of the three groups listed below:

UW infeasible These schedules would be infeasible for the UW relaxation of the problem as well.

ZW infeasible, UW feasible These schedules are feasible for the UW relaxation but they violate ZW constraints

ZW feasible These schedules are feasible for the problem.

The original S-graph algorithm fails to differentiate the schedules in the second group from the ones in the third. It does, however, eliminate all the schedules in the first group. In order to identify schedules in the second group, the approach described in the previous section introduced additional, negative weighted arcs to the problem. The schedules in the second group would result in a non-negative weighted cycle by using that approach, which consists of two type of alternating parts:

- "Forward", positive arcs belonging to either UW or ZW intermediates
- "Backward", negative weighted arcs of ZW stages

An example is given in Figure 5.2 with both the schedule graph and the corresponding Gantt chart. The same schedule is infeasible if the outputs of tasks $i1, i2$, and $i3$ have ZW policy, as shown in Figure 5.3. The path indicated by thick arcs has a weight of 6, which is longer than the ZW path from $i2$ to $i4$, thus it results in a positive cycle.

5.3.1 Recursive search

Even if the negative weighted arcs are not inserted to the S-graph, the positive weighted cycles can be identified by a search[50]. Suppose that a partial schedule is ZW feasible, and a new schedule arc is just inserted to the graph. Obviously, if the partial schedule is now ZW infeasible, the newly added arc must be part of the non-negative cycle.

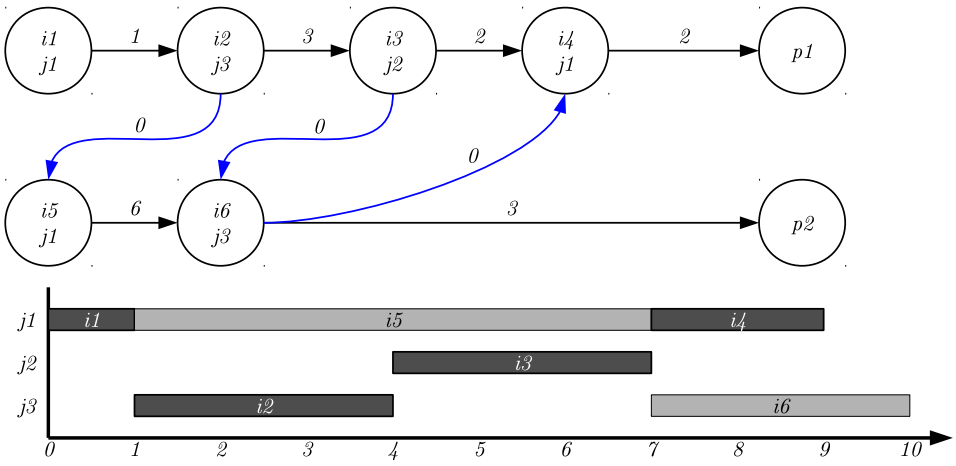


Figure 5.2: Feasible UW schedule

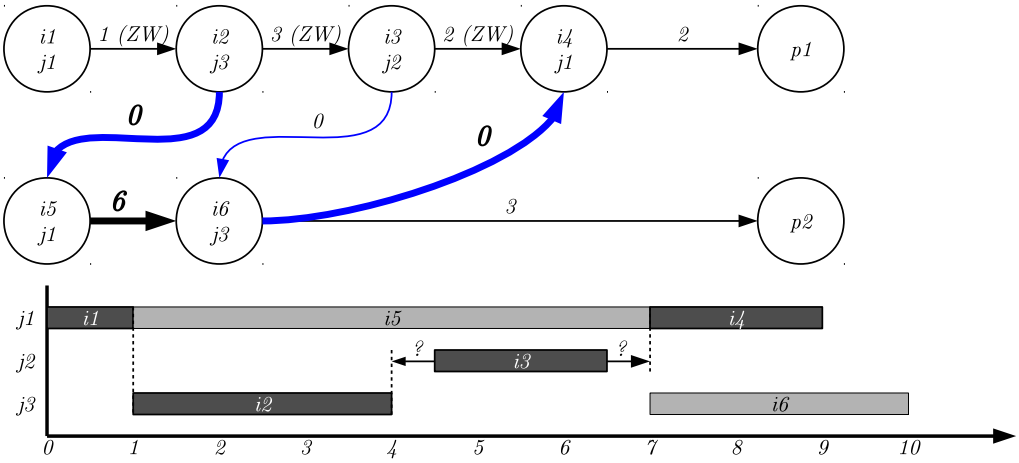


Figure 5.3: Infeasible ZW schedule

Even without the negative arcs, this cycle could be found by a simple approach: from the endpoint of the new arc, it has to be checked recursively whether the starting point of the schedule arc is reachable via "forward" and "backward" steps with a non-negative weight or not.[1]

The algorithm for that search is given in block 5.1.

This approach basically checks the existence of a non-negative cycle instead of maintaining the extended longest-path matrix of the previous approach. The algorithm looks for such nodes in the S-graph in a recursive way, whose starting time can be bounded with the starting time of i^d (the destination of the newly inserted schedule arc) from below. Nodes like that are added initially to the set *Unexamined* together with their lower bounds. In each iteration a node is selected from *Unexamined*, and if i^s (the source of the newly inserted schedule arc) is found among them with greater or equal lower bound then $-c$ (c is the weight of the newly inserted schedule arc) then the algorithm has found a positive weighted "cycle", and the partial schedule is infeasible. Otherwise, the node-bound pair is added to *Examined* and the neighbor vertices are added to *Unexamined*. The first `for all` loop adds the vertices that are subsequent via a ZW or UW arc, and the second loop looks backwards with only ZW arcs. In both of the loops three cases are investigated, and the sets are modified accordingly: i) when the node does not appear in either *Unvexamined* or *Examined*; ii) The node is waiting to be selected from *Unexamined* but with a weaker lower bound found on a different path previously; and iii) when the node was already examined but with a weaker bound. When all of the necessary nodes are examined, and no violation is reported, the feasibility of the partial schedule is ensured.

Note that this approach can easily be extended to LW policy by using the sum of the processing time, and maximal waiting time instead of $w(i', i)$ in the second loop. Though the model does not need to be extended with negative-weighted arcs, this function can take up a lot of cpu time during the optimization.

A way of accelerating this approach is to directly add the longest paths between those pairs, where this path is non-negative.

This technique is illustrated in Figure 5.4. The figure represents a part of a partial schedule, where the schedule arc $(i10, i6)$, indicated by blue color is recently added to the graph by adding $i6$ to the processing queue of $j3$ after $i9$. Some of the additional arcs that can be inserted by the algorithm are shown in the figure with green color. As an example, the arc between $i9$ and $i5$ with weight 3 is justified, as there is a "forward path" from $i9$ to $i6$ via $i10$, thus $i6$ must start at least 6 time units later than $i9$. Moreover, $i5$ can not start more than 3 time units earlier than $i6$, i.e., it must start at least $6 - 3 = 3$ time units later than $i9$.

[1] If the schedule arc has the weight of c a path with the weight of at least $-c$ is enough of course.

Algorithm 5.1 Finding non-negative path between two vertices for detecting ZW infeasibility

zw_feasibility_test$(G(N, A_1, A_2, w), i^s, i^d, c)$

$G(N, A_1, A_2, w)$ partially scheduled graph

i^s source of the newly inserted schedule arc

i^d destination of the newly inserted schedule arc

c weight of the newly inserted schedule arc

$Unexamined := \{(i^d, 0)\}$
$Examined := \emptyset$
while $Unexamined \neq \emptyset$ **do**
  Select $(i, w_i) \in Unexamined$ arbitrary
  if $i = i^s \wedge w_i \geq -c$ **then**
    return false
  end if
  $Unexamined := Unexamined \setminus \{(i, w_i)\}$
  $Examined := Examined \cup \{(i, w_i)\}$
  for all $(i, i') \in A_1 \cup A_2$ **do**
    if $\nexists (i', w') \in Unexamined \cup Examined$ **then**
      $Unexamined := Unexamined \cup \{(i, w_i + w(i, i'))\}$
    else if $\exists (i', w') \in Unexamined \wedge w' < w_i + w(i, i')$ **then**
      $Unexamined := Unexamined \setminus \{(i', w')\} \cup \{(i, w_i + w(i, i'))\}$
    else if $\exists (i', w') \in Examined \wedge w' < w_i + w(i, i')$ **then**
      $Examined := Examined \setminus \{(i', w')\}$
      $Unexamined := Unexamined \cup \{(i, w_i + w(i, i'))\}$
    end if
  end for
  for all $(i', i) \in A_1$, where (i', i) is a ZW arc **do**
    if $\nexists (i', w') \in Unexamined \cup Examined$ **then**
      $Unexamined := Unexamined \cup \{(i, w_i - w(i', i))\}$
    else if $\exists (i', w') \in Unexamined \wedge w' < w_i - w(i', i)$ **then**
      $Unexamined := Unexamined \setminus \{(i', w')\} \cup \{(i, w_i - w(i', i))\}$
    else if $\exists (i', w') \in Examined \wedge w' < w_i - w(i', i)$ **then**
      $Examined := Examined \setminus \{(i', w')\}$
      $Unexamined := Unexamined \cup \{(i, w_i - w(i', i))\}$
    end if
  end for
end while
return true

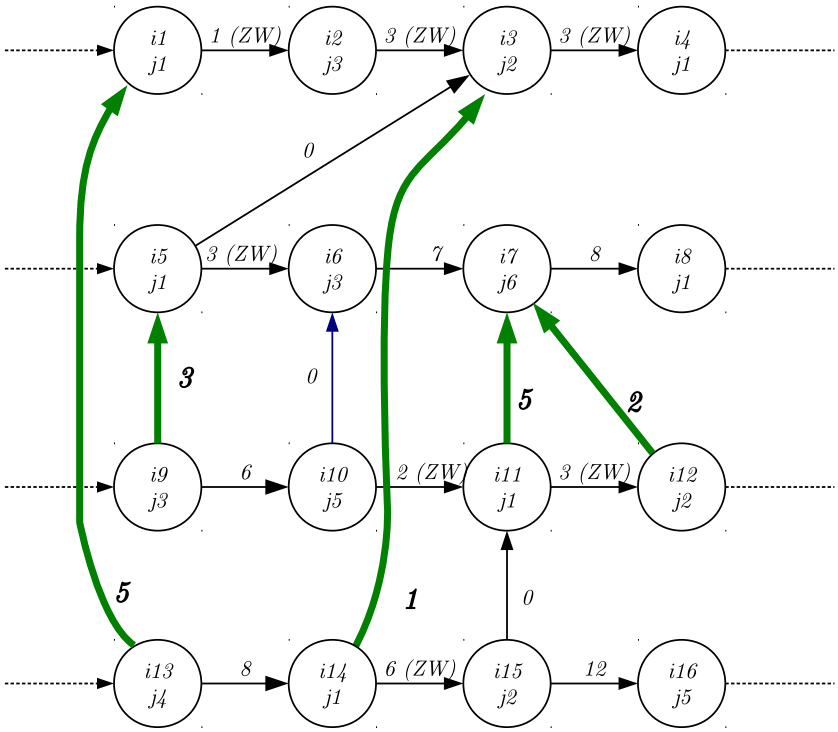


Figure 5.4: Acceleration by inserting additional arcs in the graph

5.3.2 Auxilary graph

The key idea behind the approach to be described in this section lies in the fact that the starting time of a task determines the starting time of all other tasks, which are "ZW-connected" to it, i.e., they are connected via one or several ZW stages. As an example, if the starting time of task $i11$ of Figure 5.4 is decided, so is the starting time of $i10$ and $i12$.

In this approach, the tasks are assigned to ZW classes before the optimization procedure starts. Let these classes be denoted by $Z_1, Z_2, \ldots$. Two tasks belong to the same class if and only if they are connected by a sequence of ZW arcs. For simpler notation, the class of task i is denoted by $Z(i)$. Moreover, each class has a "reference" task that is chosen arbitrary, and denoted by $i(Z_k)$. If task i belongs to the class with the reference task i^* , i.e., $i \in Z(i^*)$, then a relative distance, $d(i)$ is calculated for i that is the signed sum of the weight of the ZW arcs connecting i and i^*. (Starting from i^*, the arcs taken forward count as positive, the ones used backwards counts as negative.)

Finally, a new directed graph is constructed: $G^* = (\mathcal{Z}, A, w^*)$, where

$\mathcal{Z}$ is the set of ZW classes

$A = \{(Z, Z') \mid Z, Z' \in \mathcal{Z}\, \exists (i, i') \in A_1 \cap Z \times Z'\}$, i.e., there is an arc between those ZW classes, where there is a recipe arc between any of their tasks.

$w(Z, Z') = \max_{(i,i') \in A_1 \cap Z \times Z'} (d(i) + w(i, i') - d(i'))$, is the weight on the arc between Z and Z' if exists. It defines the relative minimal delay between the reference

points of the two ZW class.

An example for the recipe of the problem from Figures 5.2 and 5.3 is given in Figure 5.5.

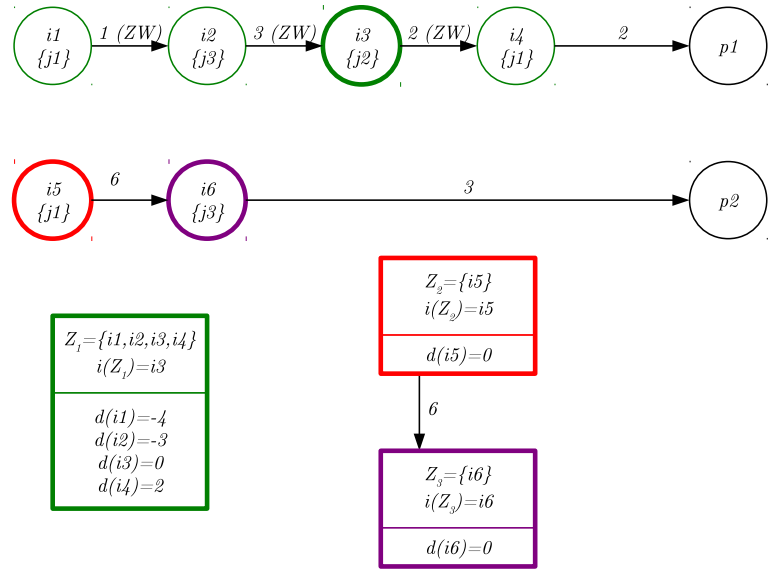


Figure 5.5: Example for the graph of ZW classes constructed from the recipe

There are three ZW classes indicated by colors green, red, and magenta, respectively. The first one contains 4 tasks, while the others are singletons. The reference points are indicated by thick lines in the recipe graph. Initially, there is only one arc, between Z_2 and Z_3 with weight 6 due to the recipe arc between $i5$ and $i6$.

After this graph has been created, the optimization procedure can start. At each iteration, when a new schedule arc (i, i') or (p, i') is added to the schedule graph with the weight of c, the following has to be done:

- If $Z(i) = Z(i')$ and $d(i) + c > d(i')$, then the partial schedule is ZW infeasible, it has to be pruned from the B&B tree.

- If $Z(i) \neq Z(i')$, then add $(Z(i), Z(i'))$ to A with the weight of $d(i) + c - d(i')$. If there was a previous arc between these two vertices, the one with the higher weight is to be kept.[2]

- If the schedule arc starts from a product, i.e., (p, i') is inserted, then for all $(i, p) \in A_1$ the previous two steps are to be followed, as if the arc (i, i') would have been inserted with the weight $w(i, p) + c$.

- If the graph G^* contains a directed cycle with non-negative weight, the schedule is ZW infeasible.

[2]Deleting the one with the smaller weight is optional, it does not change the soundness of the approach.

Using the schedule from Figure 5.3, the graph G^* will be extended by 3 additional arcs, as shown in Figure 5.6. It is easy to see, that there is a cycle of weight 1 between $Z_1 \to Z_2 \to Z_3 \to Z_1$, which indicates the infeasible schedule.

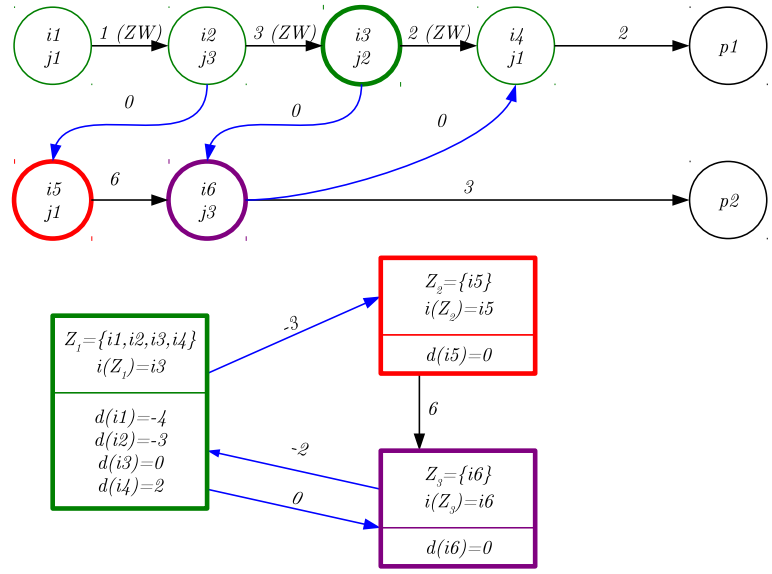


Figure 5.6: Example for the extended graph of ZW classes based on a schedule

5.3.3 Model-level conversion of LW problems to ZW

The approaches in the previous two subsections were developed to solve problems where each material has either ZW or UW policy. As it has been noted at the first approach, they can be modified to tackle LW policy as well. There is, however, an other way of solving problems with LW policy. After a simple transformation on the recipe graph, these approaches can solve the problems without any modifications.

The key idea behind the method is introducing additional vertices and arcs for each LW task. If there is a LW material between tasks i and i', then two additional vertices are inserted into the graph, i_c and i_w which represents the completion and the weiring of the LW material respectively, as shown in Figure 5.7. Moreover, several arcs are inserted into the graph:

(i, i_c) is a ZW recipe arc with the weight of t^{pr},i.e., the processing time of i. This arc ensures, that the i_c, i.e., the completion of the intermediate material is exactly t^{pr} later than the start of the execution of task i. Similarly,

(i_c, i_w) is a ZW recipe arc with the weight of t^w, that is, the maximal waiting time possible after the completion of the intermediate.

$(i\_c, i'), (i', i\_w)$ are zero-weighted schedule arcs that ensure that i' must start between the completion and wearing of the intermediate.

(i, i') is a UW recipe arc with weight of t^{pr}. This arc is needed so that the schedule arc towards the task that follows i in the same unit could start from i' not only from i_c.

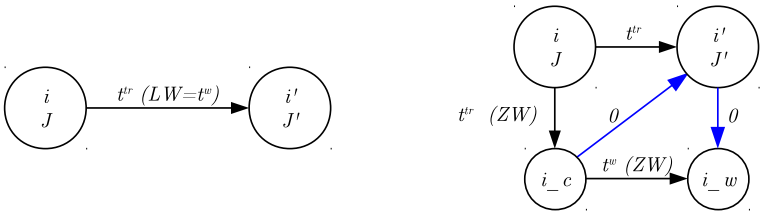


Figure 5.7: Example for model transformation of LW stages

5.4 Comparison of approaches

The efficiency of the algorithms from the previous sections is illustrated on a literature example. The example features 5 different sequential products and 6 units to be scheduled. The recipe graph for the example is shown in Figure 5.8.

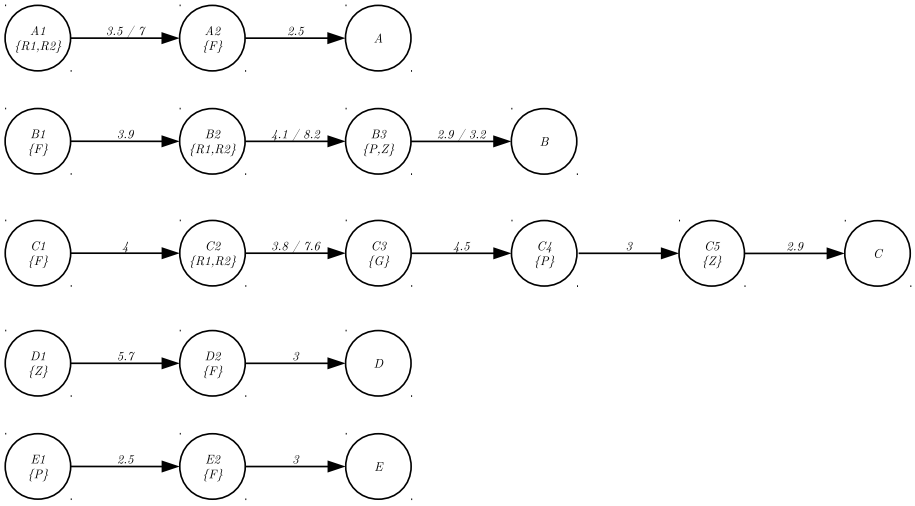


Figure 5.8: Example for the comparison of LW/ZW approaches

For the sake of the comparison, each intermediate is assumed to have ZW storage policy. The investigated algorithms were compared on 14 different configurations with batch numbers.

These algorithms are:

sLP is the simple LP approach

aLP is the advanced LP approach that copies the model for each subproblem

aLP' is the advanced LP approach that uses only one model as a global variable

Neg is the approach with negative weighted arcs

Rec is the approach relying on the recursive search

Rec+ is the recursive approach with the extended positive arc additions

Each test run were set within a 1000 s time limit. For the 11 smaller configurations, the approaches have not reached this limit except for couple of cases, and were able to find the optimal solution. The data for the CPU times of the approaches are given in detail in Section C.2, and illustrated in Figure 5.9

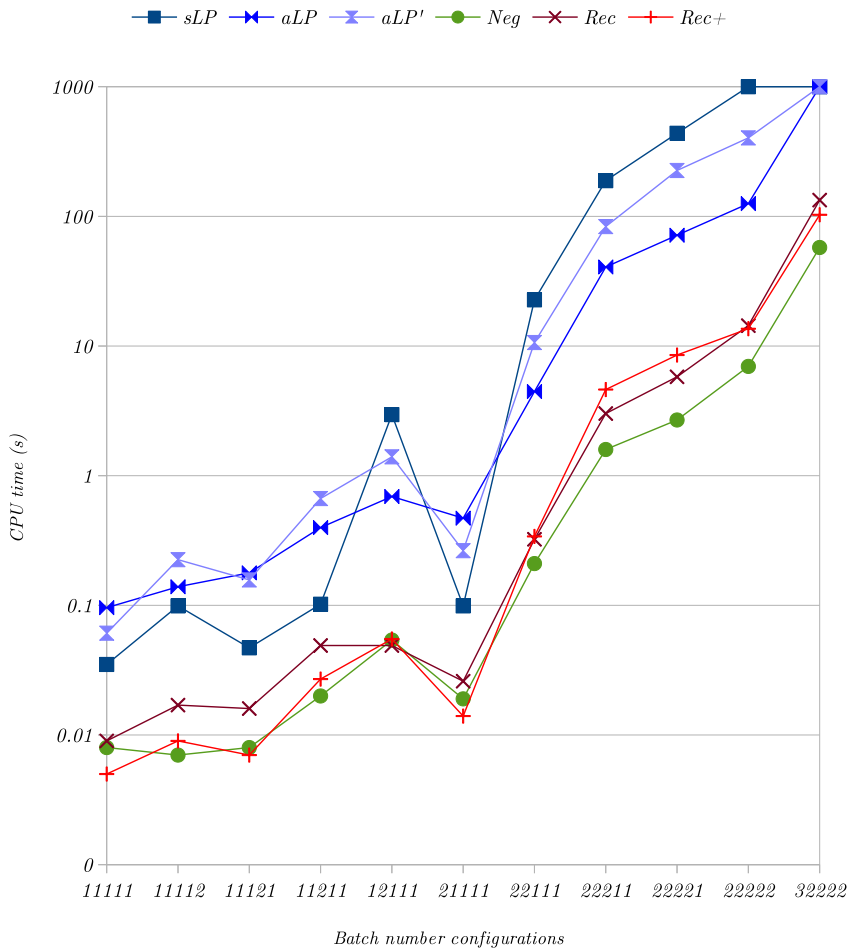


Figure 5.9: CPU time of ZW/LW approaches for the smaller cases

The LP based approaches were usually 1 or 2 magnitudes slower than the combinatorial approaches, among which the negative arc based proved to be the most efficient.

For the 3 larger configurations all of the approaches have reached the 1000 second time limit, and thus stopped. For these cases the comparison of the quality of the best found

(probably suboptimal) solution is important, and shown in Figure 5.10. The figure also includes the 2 largest configurations from the previous ones, where the LP based approaches have reached the limit.

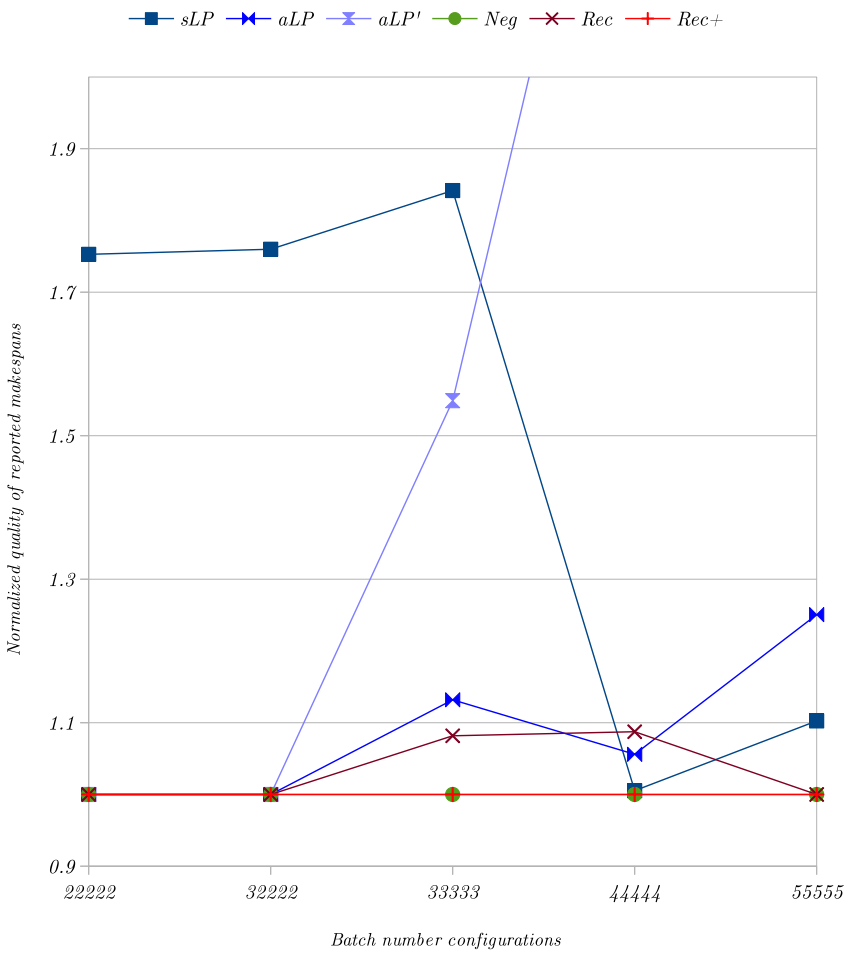


Figure 5.10: Quality of reported solutions of ZW/LW approaches for the larger cases

The **aLP'** approach could not even report a feasible schedule for the larger cases. The best solutions were reported by the negative arc based approach. The extended recursive approach had very close results. The other approaches fluctuated.

In general, it can be stated that the most favorable approach seems to be the one using negative weighted arcs.

Summary and concluding remarks

The original algorithms of the S-graph framework was developed for Unlimited Wait storage policies. In this chapter, different options of extending the framework for Limited- and Zero-Wait storage policies has been presented and investigated. These options were implemented

and compared on a wide set of examples in terms of computational needs. The results showed that an extended model with negative-weighted arcs, and slightly modified algorithms is the most efficient option.

Related publication

- Hegyhati, M., T. Holczinger, A. Szoldatics, F. Friedler, Combinatorial Approach to Address Batch Scheduling Problems with Limited Storage Time, Chemical Engineering Transactions, 25, 495-500 (2011)

Related conference presentations

- Hegyhati, M., T. Holczinger, A. Szoldatics, F. Friedler, Combinatorial Approach to Address Batch Scheduling Problems with Limited Storage Time, presented at: PRES'11, Florence, Italy, May 8-11, 2011.
- Hegyhati, M., T. Majozi, F. Friedler, Throughput maximization in a multipurpose batch plant, presented at: PRES 2008, Prague, Czech Republic, August 24-28, 2008

Chapter 6

Maximizing expected profit in a stochastic environment

In Chapter 4 the general algorithm for throughput maximization was introduced. For many problems, however, several parameters of these types of problems are not deterministic. Pistikopoulos et al.[104] gave a classification of stochastic scheduling problems based on the source of the uncertainty in the process. In a changing market environment, for example, the price and marked demands can usually be considered stochastic. Both of these values can have impact on the overall profit, as in case of a overproduction, for example, the storage of the surplus may result in additional expenses.

Obviously, in such uncertain circumstances, it becomes ambiguous which solution can be considered as optimal: the most robust one, or the one with highest expected profit, etc. Li and Ierapetritou [76] gave an extensive review of the approaches that deal with different types of uncertainties in batch process scheduling. Without attempting to be comprehensive, the three main direction of research is illustrated in Figure 6.1.

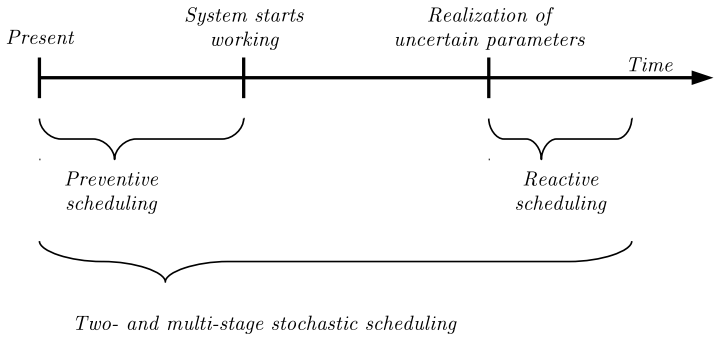


Figure 6.1: Classification of approaches dealing with uncertainty

Preventive scheduling In this direction of research[72, 14], the schedule must be given a-priori, and cannot be modified after the system starts, or the realization of uncertain events. If there is no information about the probability distribution

of the uncertain parameters, a reasonable objective could be a production schedule, that exceeds a certain profit, and have the highest level of robustness. On the other hand, an estimation about the probability distribution gives room for optimization towards the highest expected profit.

Reactive scheduling Reactive approaches[92] assume an existing schedule, which gets disturbed by some uncertain event. The objective is to modify the rest (unexecuted part) of the schedule in a way, that leads to the best available performance. Unlike in the previous case, the optimization is carried out, when the system is already running, thus usually there is a very limited amount of time to deliver the solution. As a result, heuristics are often favored for this purpose. On the bright side, the approach does not have to deal with any uncertain parameters, as they are already realized.

Two- and multistage approaches In a two-stage optimization approach[59], it is assumed that there are some decisions that must be done prior to the start of the system, however, some decisions can be altered later. As an example, a schedule may has to be decided in advance, but the load of the units can be adjusted after the uncertain parameters occur. In a multi-stage approach, this concept is generalized, and it is not assumed, that all of the uncertain events realize at the same time.

Naturally, there are a lot of developments, which do not fit into these three categories, some approaches, e.g., combine preventive and reactive scheduling. Sensitivity analysis in itself is a broad area to research, but it can also participate as a subroutine for preemptive scheduling, especially in case of heuristic approaches.

The following sections introduce an S-graph based approach for the scheduling of throughput maximization problems considering uncertain cost and demand parameters with discrete probability distribution functions. Section 6.1 gives an exact definition of the problems to be solved, and Section 6.2 and 6.3 presents the S-graph approaches to address this set of problems. In Section 6.4 an extension is shown to continuous probability distribution functions. Last, Section 6.5 illustrates some of the algorithms via an example, and draws some conclusions.

6.1 Problem definition

The problems to be solved are given by similar parameters to those of a general throughput maximization problem, i.e., each product is given with its recipe, along with the set of equipment units and the time horizon. At this point it is assumed that there is a one to one relation between products and recipes, i.e., there is no recipe producing multiple products, and there are no two different recipes producing the same product.

There is, however, a set of uncertain parameters for each product, whose probability distribution is discretized into joint scenarios. Thus, for each discrete scenario the following parameters are given:

- probability of the scenario
- for each product:
  - price for one batch
  - demand
  - over- and underproduction cost

The objective is to make decision about the number of batches and provide a feasible schedule in way to achieve maximal expected profit. Note that it is assumed that if the actual production is higher than the demand, the surplus is not sold.[1]

Based on the possible decisions related to the sizes of each batch, three different problems are identified:

Preventive problem with fixed batch sizes when the batch size for each product is given, and the only preventive decision to be made is to decide the number of batches for them.

Preventive problem with variable batch sizes is a more flexible version of the previous problem, where not only the number of batches, but also their sizes can be altered in advance before the uncertain events realize.

Two stage problem where the batch numbers have to be decided in advance, but the batch sizes can be altered accordingly after the uncertain events realized.

Section 6.2 introduces the approaches that can solve these three different problems. In Section 6.3, these approaches are extended to tackle cases, when a recipe can produce several products, and the same products can be produced by several recipes. [2].

6.2 S-gaph based approaches

The three approaches to solve the problems presented in the previous section are all based on the throughput maximization algorithm presented in Section 4.1. The main algorithm that examines different batch size configurations and the feasibility tester can remain the same, as in the case of the general throughput maximization problems. The subroutines to select a configuration and to update the set of open configurations can be altered in order to

[1] Taking the opposite assumption would not alter the structure of the algorithm, the same approaches would be applicable with modified input parameters.

[2] The approach for the preventive version of this case has been published in the literature[72]

achieve higher efficiency, however, any of the previously mentioned implementations would suffice.

The key difference between the different approaches is the **revenue** function, that should provide the expected profit, or the highest expected profit for a configuration, as it will be discussed in the following subsections.

In order to simplify these descriptions, the following additional notations will be used throughout this chapter next to the ones used previously:

P the set of products

b_p the number of batches for product p in the actual configuration

s_p the size of a batch for product p (for the first case)

s_p^{min}, s_p^{max} the minimal and maximal size of a batch for product p (for the second and third case)

S the set of scenarios

$prob_s$ the probability of scenario $s \in S$

$dem_{s,p}$ the demand for product p in scenario $s \in S$

$price_{s,p}$ the price of p in scenario $s \in S$

$oc_{s,p}, uc_{s,p}$ the over- and underproduction cost of p in scenario $s \in S$

For further simplification, the function $Profit_{s,p}(x)$ is introduced, that gives the profit for x amount of product p in scenario s. The function is calculated as follows:

$$Profit_{s,p}(x) = \begin{cases} price_{s,p} \cdot x - (dem_{s,p} - x) \cdot uc_{s,p} & \text{if } x < demand_{s,p} \\ price_{s,p} \cdot dem_{s,p} - (x - dem_{s,p}) \cdot oc_{s,p} & \text{otherwise} \end{cases}$$

6.2.1 Preventive scheduling with fixed batch sizes

In this case, the only decision to be made is the number of batches for each product, i.e., finding the optimal feasible configuration for the number of batches. Thus, there is no decision to be made for a single configuration, and the calculation of the expected profit is rather simple:

$$\sum_{s \in S} \left(prob_s \cdot \sum_{p \in P} Profit_{s,p}(s_p \cdot b_p) \right)$$

As this **revenue** function is really simple to evaluate, the **update** function can similarly remove the batch number configurations that do not provide higher expected profit than the current best solution. In order to do this the revenue function of each configuration

need to be calculated in advance, as the configurations having the same expected profit do not lie on a line, plane, etc. Moreover, if these values are calculated in advance, the list of open configurations can be ordered in decreasing order, and it could be an efficient selection strategy to find the first feasible configuration there. This technique requires, however, that the number of open subproblems should be finite, so the initial steps described in Subsection 4.2.1 must be carried out.

An additional modification can be applied here: as the batch number of a product has no influence on the number of batches for an other product, and thus on its profit, the initial steps along the axises can also stop, when the revenue function stops increasing. To back this statement, the above expression is reformulated in this equivalent form:

$$\sum_{p \in P} \left(\sum_{s \in S} prob_s \cdot profit_{s,p}(s_p \cdot b_p) \right)$$

The expression in the braces is the expected profit of $s_p \cdot b_p$ amount of product p. As this expression will be used in the following subsection as well, an additional notation $ExpProfit_p(x)$ is introduced:

$$ExpProfit_p(x) = \sum_{s \in S} prob_s \cdot profit_{s,p}(x)$$

Note that the expected profits given by the different products are independent, as none of them shares a recipe. Thus, if there are two configurations with the same number of batches for a certain product, increasing or decreasing it will have the same effect on the expected profit, regardless of the number of batches from the other products.

In order to prove the soundness of the aforementioned initial configuration selection strategy, it has to be shown, that the $ExpProfit$ function will never increase, after it started decreasing. It is easy to see that $ExpProfit$ is a continuous, piecewise linear function, thus it is enough to show, that the slope is always decreasing. At minus infinity the slope is $\sum_{s \in S} prob_s \cdot (price_{s,p} + up_{s,p})$, and when the function passes a demand value, dem_s, the slope decreases by $price_{s,p} + up_{s,p} + op_{s,p}$. As all these three parameters are considered to be non-negative, the above mentioned technique is justified.

6.2.2 Preventive scheduling with variable batch sizes

Unlike in the previous case, the number of batches does not determine the amount of the product to be produced, it is another decision to be made by the optimizer. For each batch, the amount should be between s_p^{min} and s_p^{max}. Without the loss of generality, it can be assumed that the batches belonging to the same product have the same batch size, as all the other solutions can be converted to such without changing the expected profit.

Thus, the only decision to be made at a batch number configuration is choosing the amount of product, x_p to be produced from the interval $[b_p \cdot s_p^{min}, b_p \cdot s_p^{max}]$. Since the

products are still independent from each other, and the expected profit can be expressed as $\sum_{p \in P} ExpProfit_p(x_p)$, x_p needs to be the maximizer point of $ExpProfit$ in the $[b_p \cdot s_p^{min}, b_p \cdot s_p^{max}]$ interval.

As it was discussed in the previous subsection, $ExpProfit$ is a piecewise linear function with decreasing slope, thus the funtion takes its maximal value at one of the demand values, let it be $dem_{s'}$.[3] Now, the selection of the optimal x_p is straight-forward:

$$x_p(b_p) = \begin{cases} b_p \cdot s_p^{max} & \text{if } b_p \cdot s_p^{max} < dem_{s'} \\ dem_{s'} & \text{if } b_p \cdot s_p^{min} \leq dem_{s'} \leq b_p \cdot s_p^{max} \\ b_p \cdot s_p^{min} & \text{if } b_p \cdot s_p^{min} > dem_{s'} \end{cases}$$

After identifying the x_p values for all b_p, the same initialization technique, and batch number configuration selection strategy can be used as in the previous case.

6.2.3 Two stage approach

In the previous subsection, the amount of product p to be produced, had to be given a priori. In this section it is assumed, that this decision can be made after the realization of the uncertain events, i.e., when the scenario is already known. For a batch number configuration, the value of the produced amount will depend on the scenario to happen. Let it be denoted by $x_{s,p}$.

Selection of $x_{s,p}$ is straight forward:

$$x_{s,p}(b_p) = \begin{cases} b_p \cdot s_p^{max} & \text{if } b_p \cdot s_p^{max} < dem_s \\ dem_s & \text{if } b_p \cdot s_p^{min} \leq dem_s \leq b_p \cdot s_p^{max} \\ b_p \cdot s_p^{min} & \text{if } b_p \cdot s_p^{min} > dem_s \end{cases}$$

Based on this, the expected profit for a batch configuration can be calculated as follows:

$$\sum_{p \in P} \left(\sum_{s \in S} prob_s \cdot Profit(x_{s,p}(b_p)) \right)$$

Again, the calculation is rather simple, and can be carried out in advance, in order to generate an ordered list of batch number configurations.

[3] If a slope becomes 0, there are infinitely many maximizer points, in this case any of them can be arbitrary chosen instead of $dem_{s'}$.

6.3 Extended approaches for recipes with multiple products

In this section the assumption on the one-to-one relation between products and recipes is expunged, which - as it will be shown later - brings an additional complexity in the optimization process. For the sake of simpler description of the approaches, the following additional notations are introduced:

R set of recipes

P_r set of products produced by recipe $r \in R$

R_p set of recipes producing product $p \in P$

b_r the number of batches for recipe $r \in R$ in the actual configuration

$s_{r,p}$ the maximal amount of $p \in P$ that can be produced with the recipe $r \in R$

min_r the minimal proportion ratio on which the recipe $r \in R$ can be executed

Several recipes producing the same product often appear when they are generated with the algorithm described in 4.3. However, this does not increase the computational complexity in any way if there are no recipes producing several products, as the approaches remain sound by the following modifications.

- In the preventive case with fixed batch sizes, the produced amount is $\sum_{r \in R_p} b_r \cdot s_{r,p}$ instead of $b_p \cdot s_p$,
- In the other two cases the interval from which x_p and $x_{s,p}$ should be chosen is $\left[\sum_{r \in R_p} min_r \cdot b_r \cdot s_{r,p}, \sum_{r \in R_p} b_r \cdot s_{r,p}\right]$. Based on the x_p and $x_{s,p}$ values, the sizes of the batches for the routes can be calculated.

The key point is, that even though the recipes are not homogeneous for a product, the produced amounts are still independent. This, however, does not remain true when a recipe is producing several different products. Increasing the batch size to satisfy the demands of a product can cause additional costs if there is a surplus already from the other product produced in the same recipe.

6.3.1 Preventive scheduling with fixed batch sizes

When the batch sizes are fixed, there is no additional decision to be made, just as in the previous section. The expected profit can be evaluated and returned.

6.3.2 Deterministic scheduling for variable batch sizes

Before investigating the stochastic case, it is advantageous to solve the problem with complex recipes for the deterministic case, i.e., given a batch number configuration and the value of demand, price, over- and underproduction cost parameters, what are the optimal batch sizes to maximize the profit.

This question already requires more sophisticated tools, it can be answered by using an LP model for example. The continuous non-negative x_r variables represent the decision about the sizing of all of the recipes of type r. These variables must take values between the possible range of a recipe, i.e.,

$$min_r \leq x_r \leq 1 \qquad\qquad r \in R$$

Two additional variables are introduced: x_p^{op}, x_p^{up} for the over- and underproduction of each product, respectively.

Having these variables, the following goal programming constraints express the relation between the produced amount of a product, the over- and underproduction and the demand:

$$x_p^{up} - x_p^{op} + \sum_{r \in R_p} b_r \cdot s_{r,p} \cdot x_r = dem_p \qquad\qquad \forall p \in P$$

For the sake of simpler description, an additional variable can be introduced for the produced amount of a product that will not make the model more complex, as it is a linear combination of the previous variables:

$$x_p^{pr} = \sum_{r \in R_p} b_r \cdot s_{r,p} \cdot x_r \qquad\qquad p \in P$$

And then the objective function can be expressed as:

$$\sum_{p \in P} ((x_p^{pr} - x_p^{op}) \cdot price_p - x_p^{op} \cdot oc_p - x_p^{up} \cdot uc_p) \rightarrow \max$$

Note, that the number of variables is equal to the number of recipe types plus twice the number of products[4], which is usually small, thus solving this LP model at each batch number configuration does not require a vast amount of time.

6.3.3 Preventive scheduling with variable batch sizes

In the stochastic case when there are several scenarios, with different parameters and probabilities, the model must be changed, as follows:

The x_r (and thus the x_p^{pr}) variables remain the same, as they are first-stage, and the

[4]not counting the x_p^{pr} variables, as they are removed immediately by the LP solver.

following constraints are also unaltered:

$$min_r \leq x_r \leq 1 \qquad r \in R$$

$$x_p^{pr} = \sum_{r \in R_p} b_r \cdot s_{r,p} \cdot x_r \qquad p \in P$$

The over- and underproduction, however, will be different for each scenario, thus the introduction of variables $x_{s,p}^{op}$ and $x_{s,p}^{up}$ is necessary. The balance constraint must be changed accordingly:

$$x_{s,p}^{up} - x_{s,p}^{op} + x_p^{pr} = dem_{s,p} \qquad \forall s \in S, p \in P$$

Similarly, the objective function must express the expected profit, thus:

$$\sum_{s \in S} prob_s \cdot \sum_{p \in P} ((x_p^{pr} - x_{s,p}^{op}) \cdot price_{s,p} - x_{s,p}^{op} \cdot oc_{s,p} - x_{s,p}^{up} \cdot uc_{s,p}) \rightarrow \max$$

The number of continuous variables increased to $|R| + 2 \cdot |S| \cdot |P|$, thus, as the number of scenarios increase, i.e., discretization gets smoother, the LP models got more difficult to solve. It would need, however, a large number of scenarios to make this solution time comparable to that of the feasibility test of a batch number configuration.

It is possible to reduce the search space of the LP models by eliminating a simple redundancy: since the size of the batches for each recipe can be chosen between min_r and 1, it is possible in two neighbor configurations (only one of the batch numbers differs by 1) to have the same LP optima, with the same x_p^{pr} values. The values taken by the x_r variables are of course different, but the solution is essentially the same, the bigger configuration is producing the same amount of products with more and less loaded batches. If no other parameters are included (load dependent energy cost, cleaning costs after a certain amount of idle state of a unit, etc.) there is no rational reason for the solution with the larger configuration, as its schedule (if even feasible) would be more dense with providing the same expected profit. This type of redundancy could be avoided by a simple inequality:

$$x_r \geq \frac{b_r - 1}{b_r} \qquad r \in R$$

6.3.4 Two stage approach

When the sizes of the batches can be decided after the realization of the scenario, the corresponding variables become second-stage, i.e., $x_{s,p}$ is introduced, and the constraints, objective function must change accordingly:

$$min_r \leq x_{s,r} \leq 1 \qquad s \in S, r \in R$$

$$x_{s,p}^{pr} = \sum_{r \in R_p} b_r \cdot s_{r,p} \cdot x_{s,r} \qquad s \in S, p \in P$$

$$x^{up}_{s,p} - x^{op}_{s,p} + x^{pr}_{s,p} = dem_{s,p} \qquad \forall s \in S, p \in P$$

$$\sum_{s \in S} prob_s \cdot \sum_{p \in P} ((x^{pr}_{s,p} - x^{op}_{s,p}) \cdot price_{s,p} - x^{op}_{s,p} \cdot oc_{s,p} - x^{up}_{s,p} \cdot uc_{s,p}) \to \max$$

The number of LP variables in this case is $|S| \cdot (|R| + 2 \cdot |P|)$, and the space reduction technique from the previous subsection cannot be applied here.

6.4 Continuous probability distribution

In many applications, the stochastic parameters are not given by a discrete distribution function, but with a continuous one, e.g., normal or uniform distribution. In this case, an option is to discretize this function, and use the approaches described in the previous two sections. In this case, the discretization must be smooth in order to achieve accurate results. However, too large number of discrete scenarios may increase the computational needs significantly, especially in the case of the two-stage approach with complex recipes. Another way is to tackle the problem directly in its continuous form.

In this section the following assumptions are taken:

- Several recipes may produce the same product, but no recipe produces several products.
- The only stochastic parameters are the demands for each products, which are given by invertible cumulative distribution functions.
- The prices, over- and underproduction costs are deterministic.

The approaches to solve problems with this type of uncertainty are similar to that of the approaches in Sections 6.2 and 6.3, as they rely on the same evaluation of batch number configurations, and an altered revenue procedure.

6.4.1 Preventive case with fixed batch sizes

If the batch sizes are fixed, there is no further decision to be made for a batch number configuration, the expected profit for a batch configuration can be calculated based on the following expression:

$$\sum_{p \in P} \left(\int_{d=0}^{x_p} f(d) \cdot (d \cdot price_p - (x_p - d) \cdot oc_p) \, \mathrm{d}d + \int_{d=x_p}^{\infty} f(d) \cdot (x_p \cdot price_p - (d - x_p) \cdot uc_p) \, \mathrm{d}d \right)$$

Where x_p is the produced amount from product p, and as previously,$x_p = \sum_{r \in R} s_{p,r} \cdot b_r$, and f is the probability distribution function of the demand.

6.4.2 Preventive case with flexible batch sizes

If the batch sizes are flexible, it is not trivial to chose the load with maximal expected profit. Fortunately, the purpose of the *Newsvendor model*[29, 10] is to tackle a similar problem, where a vendor must decide on the amount of a product to be produced (x), without knowing the exact demand (d) for it, only its cumulative probability distribution function (F).[5] There is a cost of producing a single product (c), and a price for which it can be sold (p). The expected profit can be expressed as:

$$E\left(p \cdot \min(x, d)\right) - x \cdot c$$

In this case, the value of x for which the above expression takes its maximal value is:

$$x^* = F^{-1}\left(\frac{p-c}{p}\right)$$

The newsvendor model can be extended with a salvage value s for the surplus production, changing the expected profit to:

$$E\left(p \cdot \min(x, d)\right) + E\left(s \cdot \max(x-d, 0)\right) - x \cdot c$$

In this case the optimal quantity to be produced is at

$$x^* = F^{-1}\left(\frac{p-c}{p-s}\right)$$

Note, that this case in fact has the same complexity as the previous one. Increasing all the parameters (p, c, s) with the same value, the expected profit, and the optimal solution will not change. In the simpler model, the cost of the surplus production is c, which is equivalent of the overproduction cost in our case. The profit for the production below the demand is $p - c$, from these:

- $p = price_p + oc_p$
- $c = oc_p$

This reformulation is equivalent to the original problem, except that the underproduction cost is not considered. Obviously, the newsvendor model can be applied if $uc_p = 0$, but that was not assumed in the problem definition above.

If $uc_p = 0$, the problem can be converted to a nearly equivalent one:

- $oc'_p = oc_p$
- $uc'_p = 0$
- $price'_p = price_p + uc_p$

[5] This function is assumed to be invertible.

In this altered problem, there is no underproduction cost, and the expected profit after choosing to produce x_p amount of p is:

$$
\begin{aligned}
& \int_{d=0}^{x_p} f(d) \cdot (d \cdot price'_p - (x_p - d) \cdot oc'_p) \, \mathrm{d}d + \int_{d=x_p}^{\infty} f(d) \cdot (x_p \cdot price'_p - (d - x_p) \cdot uc'_p) \, \mathrm{d}d \\
= & \int_{d=0}^{x_p} f(d) \cdot (d \cdot (price_p + uc_p) - (x_p - d) \cdot oc_p) \, \mathrm{d}d + \int_{d=x_p}^{\infty} f(d) \cdot (x_p \cdot (price_p + uc_p)) \, \mathrm{d}d \\
= & \int_{d=0}^{x_p} f(d) \cdot (d \cdot price_p - (x_p - d) \cdot oc_p) \, \mathrm{d}d + \int_{d=0}^{x_p} f(d) \cdot d \cdot uc_p \, \mathrm{d}d \\
& + \int_{d=x_p}^{\infty} f(d) \cdot (x_p \cdot price_p + x_p \cdot uc_p)) \, \mathrm{d}d \\
= & \int_{d=0}^{x_p} f(d) \cdot (d \cdot price_p - (x_p - d) \cdot oc_p) \, \mathrm{d}d + \int_{d=0}^{x_p} f(d) \cdot d \cdot uc_p \, \mathrm{d}d \\
& + \int_{d=x_p}^{\infty} f(d) \cdot (x_p \cdot price_p + d \cdot uc_p x_p \cdot uc_p - d \cdot uc_p)) \, \mathrm{d}d \\
= & \int_{d=0}^{x_p} f(d) \cdot (d \cdot price_p - (x_p - d) \cdot oc_p) \, \mathrm{d}d + \int_{d=0}^{x_p} f(d) \cdot d \cdot uc_p \, \mathrm{d}d \\
& + \int_{d=x_p}^{\infty} f(d) \cdot (x_p \cdot price_p + (d - x_p) \cdot uc_p) \, \mathrm{d}d + \int_{d=x_p}^{\infty} f(d) \cdot d \cdot uc_p \, \mathrm{d}d \\
= & \int_{d=0}^{x_p} f(d) \cdot (d \cdot price_p - (x_p - d) \cdot oc_p) \, \mathrm{d}d + \int_{d=x_p}^{\infty} f(d) \cdot (x_p \cdot price_p + (d - x_p) \cdot uc_p) \, \mathrm{d}d \\
& + \int_{d=0}^{x_p} f(d) \cdot d \cdot uc_p \, \mathrm{d}d + \int_{d=x_p}^{\infty} f(d) \cdot d \cdot uc_p \, \mathrm{d}d \\
= & \int_{d=0}^{x_p} f(d) \cdot (d \cdot price_p - (x_p - d) \cdot oc_p) \, \mathrm{d}d + \int_{d=x_p}^{\infty} f(d) \cdot (x_p \cdot price_p + (d - x_p) \cdot uc_p) \, \mathrm{d}d \\
& + \int_{d=0}^{\infty} f(d) \cdot d \cdot uc_p \, \mathrm{d}d
\end{aligned}
$$

This expected profit differs only by $\int_{d=0}^{\infty} f(d) \cdot d \cdot uc_p \, \mathrm{d}d$ from the original expected profit, and it is not dependent on x_p. Thus, this problem will take its maximum for exactly the same values as the original. Subsequently, the newsvendor model can be applied with the following parameters:

- $p = price'_p + oc'_p = price_p + oc_p + uc_p$
- $c = oc'_p = oc_p$

And thus, the optimal amount for production is:

$$x^* = F^{-1}\left(\frac{price_p + oc_p + uc_p - oc_p}{price_p + oc_p + uc_p}\right) = F^{-1}\left(\frac{price_p + uc_p}{price_p + oc_p + uc_p}\right) = F^{-1}\left(1 - \frac{oc_p}{price_p + oc_p + uc_p}\right)$$

6.5 Test results and coments

In this section the algorithms with multiple products and discrete scenarios are illustrated via the same example, that has been used in Section 4.3 and Subsection 4.4.3.

The STN recipe for the problem is shown in Figure 1.5 with stoichiometric data. As discussed in Section 4.3, 6 different fixed recipes can be generated based on the capacities of the four available units. These recipes and their maximal revenue are shown in Figure 4.5, considering 10 cu/kg price[6] for both of the products.

In the stochastic case, it is assumed, that the price of the products do not change in the different scenarios, neither do the under- and overproduction costs, which are 1.5 and 2.5 cu/kg, respectively.

The time horizon is set to 18 hours, and six different scenarios are assumed for the demands of the products, as given in Table 6.1

Scenario	Probability	Demand (kg)	
		P1	P2
SC1	0.167	102.3	174.8
SC2	0.167	148.8	344.2
SC3	0.167	158.6	128.2
SC4	0.167	0.0	225.1
SC5	0.167	72.0	109.1
SC6	0.167	54.6	268.8

Table 6.1: Scenarios for the illustrative example

Fixed batch sizes

First it is assumed, that each batch size is fixed to its maximal capacity. In this case, the highest expected profit is 2474.58 cu by producing $1-0-3-0-1-0$ batches from the six generated recipes.

Flexible batch sizes

If the batch sizes are allowed to be changed a-priori, the highest expected profit slightly increases to 2475.31 cu with the configuration of $1-0-3-0-0-1$ and the batches of the third recipe (C) are scaled down to 91 percent in average.

The optimal schedule is shown in Figure 6.2. [7]

[6] "cu" stands for cost unit

[7] The letters A,C, and F correspond to the different recipes, while tasks 1,2,3,4,5 are Reaction 1, Reaction 2, Reaction 3, Separation, and Heating respectively.

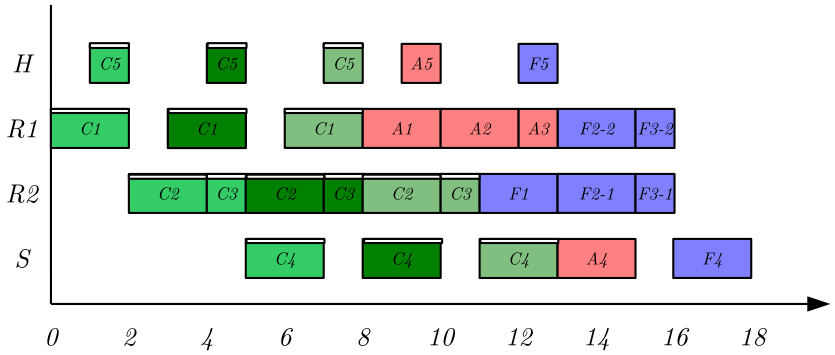


Figure 6.2: Optimal preventive solution for the example

Two stage case

If the sizes of the batches can be decided after the uncertainty realizes, the highest expected profit becomes 2689.87 cu. This can be achieved by selecting the same previous configuration, $1-0-3-0-0-1$.

Remarks

In a realistic industrial situation, this type of optimization problems can occur rather frequently, even on a daily basis, which requires high efficiency. It has to be noted, however, that some of the parameters do change frequently, others very seldomly. In the S-graph based approach these parts are separated, and addressed with different techniques as illustrated in Table 6.2.

Frequently changing part	Permanent part
Market related parameters (prices, demands, scenario parameters, ...)	Technology related parameters (set of products and units, processing times, ...)
Expected profit evaluation	Scheduling feasibility test
computationally easy LP problems and minimum search	computationally difficult S-graph based B&B procedure

Table 6.2: Polarity of the S-graph based approach

This polarity gives rise to the following beneficial application:

1. As an initialization, for all of the configurations the optimal makespan is identified via the S-graph framework within a given time horizon. Based on this horizon, this may take up a considerable amount of time, however, it will never be repeated, until either the set of the products, or the technology changes, and that is assumed to happen seldomly.

2. At each occurrence, when a stochastic optimization problem appears, the following steps need to be executed:

   (a) The time horizon must be identified, if it is not part of the problem definition.

   (b) The configurations with smaller makespan than the time horizon are selected from the database.

   (c) Each of these configurations is evaluated for expected profit using the most recent market data. This will take at most minutes.

   (d) The configuration with the highest expected profit is selected, and the corresponding schedule is loaded from the database.

Using this procedure, the difficult part of the optimization problem must be carried out only very seldomly, while the frequently changing market related parts can be addressed each day. Also if the optimization has already been carried out, but a more recent forecast arrives, the algorithms can be quickly executed again to have a more accurate estimation, thus a better solution.

The tests above had been carried out with a similar procedure as well. The 6 dimensional search space was tested for feasibility. After the initialization, the search space was reduced to 6400 configurations, among which 318 was tested for feasibility, and 187 were found to be feasible.[8] In each of the three cases, these 187 configurations were evaluated and a minimum was found. Even for the most complicated two-stage LP model, the solution of that 187 LPs took less then 1 minute to solve. This time would obviously increase, if the number of scenarios grow. However, as a comparison, the deterministic throughput maximization of this example for just 17 hours could not be done within 1 hour.

The additional advantage of this procedure is, that the decision maker can see an ordered list of all of the feasible schedules, and easily select from them based on other aspects that may have not been included in the model.

Summary and concluding remarks

In this chapter several S-graph based approaches has been presented for maximizing the expected profit in uncertain environments. The algorithms are based on the throughput or revenue maximization algorithm published in Chapter 4. The algorithms may consider

- fixed or variable batch sizes
- purely preventive or two-stage problems
- discrete or continuous distribution of the uncertain parameters

The advantages and capabilities of the approaches were discussed in detail, and illustrated via a case-study.

[8] The set of feasible configurations and the corresponding expected profits are given in Section C.3.

Related publication

- Lainez, J. M., M. Hegyhati, F. Friedler, L. Puigjaner, Using S-graph to address uncertainty in batch plants, Clean Technologies and Environmental Policy, 12, 105-115 (2010). [IF = 1.120]

Related conference posters

- Hegyhati, M., J. M. Lainez, L. Puigjaner, F. Friedler, Preemptive optimization of chemical batch processes with continuous external uncertainties based on a novel S-graph branching strategy presented at: PRES 2010, Prague, Czech Republic, August 29 - September 1, 2010.
- Laínez, J., M. Hegyhati, L. Pugjaner, F. Friedler, Using S-graph to address uncertainty in batch plants, presented at: PRES 2008, Prague, Czech Republic, August 24-28, 2008

Chapter 7

Generalized S-graph model: the Event S-graph

The previous chapters have shown that the S-graph framework provides efficient modeling and optimization tools for the scheduling of batch chemical processes. Since its original introduction, the framework has been extended to many problem parameters. Due to the limited capabilities of the original model, some problem parameters are difficult to be considered, or it can be done only in a workaround fashion. Section 7.1 shows some examples.

The existing extensions vary in the way they modified the framework in order to extend it to several problem parameters. Some of them modified only the algorithms, others altered the underlying mathematical model too. Having these inhomogeneous, separate extensions can, however, cause some difficulties:

Compatibility issues If the trunk framework is developed to several different directions, the merge between the different branches can be troublesome in some cases. This is especially true when not only the algorithms are modified, but it has also been necessary to change the underlying model.

Implementation and developement software tools If the model is changing, the implementation needs to be changed accordingly in order to be efficient. Development of a graphical modeling tool becomes also difficult, as it should be flexible towards unknown future changes.

To overcome the limits on the modeling capabilities of the S-graph framework, and to provide a more uniform and generic platform for future developments, the re-thinking and extension of the S-graph framework is needed. The extension requires a careful review on the current structure of the S-graph framework, including its modeling tools, models, and optimization algorithms. This and the possible ways of future extensions is discussed in Section 7.2.

Based on the experiences from these investigations, the developed, new framework is based on three cornerstones:

Extended model The most significant enhancement. The major aspect for the new model was to be general enough to address a wide range of scheduling problems; it should not be restricted to chemical batch processes, and it should make the modeling process more transparent. This model, the so-called Event S-graph or *e*S-graph is introduced in Section 7.3.

Improved modeling techniques Due to the very simple and restricted original S-graph model, the modeling step was obvious and straightforward. Due to the extended model, however, the *e*S-graph framework has a bigger emphasis on the modeling step as well. In Section 7.4, it is illustrated, how several common problem features can accurately be modeled using the *e*S-graph.

General purpose scheduler Keeping the original idea of the equipment based makespan minimizer, a general purpose algorithm is developed and introduced in Section 7.5 that performs makespan minimization on any kind of problems that can be purely formulated with the *e*S-graph, without any additional information.

7.1 Modeling difficulties with the original framework

In this section it is illustrated that some features of batch (or semi-batch) processes occurring in most of the scheduling problems require various extensions of the original approach, which in some cases can be circuitous.

7.1.1 Transfer times

The original S-graph framework does not consider the transfer of the intermediate materials. The only transfer that is represented in the model is the removal of the final product from the last processing unit. For approaches that cannot properly address material transfers, there are two common practices to tackle the problem:

- neglecting the transfer times if they are very small compared to processing times
- lumping the transfer times to the processing time of the task producing the intermediate

The key problem with both of these methods is that in case of continuous material transfer both units (the one that produced the intermediate material, and the other that receives it) must be free for the time of the transfer. Disregarding this can result in undesired solutions: see Section 3.2.

These two methods are obviously easily applicable in the S-graph framework as well. It is possible, however, to model the material transfers correctly without any simplification.

Originally, if a task i' is scheduled to be performed after i in unit j, and $i2$ is the subsequent task of i, then a zero weighted scheduling arc is inserted from $i2$ to i' considering NIS policy as it is shown with the corresponding Gantt chart in Figure 7.1.

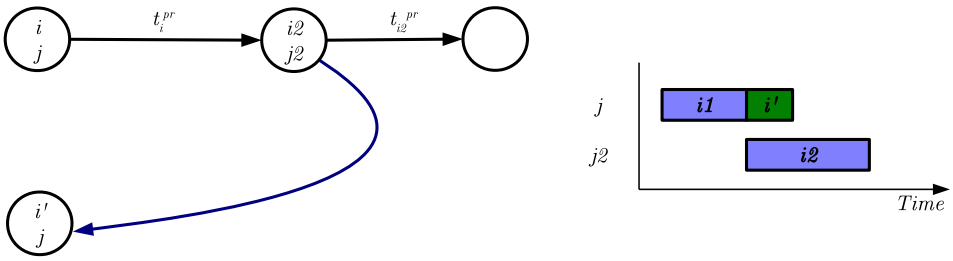


Figure 7.1: Schedule arc in the original framework and the corresponding Gantt chart

The Gantt chart however should look like as it is illustrated in Figure 7.2 if there is a transfer time (no matter how tiny it is).

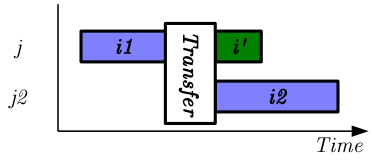


Figure 7.2: Accurate Gantt chart representing the transfer

Having a similar, equivalent Gantt chart can be achieved by increasing the weight of the schedule arc by the transfer time (t^{tr}). However, this is not enough, as the unit j would still be able to start performing i' before the transfer finishes. A simple workaround to tackle this issue is to increase the processing time of i' by the transfer time as well, as shown in Figure 7.3.

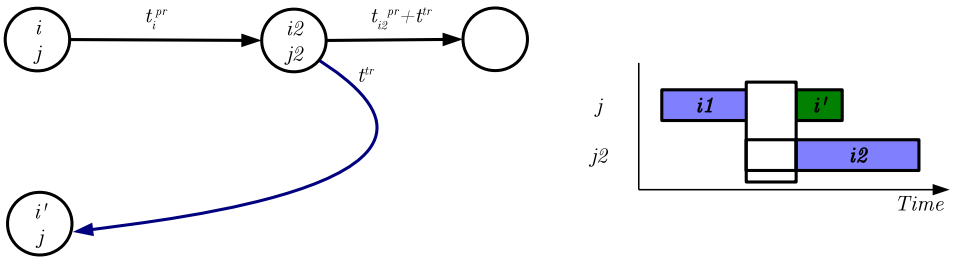


Figure 7.3: Modified S-graph to address transfer time

For the case when either i or $i2$ has several subsequent task, the approach is similar as illustrated in Figure 7.4.

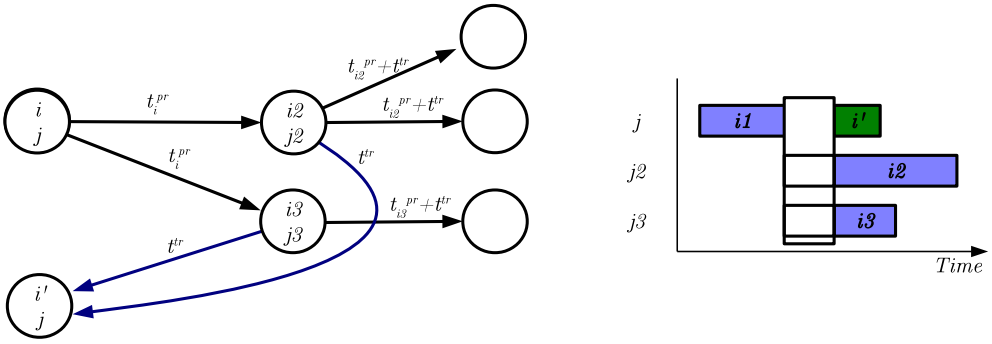


Figure 7.4: Addressing transfer time with several subsequent tasks.

Although the solutions provided by this method meet the requirements above, it is more a modeling trick than an appropriate solution. Moreover, some questions may arise, such as how to tackle the problem when the transfers for $i2$ and $i3$ take different times and may happen independently, etc.

Thus, an other option is also available, which requires additional vertices inserted into the graph which represent material transfers, as shown in Figure 7.5.

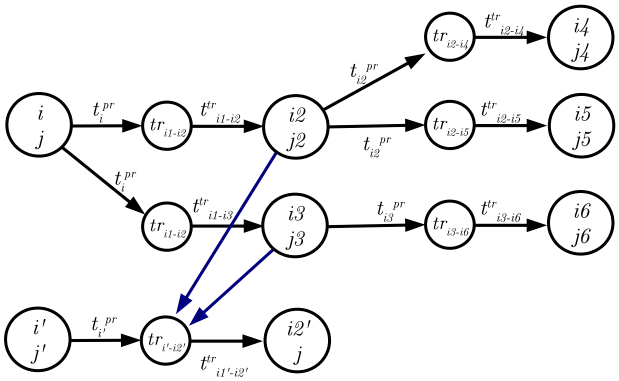


Figure 7.5: Addressing transfer time with additional nodes.

The schedule arcs still start from the task nodes. However, they yield to the inlet transfer of the next scheduled task. As it is shown, the arcs representing the processing time of $i2$ and $i3$ does not change their weight, which gives a more straight-forward modeling of the problem.

7.1.2 Waiting before production

Considering NIS policy, the intermediates are allowed to wait in the units that produced them until they can be transfered to the upcoming task in the recipe. However, this description does not specify whether the intermediate can wait in the unit performing the upcoming task before its execution starts. If inputs of each task are either raw materials, or produced by another single task, this question is irrelevant with UW policy (which is considered for

now), as the processing and the storage phases can be shifted arbitrary. The situation is different, however, if two or more tasks provide the inputs for a task, as illustrated on the example in Figure 7.6.

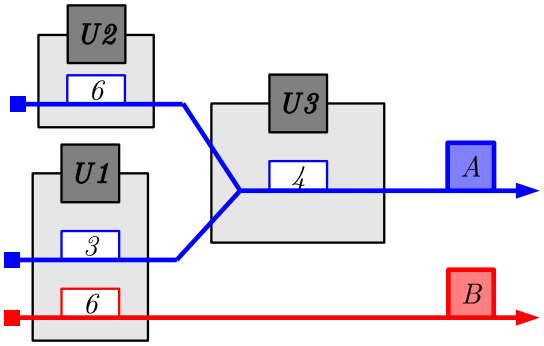


Figure 7.6: Example for product with task of multiple inputs

In this problem, there are two products and the third step of producing A requires two different intermediates from both of the first two steps. After modeling the problem with the S-graph framework, and applying the B&B algorithm, two different solutions can be generated, as shown in Figure 7.7 with both the schedule graphs and the corresponding Gantt-charts. The solutions have the makespan of 12 and 13 time units, respectively.

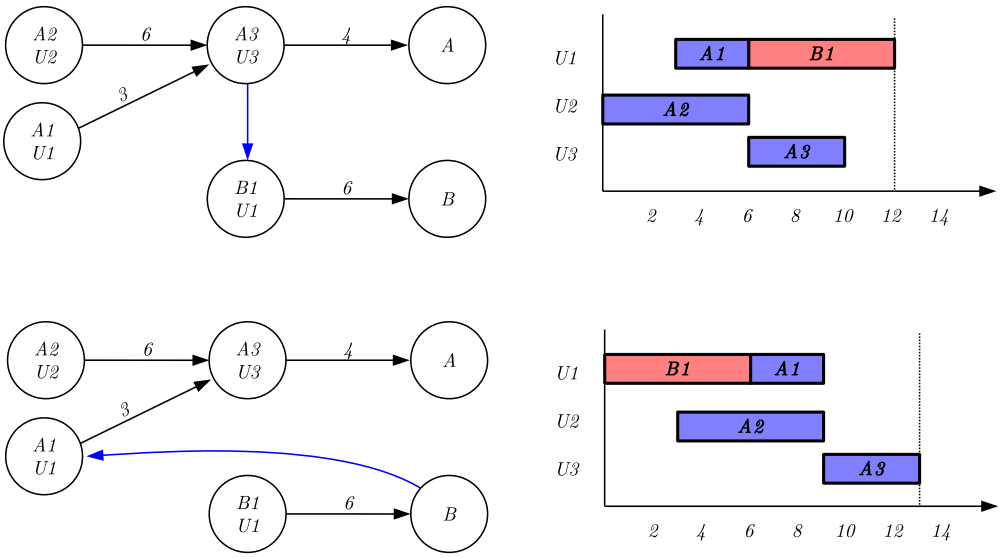


Figure 7.7: Solutions provided by the S-graph algorithm for the example in Figure 7.6

However, the question arise, whether it is possible to store the output of $A1$ in $U3$ and wait, until $U2$ can also load the output of $A2$ into $U3$. If that is possible, the makespan of 10 hours could be achieved, as illustrated in Figure 7.8.

In many applications, the safety or other regulations of the facility forbid the storage of the intermediate in a unit before processing, thus the solution shown in Figure 7.8 is practically infeasible. However, if it is not forbidden, the original S-graph approach needs

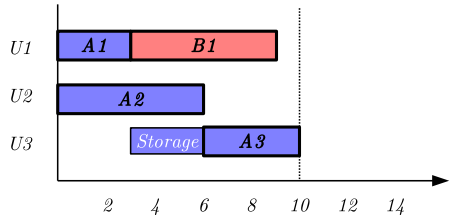


Figure 7.8: Solution with better makespan if storage if allowed before processing

to be extended in order to address this situation properly. This can be done similarly as in the previous section, by introducing some additional nodes. Holczinger [52] has investigated a similar issue, and provided an extended S-graph model in the case when the order of the input materials and the required delay between them is fixed by the recipe.

7.1.3 Continuous processes and multiple resources

Although the scope of this work is mainly on batch processes, in many cases batch and continuous units work simultaneously. If there is a storage before and after the continuous unit, they can be viewed and modeled as a single batch unit. Otherwise, the batch unit that produced the input of a continuous process must be available throughout the whole operation of the continuous unit. This behavior shows a lot of resemblances to the material transfers that have been discussed formerly. As a matter of fact, the transfer itself is a continuous task carried out by a compressor for example. The problem becomes even more complex if there are several input or output task of a continuous task, with some of them being continuous as well. The original S-graph framework considers only batch processes. However, as it has been pointed out, addressing them is rather similar to addressing the transfers, which needs to be done for batch processes as well.

In many industrial applications, a task needs several resources to be executed, e.g., in a furniture factory, a unit can usually work with different "heads" that are shared between the units[53]. Alternatively, operators, electricity, coolers, etc. can and must be included in the scheduling of a single process. In the original S-graph framework only one unit can be assigned to the a task, and that unit is assumed to be the only unit that performs that task.

7.2 Scheme of optimization with the S-graph framework and its extensions

In this section the general optimization procedure of the S-graph framework is analyzed. The basic concept is shown in Figure 7.9 [1] .

[1]Note that the selection of the considered variables, problem parameters, and real execution of the generated schedule is not included in the figure, as it is outside of the scope of the current interest.

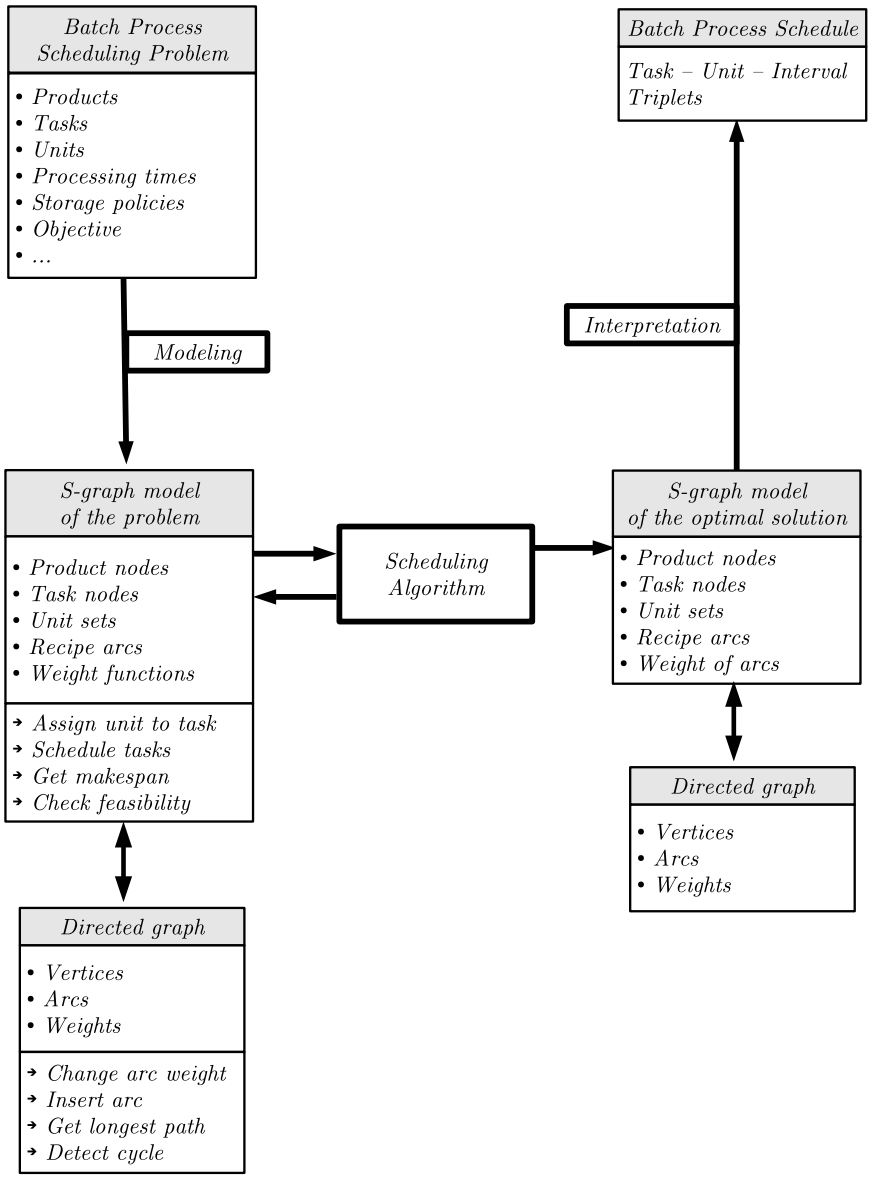


Figure 7.9: General scheme of the S-graph based optimization procedure

The problem itself consists of all the problem data, as discussed in Section 1.2. The first step is the construction of the mathematical model of the problem, namely an S-graph. There is not much discussion about this step in the literature, as this step is rather straightforward for those problems that the framework can address. To each product and task a node is assigned, production dependencies are represented by arcs, etc., as described in Section 2.3. Unlike in the case of MILP approaches, there are no different ways the model should be formulated for the problem; it is a clear and unambiguous step. This highlights a great advantage of the S-graph framework compared to MILP approaches: the mathematical model that is used by the algorithms has a straightforward relation to the problem, and the modeling does not require any special skills.

It has to be mentioned, however, that the S-graph model has a kind of embedded model behind the scenes. This is the directed graph itself, not including any scheduling problem specific information like the plausible unit sets, etc. At the moment, differentiating between this subset of the model may seem unnecessary, but its importance will be clarified later. As for now, the only important thing is that the scheduling algorithm operates strictly only on the S-graph model via operations like assigning a unit to a task node. This action then is interpreted in the graph model as well by changing the weights of recipe arcs and inserting schedule arcs. [2]

Finally, the Scheduling Algorithm generates the S-graph corresponding to the optimal schedule, which also contains a final form of the graph itself. Note that the Scheduling Algorithm here can refer to many approaches that have been published so far, not only for the equipment based makespan minimizer. Last but not least, this solution is interpreted, and a Gantt chart is generated.

Extensions

There are several ways the extensions affect the structure introduced above. Some of them are briefly introduced here. The structural differences between the different types of extensions render it obvious how difficult it can be to keep these extensions compatible with one another.

Model conversion In case of the S-graph based Wet-etch optimization[71, 97], or the scheduling of tunnel boring machines[4], the structure of the optimization process remains intact, but the original problem is transformed to a mathematically equivalent chemical batch process scheduling problem. The whole process is shown in Figure 7.10. Converting LW problems to ZW equivalents in Subsection 5.3 was based on the same idea.

The obvious benefit of this approach is that there is no need to change the S-graph algorithms, or the solver implementation. Thus the time needed for the developement and

[2] In Section 2.3 the presented algorithm has direct access to the (N, A_1, A_2) structure, however: a) this structure does not really represent all the information stored by a scheduling problem, and b) the state-of-the-art implementations do not follow that paradigm, but a layered one, as mentioned.

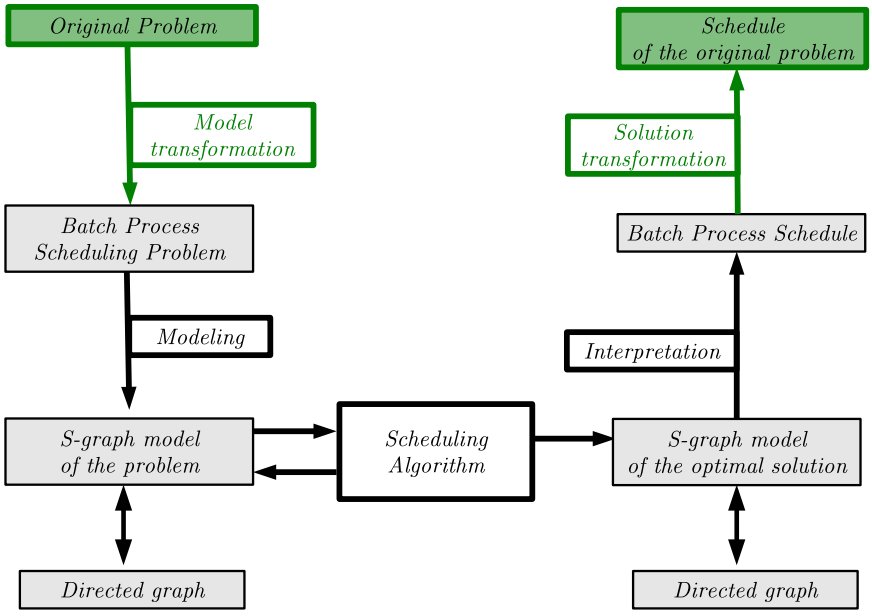


Figure 7.10: Optimization procedure of the S-graph framework with model transformation

to get results is considerably low. On the other hand, however, the approach no longer holds the aforementioned advantage, i.e., the model is no more a straightforward representation of the problem. Moreover, this approach usually does not exploit any problem specific feature, which can have a serious effect on the efficiency, as discussed by Osz[97] for example. Also, if the problem description of the original approach is changed for some reason, it may render the transformation between the models impossible.

This type of extensions make the structure of the S-graph based optimization procedure similar to that of the MILP approaches, where the MILP model is an intermediate model that is solved by a general purpose solver, and the modeling of the problem is in itself an important step.

Algorithmic extension In some cases, the framework is extended to problem classes featuring some additional parameters that are entirely handled by the algorithmic part of the approach. The S-graph model and its interaction with the internal graph model is unaltered. This type of extension can be observed, e.g., at the LP based LW scheduling (Section 5.1), extension to throughput or expected profit maximization (Chapters 4 and 6), or heat integration [2]. The scheme is shown in Figure 7.11

In a way, this type of extension is the simplest, and clearest. However, its merge with other extensions can be rather difficult. It also has to be noted that the additional data of the problem needs to be parsed and passed to the algorithm. In the earlier implementations of the S-graph framework this required a new input format, a new parser, and a new data structure. Since these additional data are not required by the S-graph model itself, and usually seldomly accessed by the algorithm as well, Kovacs [71] developed a framework in the S-graph implementation that allows arbitrary extensions of the input file, and the parser

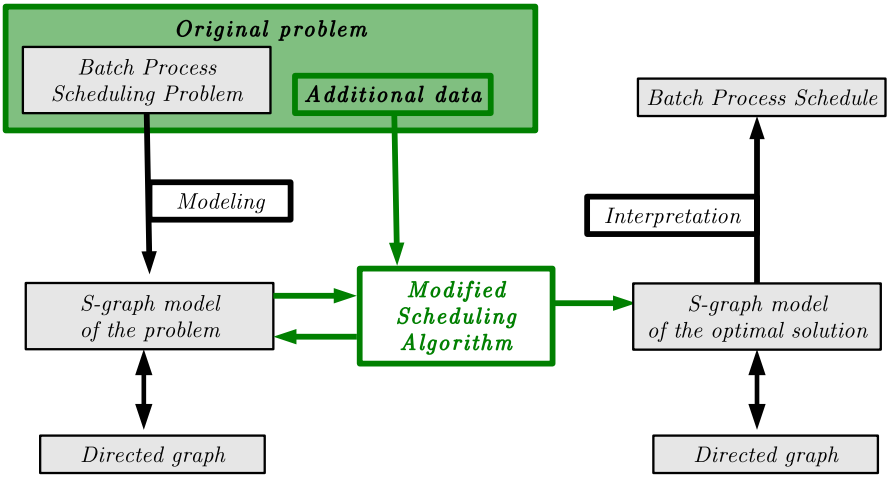


Figure 7.11: Optimization procedure of the S-graph framework with algorithmic extension

automatically passes on the data in a generic structure to the algorithm responsible for the optimization without the need of any modification for the parser or main data structure.

Model based extension There are several problem features that change the scheduling problem in its core, thus it is reasonable to implement these changes on the model level. A typical example can be the negative arc based LW algorithms (see Section 5.2), and some acceleration, e.g., so-called auxiliary batching arcs[51], predictive schedule arc insertion, or the enhanced wet-etch approaches[97]. The scheme is shown in Figure 7.12

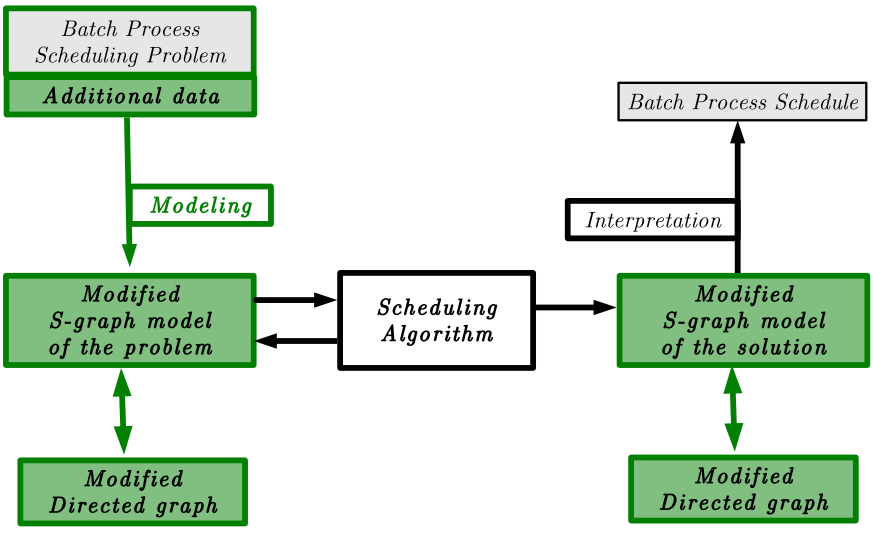


Figure 7.12: Optimization procedure of the S-graph framework with model based extension

Note that the problem may not have any additional information, like in the case of the previously mentioned accelerations. In those cases even the interface of the model remained the same, just the model handled the internal directed graph model differently. This is completely concealed by the S-graph model from the algorithms, or the modeler,

or the solution interpreter. In other cases, like the LW example, the problem itself has additional information (maximal waiting times) that needs to be included in the modeling step. However, the interface for the algorithm remains the same, i.e., the equipment based algorithm for example should not be altered in any way.

An obvious advantage of the approach is that these extensions are usually automatically compatible with the other algorithmic or model based extensions. On the other hand, combining several model based extensions may be challenging, not to mention that the model can get unnecessarily complicated for even the simpler problem classes.

Other There are of course other ways of extensions as well, that are usually the combination of the above three. The combinatorial recursive LW approach (Section 5.3) that inserts additional arcs into the model is a fine example for a rather lumpish extension. The algorithm is changed and it directly modifies the internal graph model by bypassing the interface of the S-graph model.

7.3 Mathematical description of the *e*S-graph model

The former two sections showed some of the limits of the S-graph framework and the complications that the different type of extensions can cause. In this section a generalized S-graph model is introduced that is the result of the thorough consideration of the formerly mentioned issues.

There were a couple of major aspects kept during the development of the new model:

- The model should be general enough to address a wide range of scheduling problems, and it should not be restricted to chemical batch processes. This reduces the need for model conversions and model extensions for future developments.
- The range of parameters that the model can address should be restricted to timing and scheduling related elements. The model should not include other common parameters, like heat integration data, as the model would most probably lose its efficiency. These parameters should be addressed via external tools like LP models in the future too.
- As a secondary objective, the model should be kept as simple as possible, and similar to the original problem, so that the mathematical model (and its straight forward graphical representation) should be understandable.
- The complexity and detailedness of the model should also depend on the problem at hand, i.e., if the model is capable of addressing, e.g., transportation times with some additional nodes, then problems without transportation should not have a more complicated model.
- As far as possible, the model should conceal its inner operations from the algorithms, and provide a general interface with a basic scheduling decision interface.

From the practical point of view, the model should be able to address the following problem features with ease:

- Transfer, cleaning, changeover times
- Continuous tasks
- Tasks requiring several units simultaneously

In the following subsections first the original S-graph model is analyzed to provide the basis for the basic idea of the model in the second section. Then the model is formally defined in the last Section.

7.3.1 Analysis of the original S-graph model

It has already been mentioned, that the graphical representation of the S-graphs is the mathematical model itself. As a representation, the nodes of an S-graph represent tasks and products. However, taking a closer look at the mathematical approaches reveals that the nodes represent the starting of the execution of a task, or the removal of a product. These are only some of the events that occur, several others are not included in the model:

- ending of processing a task (and starting the storage in the unit in case of NIS policy)
- starting end ending the transfer of an intermediate
- loading the raw material into the first unit
- etc.

The original S-graph framework disregarded these events, as everything under the consideration of the original problem set could appropriately be modeled with only the starting of tasks and removal of the products.

Basically, the following entities have a 1-to-1 relation between them in the framework:

- task, that should be assigned to equipments
- beginning of tasks, that correspond to exact timings
- nodes of the S-graph, from where schedule arcs can start

The general idea for the the extended model is to separate the first from the latter two, as described in the next subsection.

Moreover, the original formal description, the S-graph model was a (N, A_1, A_2, w) quadruplet[3], describing only part of the scheduling problem and part of the scheduling decisions, which is basically the internal graph model mentioned in the previous section. Additional

[3] N: the set of nodes; A_1: the set of recipe arcs; A_2: the set of schedule arcs; w: the weight function on the arcs. See page 28 for more detail.

information such as the set of unscheduled tasks, plausible task-unit pairs, processing times, etc. is given separately for the algorithm.

The aim in the extended model is to include all these information for modularity and clear description.

7.3.2 General concept of the *e*S-graph model

As already mentioned in the previous subsection, the general concept behind the new model is the separation of the tasks that need to be scheduled, and the nodes of the graph in the model that represent events. The basic principle of the generalization is this two-layered structure where an entity to be scheduled (task, transfer, cleaning, etc.) can span over several events of the system, and thus, several nodes of the graph. These entities in the model are called *subprocesses*, and there is a many-to-many relation between the events/nodes and them, unlike the one-to-one relation between the tasks and nodes/events in the original framework.

The subprocesses may require resources or a set of resources to be carried out. Subprocesses are assumed to be *non-interruptible*, like the tasks in the original S-graph framework. Thus, if a resource (unit, operator, pipeline, etc.) is assigned to a subprocess, the resource is considered busy in the time interval spanned by the events belonging to the subprocess. Because of non-interruptibility, if a resource is assigned to several subprocesses (either alone or as a member of a resource-set), the intervals of them should be distinct. This also means that a clear sequencing should be made between the subprocesses assigned to the same resource, similarly to the sequencing of tasks assigned to the same unit in the original framework. The sequencing will be implemented as a zero-weighted schedule arc[4] from the node of "latest" event of the former subprocess to the node of the "earliest" event of the latter subprocess. Note that the dependencies between the events belonging to the same subprocess is not necessary linear; a good example for this is a task with multiple inputs and/or outputs, that arrive asynchronously. In this case[5], it is not evident which the latest or earliest event of the subprocess is, and it may also depend on the schedule of the other subprocesses. Thus, when a sequencing decision is made, the model inserts zero-weighted schedule arcs from all of the nodes of the former subprocess to all of the nodes of the latter one. Obviously, many of this might be implied by many others, but in this way, the soundness is ensured, and the cost in terms of memory or CPU is not significant.[6]

An important modification is that unlike in the original framework, several resources can be utilized simultaneously in the new framework. The model allows two ways of implementing such problem features:

- A set of resource can be assigned to a subprocess.

[4] More precisely, a $[0, \infty]$ weighted arc, as discussed later.

[5] Strictly speaking, linear ordering is not needed, only a unique minimal or maximal element.

[6] A more clever approach could be to find the set of minimal and maximal nodes for each subprocess and use them. Both could be correct implementation of the same concept.

- The events of two subprocesses overlap, and in that interval the assigned resources to both of the subprocesses are busy.

This provides a lot of expressive power to the new model that can be exploited, as illustrated in some examples in Section 7.4.

The limited and zero wait storage policies showed an example, why "not later than" type of constraints are necessary to be included in the model to be able to address a wide range of problems. Thus, in the new model, the *weight of each arc is a pair of non-negative numbers* $[min, max]$, which gives bounds on the timing of the events corresponding the starting node s and destination node d: $t_s + min \leq t_d \leq t_s + max$. Note that max is always greater or equal to min, and it is allowed to take the value of infinity as well, when there is no upper bound on the timing difference. Also, allowing min (or max) to take negative values would not result in additional expressive power, as an arc from s to d with ,e.g., $min = -3$ is equivalent to an arc from d to s with $max = 3$. Note that the model may end up having multiple arcs between the same two nodes. They can of course be summed up as a single arc with a maximal min, and minimal max value. In this case, if the max value becomes lower than the min value, it means infeasibility.

7.3.3 Formal definitions

In this subsection, the formal definitions of the framework are given along with implementational comments. First, the scope of the interest is given by an exact mathematical definition of the problem. Then, after the formalization of a schedule, the definition of an eS-graph and its inner model is given.

Scheduling Problem

An extended scheduling problem can be given by a 6-tuple, $(E, \mathcal{SP}, D, J, \mathscr{O}, \mathscr{W})$ where

E is the set of events

$\mathcal{SP} \subseteq \wp(E)$ is a set of subprocesses[7]

D is the set of dependencies between events

J is the set of units/resources to be scheduled

$\mathscr{O} \in \mathcal{SP} \to \wp(\wp(J))$ is a set of assignment options

$\mathscr{W}$ is a weight function for dependencies based on scheduling decisions.

Note that an objective function is not considered as part of the definition, as the goal of a scheduling problem is not necessarily the search for an optimal solution. The framework

[7] $\wp(E)$ stands for the power set of E.

may be used for checking the feasibility of the problem, or generating all of the feasible schedules.

E is the set of events, that is considered in the process, and $\mathcal{SP}$ is a set of subprocesses that need to be assigned to units and to be scheduled. Each subprocess, $S \in \mathcal{SP}$ is a subset of E. In the above definition, it is not allowed for two subprocesses to have the same set of events. In implementation, and modeling, however, it can be useful to have several subprocesses with the event set. This relaxation will affect the other parts of the formal models or algorithms; thus it is highly advised to implement the model in that way. Here this form is kept for simpler formalization. Note also that not all of the events are necessarily included at least in one of the subprocesses.

D is the set of dependencies between the events. To ease formalization, for each $d \in D$, $e^-(d)$ will denote the event on which $e^+(d)$ depends. From the modeling and implementational point of view, this set is given as $D^R \cup \bigcup_{S \in \mathcal{SP}} D^S$, where D^S is the set of dependencies induced by the subprocesses, and D^R are other dependencies of the recipe independent from the subprocesses. The concept behind this partitioning is a modular view, where a subprocess is a standalone entity, and a process can be built up from these building stones, and making connections between them.[8] Note, that D may contains several parallel dependencies with the same events. From the practical point of view, only the strongest is binding. However, this flexibility is again, advantageous from modeling and implementational purposes.

As usual, J is the set of resources of the system that are needed for the subprocesses to be carried out. These can be units, pipes, operators, electricity, etc.

$\mathscr{O}$ is the function of possible assignments: for each $S \in \mathcal{SP}$, $\mathscr{O}(S)$ is a set of resource sets, that can carry out the subprocess, i.e., each element of $\mathscr{O}(S)$ is a subset of J, and one task in the scheduling is to select one of these sets.

$\mathscr{W}$ is the function that assigns weights to each dependency in the system. These weights often depend on the scheduling decisions (assignments and sequencing), thus, the formal definition of this function depends on concepts discussed later. In many problem classes, however, this function is more simple, and has a modular structure too. As an example, in the case of the previously considered problems, the weight of a dependency depended only on the assignment that has been made at a subprocess. As a result, a wide range of scheduling problems could be covered if $\mathscr{W}$ were defined as a $D \times S \times \wp(J) \to \mathbb{R}^* \times \mathbb{R}^*$, where $\mathbb{R}^*$ is the set of non-negative real numbers and ∞. This definition would, however, not cover the empty robot movement times of wet-etch processes for example; thus a more general definition will be given later. Note that an other weakness of this simpler definition would be that it does not allow to add weights to the dependencies that will be the result of sequencing decisions. This could be avoided by replacing D with $E \times E$. Following this

[8] This philosophy is followed in a further extent in the current XML description, where the set E is defined by events sets of subprocesses, additional events and alias rules, that merge the same events defined by several subprocesses. As a simple example, the arrival of an input of one subprocess may be the same as the removal of the output of an other, which depends on the actual process.

philosophy, $\mathscr{W}$ could be defined for the whole $E \times E$ domain, with the value of $[-\infty, \infty]$ where no dependency is given. However, in this case, it is more difficult to formalize parallel dependencies.

The schedule

In general, there are two types of decisions that has to be made during scheduling:

Assignement Each subprocess has to be assigned to a plausible resource set.

Sequencing If the resource sets assigned to two different subprocesses have a non-empty intersection, the order in which they are carried out by the units in the intersection must be given.

In order to allow the development of general scheduling algorithms, it is not assumed that a resource set is assigned to a subprocess always at once. On the contrary, it is allowed for a subprocess to have a single resource assigned to it, and the remaining elements of the adequate resource set are assigned later. Also, it is not assumed that if two subprocesses are assigned to the same units, their sequencing must be included in the schedule immediately. The proposed formulation poses no such constraints and allows these decisions to be included separately at different stages of the optimization.

Thus the formal description of a schedule for a scheduling problem $(E, \mathcal{SP}, D, J, \mathscr{O}, \mathscr{W})$ is $(\mathscr{A}, \mathscr{S})$, where:

$\mathscr{A} \subseteq J \times \mathcal{SP}$ is a set of resource-subprocess assignments made so far.

$\mathscr{S} \subseteq \mathcal{SP} \times J \times \mathcal{SP}$ is a set of sequencing decisions made so far.

To ease further descriptions, the following notations will be applied:

$\mathscr{A}(j) = \{S \mid (j, S) \in \mathscr{A}\} \quad \forall j \in J$

$J_{\mathscr{A}}(S) = \{j \mid (j, S) \in \mathscr{A}\} \quad \forall S \in \mathcal{SP}$

$S_1 \xrightarrow{j} S_2$ is true if $(S_1, j, S_2) \in \mathscr{S}$, false otherwise

$\overline{\mathscr{S}}$ is the transitive closure of $\mathscr{S}$, i.e., if $S_1 \xrightarrow{j} S_2$ and $S_2 \xrightarrow{j} S_3$ then $S_1 \xrightarrow{j} S_3$.

The empty schedule is obviously the $(\emptyset, \emptyset)$ pair. A schedule $(\mathscr{A}, \mathscr{S})$ is said to be complete for a scheduling problem $(E, \mathcal{SP}, D, J, \mathscr{O}, \mathscr{W})$

- for all $S \in \mathcal{SP}$, $J_{\mathscr{A}}(S) \in \mathscr{O}(S)$, and
- for all $S_1, S_2 \in \mathcal{SP}$ such that $S_1 \neq S_2$: for all $j \in \mathscr{A}(S_1) \cap \mathscr{A}(S_2)$: $(S_1 \xrightarrow{j} S_2) \oplus (S_2 \xrightarrow{j} S_1)$[9]

[9] $\oplus$ stands for the exclusive or logical operator

The first rule states, that the set of resources assigned to a subprocess must be a plausible set from the problem description. The second condition states that if a unit is assigned to two different subprocesses, exactly one of them should be scheduled earlier than the other.

Note, that a complete schedule is not necessary feasible. A very simple counterexample could be, when there are two units that are simultaneously assigned to both of two subprocesses, i.e., $j_1, j_2 \in \mathscr{A}(S_1) \cap \mathscr{A}(S_2)$ and the sequencing decisions are inconsistent, i.e., $(S_1 \xrightarrow{j_1} S_2)$ and $(S_2 \xrightarrow{j_2} S_1)$.

For simplified formulation let $D^{\mathscr{S}}$ denote the dependencies that are the results of sequencing decisions, i.e., $D^{\mathscr{S}} = \bigcup_{S_1 \xrightarrow{j} S_2} (S_1 \times S_2 \times \{j\})$. Note that the third element in the triplets is needed to separately identify the dependencies that are caused by different units.[10]

Now the weight function of the scheduling problem can be defined in a general way: $\mathscr{W} : (D \cup D^{\mathscr{S}}) \times (\mathscr{A}, \mathscr{S}) \to \mathbb{R}^* \times \mathbb{R}^*$.

Moreover, the notations $\mathscr{W}_{min}(d)$ and $\mathscr{W}_{max}(d)$ correspond to the lower and upper bounds of $\mathscr{W}(d, \mathscr{A}, \mathscr{S})$.

The eS-graph model

After the former introduction, the eS-graph model of a partially scheduled problem can simply be given as an 8-tuple: $\mathbb{S} = (E, \mathcal{SP}, D, J, \mathscr{O}, \mathscr{W}, \mathscr{A}, \mathscr{S})$, which consists of all of the problem parameters and the scheduling decisions made so far.

The inner model is a directed graph with weighted arcs: $G(\mathbb{S}) = (V, A, w)$, such that:

$$V = E$$

$$A = \{(e^-(d), e^+(d)), (e^+(d), e^-(d)) \mid d \in D \cup D^{\mathscr{S}}\}$$

$$w(v_1, v_2) = \max\left(\max_{\substack{d \in D \cup D^{\mathscr{S}} \\ e^-(d) = v_1 \wedge e^+(d) = v_2}} \mathscr{W}_{min}(d), \max_{\substack{d \in D \cup D^{\mathscr{S}} \\ e^-(d) = v_2 \wedge e^+(d) = v_1}} -\mathscr{W}_{min}(d)\right)$$

The vertices are simply the events, and the arcs represent the dependencies. To each dependency two arcs are assigned a "forward" arc for the lower bound on a time difference, and a "backward" arc for the upper bound. The weights are assigned accordingly. If there are parallel dependencies, the assigned weight is the maximal among them. The arcs with $-\infty$ weight can be neglected, as they do not pose any real constraints.

Similarly to the original S-graph framework, the longest path in this graph gives the makespan of the schedule in case of a complete schedule. For incomplete schedules, the longest path provides a lower bound on the makespan, if it is assumed that the intervals assigned by $\mathscr{W}$ satisfy inclusion after any extensions on the schedule.

Moreover, a positive cycle means infeasible schedule in a similar way. Whether a zero-weighted cycle poses an infeasibility depends on the application.

[10] For more precision the subprocesses should have been included as well, as it may be possible that there are parallel dependencies between two events because of two different subsets. However, in this case, the units should be different as well.

7.4 Modeling scheduling problems with the *e*S-graph

In this section, modeling techniques with the *e*S-graph are illustrated. The examples provide a guide to how real world scheduling problems should be modeled within the new framework. Most of the description focuses on batch process scheduling; however, the modeling patterns can be used on other fields as well. Formal definitions are omitted; only graphical representations of the *e*S-graph model of the recipe are given, where

- events are represented with nodes (circles)
- the initial dependencies between the events are represented with directed arcs.
- subprocesses are highlighted with colored dashed border blocks, along with the plausible resource sets.

On each arc, the initial interval is given. However, to simplify graphical representation, the notations in Figure 7.13 are used throughout the section.

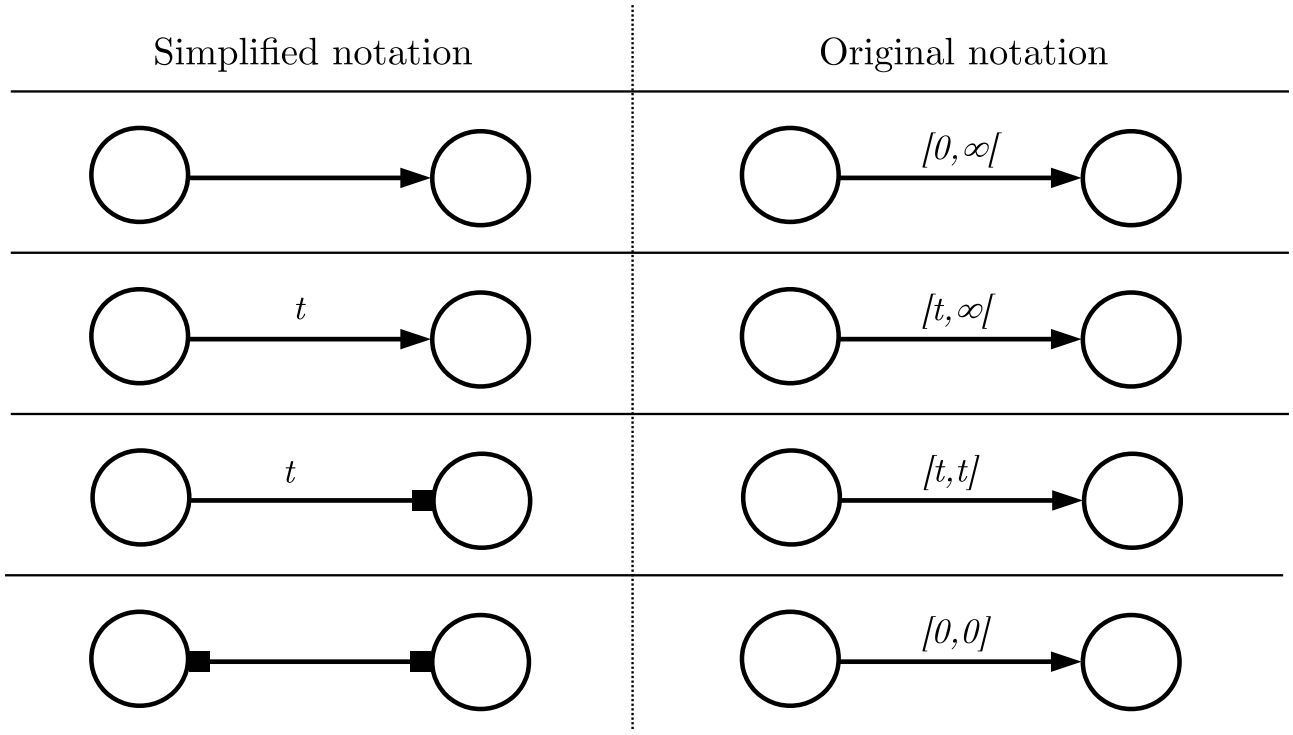


Figure 7.13: Simplified dependency notations for the *e*S-graph

It will usually not discussed in detail, how the $\mathscr{W}$ function should work. In fact, it is rather straight-forward in most of the cases.

Tasks Tasks are one of the basic subprocesses for a scheduling problem. The two basic events that correspond to this subprocess are the starting of the execution of the task and

its ending, as illustrated in Figure 7.14 with the detailed and simplified notation as well. The weight function should assign the $[t_i^{pr}, t_i^{pr}]$ values based on the assigned unit sets that

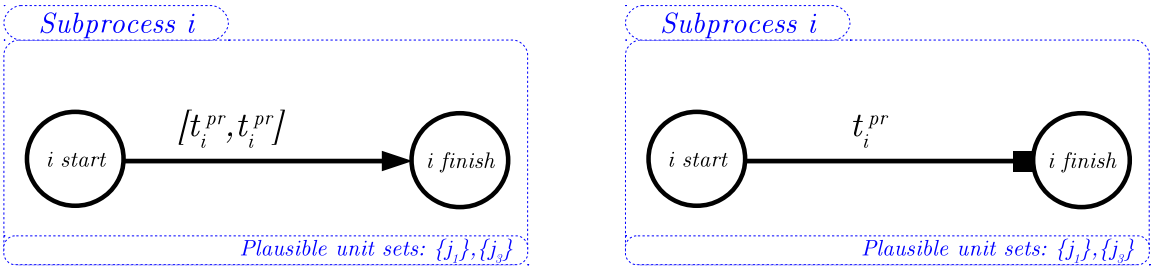


Figure 7.14: eS-graph model of a simple task

can be either j_1 or j_3 alone. The timing difference between the two events are fixed, since the process takes an exact amount of time. Note that if the processing time for the task is different for different units, then the smallest should be the lower bound of the initial interval, and the largest should be the upper bound.

Input, output transfers If there are inputs and outputs to be transfered into and from the unit that is assigned to the task, then the subprocess should also include these events, as illustrated in Figure 7.15. The process has a single input and a single output material. The

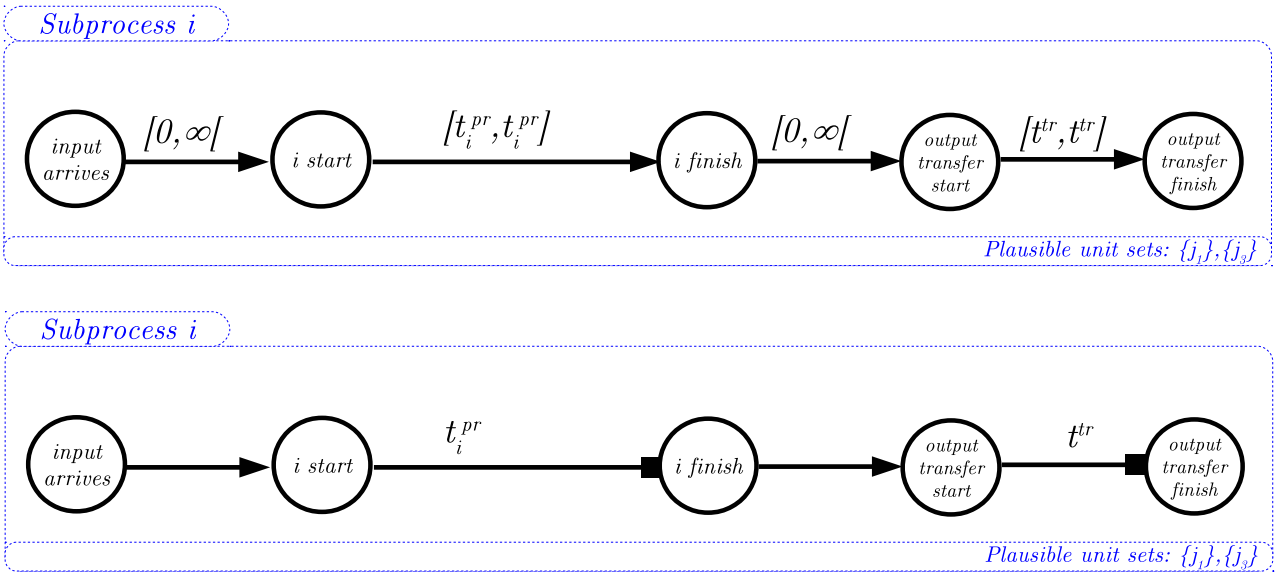


Figure 7.15: eS-graph model of a task with input and output transfers.

transfer of the input is considered to be discrete, i.e., the material arrives in a single event. On the contrary, the output is removed by continuous transfer, thus the unit to be assigned to this subprocess must remain until the transfer finishes. Similarly to the processing step, the transfer takes a certain amount of time. In this example, there is no upper limit on the

first arc, i.e., the input material can be stored in the unit after arrival arbitrarily long. The same holds for the output material as well. If for some reason, the input should not wait more then a t^{max} amount of time due to some physical or chemical properties, it could be expressed by changing the weight of the first arc to $[0, t^{max}]$.

Overlap with transfer subprocesses The transfer events may also be part of other subprocesses, as illustrated in Figure 7.16. The transfer for the intermediate is part of

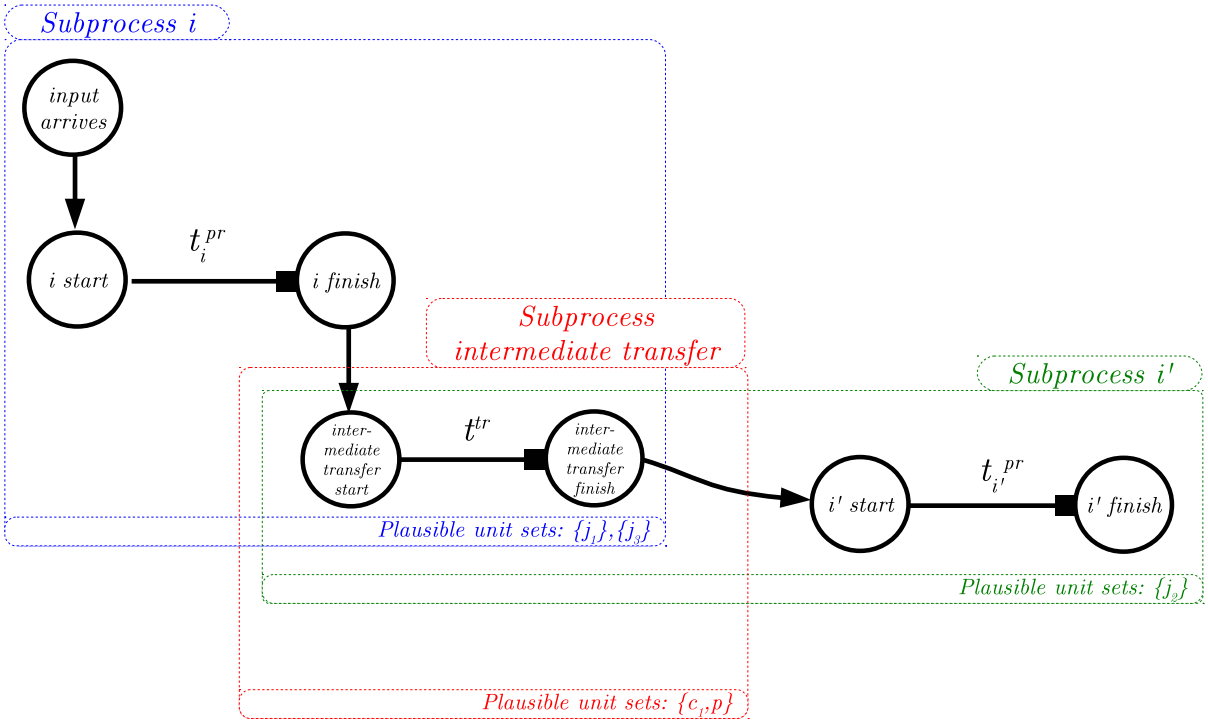


Figure 7.16: *e*Sgraph model of a task and transfers

three subprocesses, as the sending, the receiving unit, and the units performing the transfer must also be occupied with the transfer. Note that there is only a single suitable set of units for the transfer subprocess: $\{c_1, p\}$, which has two elements, as both the first compressor and the pipeline network are needed to carry out the transfer. (And an other transfer may not use them during this time.)

Complex task A task may also have several inputs and outputs, and they may need to be filled differently, in a precise order. In the example for which the Gantt chart was given in Figure 1.7, the second step of the production is the carboxylation reaction. This reaction has two inputs; however, one of them needs to be heated up before the intermediate arrives. And when it does, the process must start immediately. After the process, the output can be held for as long as wanted, but after that the unit must be cleaned immediately. Modeling this complicated recipe can be done easily by the *e*S-graph, as illustrated in Figure 7.17. As it is shown in the figure, the subprocess of carboxylation has intersection with several other

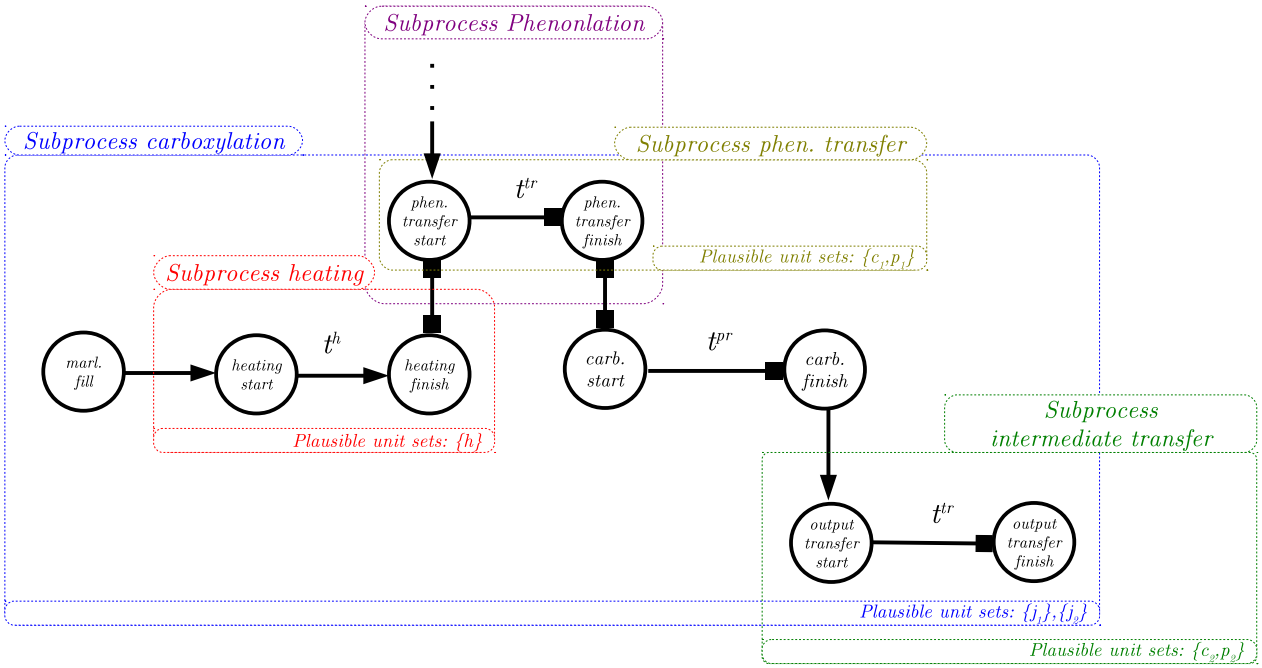


Figure 7.17: *e*Sgraph model of part of a complex recipe

subprocesses. As an example, after the marlotherm is filled, it needs to be heated up, for which an other unit is needed as well: a heater, labeled h. There are transfer subprocesses, shown with brown and green colors, analogously to the previous figures. Note that the transfer of phenolate is also part of the phenolation reaction, though it is not presented in detail in the figure.

Parallel resources The previous examples have already shown how multiple resources can be busy at the same time. There are two different ways of modeling this in the *e*S-graph framework:

- Having more resources in the plausible sets
- Overlapping subprocesses

This feature results in a modeling redundancy, i.e., it gives rise to different but mathematically correct formulations of the same problem. In most of the cases however, it is evident which one is the appropriate method. A good practice is to identify the subprocesses first, and assign the plausible units to them.

There is however case worth debating: let us assume that there is an operation which requires a machine and an operator to operate it. Also, it is assumed that there are 3 choices for both of them: m_1, m_2, m_3 and o_1, o_2, o_3, respectively. In this case, there are two options:

Option 1 A single subprocess "Operation" is created, with the plausible resource sets: $\{m_1, o_1\}$, $\{m_1, o_2\}$, $\{m_1, o_3\}$, $\{m_2, o_1\}$, $\{m_2, o_2\}$, $\{m_2, o_3\}$, $\{m_3, o_1\}$, $\{m_3, o_2\}$, $\{m_3, o_3\}$.

Option 2 Two subprocesses are created: "Operation-operator" and "Operation-machine", with plausible resource sets $\{o_1\}, \{o_2\}, \{o_3\}$ and $\{m_1\}, \{m_2\}, \{m_3\}$, respectively.

Both models are adequate and properly express all the options available in the system. It is easy to see that the latter one is more compact, and generally, it is more preferred. However, if several machines are allowed to work in parallel on the same subprocess, then there is an important question: should the number of assigned operators and machines be the same? If yes, option 2 can not be extended for that case, option 1 can.

Detailedness of the model An important feature of the *e*S-graph modeling framework is that the level of detail can depend on the problem at hand. By the level of detail the number of events associated to a subprocess is understood. As discussed above, a task may consists of only two events: arrival of the input material starts, and removal of the output material finishes, or it can include several other events as well. Whether an event is important to be included in the model or not depends on the actual problem.

Obviously, the events that are the boundaries of subprocesses must be included in the model. Also, if there is a sequence of events, for example, between which the weight of the dependencies never change, and either all of them are included in a subprocess or none of them, then only the first and the last event are important to be included in the model.

Note that having a more detailed model will never affect the soundness of the model, it will only unnecessarily increase the size of the model. It will usually also not have any effect on the computational performance. Thus including additional superfluous events is also suggested when it provides a more consistent, straightforward model.

Inclusion of the original S-graph framework

In the above examples it has been shown how detailed an *e*S-graph model can be. In this subsection, the *e*S-graph equivalent of the original S-graph models are given, which has a dual purpose:

- This model proves that everything, that could be modeled with the S-framework can be modeled in the new framework as well.
- The model shows an example that including many events is not necessary if the actual problem does not require it.

Note that the aim here is to provide the "smallest" *e*S-graph model. However, a more detailed model for real application is advisable.

The basic idea behind the model is that in the original S-graph framework a unit was busy with a task until the start of the next task. This will provide the subprocesses of the *e*S-graph model. The associated events will be the same as originally: the starts of the tasks and the removals of the products. All plausible resource sets will be singletons.

The definition of the *e*S-graph model of an S-graph can be given like this

$E = N = I \cup P$, i.e., the start of each task and the removal of products

$\mathcal{SP} = \{\{i\} \cup I_i^+ \mid i \in I\}$, i.e., a subprocess belongs to each task node, that includes all the other nodes (events) to which there leads a recipe arc

$D = A_1$, i.e., all the recipe arcs in the problem and nothing else

$J = J'$, i.e., all the units in the problem and nothing else

$\mathscr{O} = \{(\{i\} \cup I_i^+, \{\{j\} \mid j \in J_i\}) \mid i \in I\}$, i.e., to each of the subprocess all of the plausible units are assigned as singletons.

$\mathscr{W}$ this weight function works exactly as the original, and assigns the minimal available processing time to an arc as a lower bound. The upper bound is infinity except for the dependencies leading to the removal of the product.[11]

This model definition is illustrated on an example in Figure 7.18

The models generated this way can be solved to optimal makespan by the algorithm described in the next section.

7.5 General purpose makespan minimizer for the *e*S-graph

In this section a general purpose algorithm is shown for the *e*S-graph framework to minimize the makespan of any problem modeled with the *e*S-graph. The algorithm may not be the most efficient one for each problem class; the main aim is for it to stead as an illustration, how the new framework can be extended. It has to be noted, however, that the algorithm below explores an identical search space as the algorithm in block 2.1, if the problem is formulated according to the instructions in the previous section. The reason for this is that the main concept of the original makespan minimizer algorithm has been kept, and the details have just been adopted to the new model.

1. The algorithm first initializes the best makespan ($makespan^{cb}$) to infinity, and the set of open problems ($\mathcal{S}$) with the *e*S-graph model of the recipe.

2. In each iteration the *e*S-graph model of an open problem ($\mathbb{S}$) is selected from this set, and its bound is compared with the makespan of the current best solution. If the selected problem has a worse bound, it is pruned, and a new iteration starts. Otherwise:

   (a) If the problem is a solution, the best solution ($\mathbb{S}^{cb}$) and its makespan is updated.

[11] It would not change the soundness of the model if the upper bound were infinity in all of the cases.

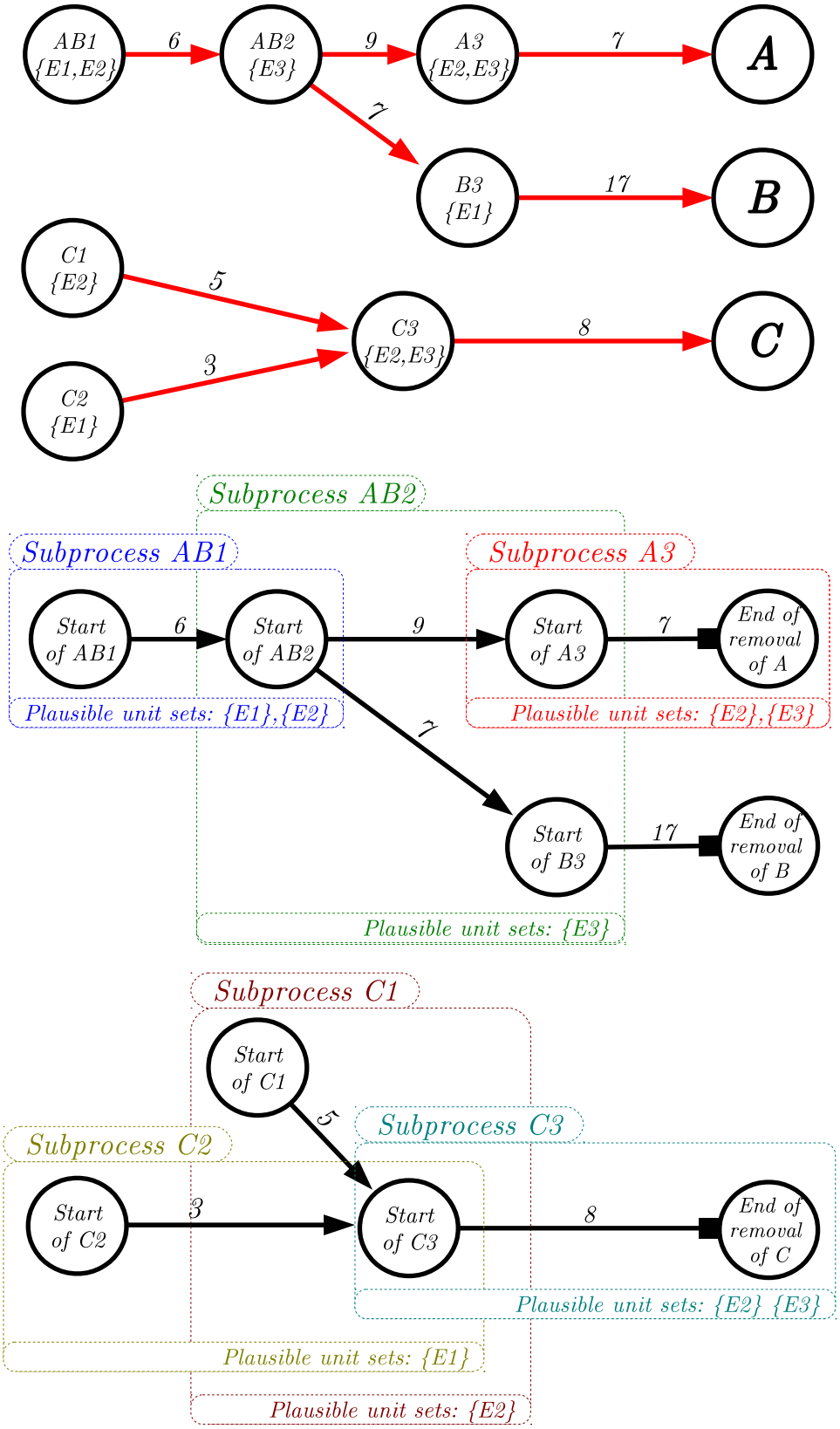


Figure 7.18: *e*Sgraph equivalent of an S-graph model

(b) An "active" resource is selected if possible, and for all subprocesses to which it still can be assigned, a subproblem is created, and the subprocesses are sequenced as last into the queue of the selected resource. Each new partial schedule is tested for feasibility, and then added to the set of open problems.

3. If the set of open subproblems is empty, and the current best makespan is not infinity, then the optimal solution is reported; otherwise, the problem has no feasible solution.

As it can be seen, the frame of the algorithm is the same, only some of the terms are changed. The formal algorithm is showed in block 7.1. For simpler description, the notation $S^+_{\mathscr{A}}(j)$ is introduced that denotes all of the subprocesses for a partial schedule whose assignment can be extended with j, formally:

$$S^+_{\mathscr{A}}(j) = \{S \in \mathcal{SP} \mid \exists O \in \mathscr{O}(S) \quad \{j\} \cup J_{\mathscr{A}}(S) \subseteq O\}$$

If this set is empty, the unit can not be assigned to any subprocesses anymore, and it is termed not active.

Algorithm 7.1 Makespan minimization with the S-graph framework

```
makespan^cb := ∞
S := {(erecipe(), ∅, ∅)}
while S ≠ ∅ do
  S = (E, SP, D, J, O, W, A, S) :=select_remove(S)
  if bound(S)< makespan^cb then
    if J_A(S) ∈ O(S)   ∀S ∈ SP then
      makespan^cb :=bound(S)
      S^cb := S
    end if
    j :=select({j ∈ J | S^+_A(j) ≠ ∅)
    if j ≠ NIL then
      for all S ∈ S^+_A(j) do
        A^S := A ∪ {(j, S)}
        S^S := S ∪ {(S', j, S) | S' ∈ S_A(j)}
        S^S := (E, SP, D, J, O, W, A^S, S^S)
        if feasible(S^S) then
          S := S ∪ {S}
        end if
      end for
    end if
  end if
end while
if makespan^cb ≠ ∞ then
  return S^cb
end if
```

Several notes on the subprocedures called in the algorithm:

select_remove As in the original case, this function is allowed to select and remove an open subproblem arbitrarily.

bound This procedure may return the longest path in $G(\mathbb{S})$.

select This function can arbitrarily return any active units, or NIL when none exists

feasible This procedure will definitely return false if there is a positive weighted cycle in $G(\mathbb{S})$. How it handles the zero-weighted cycles can depend on the problem class.

Note, that there are some major differences compared to the algorithm in block 2.1:

- Unlike $G(N, A_1, A_2)$, $\mathbb{S}$ contains all the scheduling decisions made in the subproblem, so no other parameters are needed.
- The algorithm does not modify directly the inner graph model of a subproblem. It is carried out as an inner mechanism of the model itself, when $G(\mathbb{S})$ is generated or maintained.
- The condition for a solution and for a leaf problem is not the same, i.e., even if the actual subproblem is a feasible solution, it may have children in the B&B tree. This is due to the fact that a subprocess may have plausible resource sets that are the subsets of each other.

Summary and concluding remarks

The S-graph framework has been introduced for the scheduling of batch processes nearly two decades ago. Next to its computational power, a great advantage of the framework is the straight-forward relation between the real problem, and the mathematical model. On the other hand, the simplicity of the S-graph model limits its applicability, and requires continuous extensions. In this chapter, a generalized, new framework is presented, which is based on the concepts of the original S-graph framework, and keeps its advantages, while providing a much wider range of modeling options. As one of the most important new building stones of the framework are the events of the process, it is named as the event-based S-graph framework, or *e*S-graph framework. The sections illustrated the necessity of this extension, the new model itself, the modeling techniques associated with it, and a general purpose makespan minimizer algorithm. Once the *e*S-graph framework is implemented, it will provide a tool which can be applied for a much wider range of scheduling problems with ease, and without any modification.

Related publication

- Hegyhati, M., F. Friedler, Combinatorial Algorithms of the S-Graph Framework for Batch Scheduling, Industrial & Engineering Chemistry Research, 50 (9), 5169-5174 (2011). [IF = 2.237]

Related conference presentation

- Holczinger, T. ,M. Hegyhati, F. Friedler, S-graph as an integrated modelling and optimization tool for the scheduling of industrial batch processes, presented at: VOCAL 2010, Veszprem, Hungary, December 13-15, 2010.

Appendix A

Environment for comparisons

All the empyrical test in the thesis were carried out on an IBM server with ***Intel Xeon E5504*** processor (4 physical cores, 2 GHz), and 8 GiB RAM. The operating system was Ubuntu 12.04 LTS. The MILP formulations were solved by Gurobi solver, and the S-graph solver applies the COIN-OR library for solving LP models.

In many cases, the approaches were not able to solve the larger problems in reasonable time. For many tests, a time limit of 1000 seconds were set for the solvers, which is a well accepted approach among the researchers of this topic. If the execution of an approach reached this time limit, the reported objective is the best found in that time interval.

Appendix B

Nomenclature

List of the notations used thorough the document

Parameters of the problem

Sets

P Set of products

I Set of tasks

J Set of units

Indexed sets

I_p set of tasks taking part int the production of product $p \in P$

I_i^+ set of subsequent tasks of task $i \in I$ (for precedential recipe)

I_i^- set of predecessor tasks of task $i \in I$ (for precedential recipe)

I_j set of tasks that can be performed by unit $j \in J$

J_i set of units that can perform task $i \in I$

Paramters

$t_{i,j}^{pr}$ rocessing time of task $i \in I$ in $j \in J_i$

t^{tr} transfer time, whose indices can be different based on the context (just a single material or a unit-material-unit triple, etc.)

t^H the time horizon

Other notations

p_i is the product to which i belong, if it is unambiguous

i_i^+ subsequent task of task $i \in I$, if exists (for sequential recipes)

i_i^- predecessor task of task $i \in I$, if exists (for sequential recipes)

$i_{p,k}$ kth task of the production of product $p \in P$ (for sequential recipes)

n_P number of stages for product $p \in P$ (for sequential recipes)

Throughput maximization

R_p Revenue for one batch of product p

Stochastic problems

Simple recipes

b_p the number of batches for product p in the actual configuration

s_p the size of a batch for product p (for the first case)

s_p^{min}, s_p^{max} the minimal and maximal size of a batch for product p (for the second and third case)

S the set of scenarios

$prob_s$ the probability of scenario $s \in S$

$dem_{s,p}$ the demand for product p in scenario $s \in S$

$price_{s,p}$ the price of p in scenario $s \in S$

$oc_{s,p}, uc_{s,p}$ the over- and underproduction cost of p in scenario $s \in S$

$Profit_{s,p}(x)$ is the profit for x amount of product p in scenario s.

$ExpProfit_p(x)$ is the expected profit of $s_p \cdot b_p$ amount of product p

Complex recipes

R set of recipes

P_r set of products produced by recipe $r \in R$

R_p set of recipes producing produt $p \in P$

b_r the number of batches for recipe $r \in R$ in the actual configuration

$s_{r,p}$ the maximal amount of $p \in P$ that can be produced with the recipe $r \in R$

min_r the minimal proportion ration on which the recipe $r \in R$ can be executed

Continuous case

f probability distribution function

F cumulative probability distribution function

S-graph

$G(N, A_1, A_2, w)$

$N := I \cup P$, the set of nodes

$A_1 := \{(i, i') | i \in I \quad i' \in I_i^+\}$, the set of recipe arcs

$A_2 := \emptyset$, the set of schedule arcs

$w_{i,i'} := \min_{j \in I_j} t_{i,j}^{pr}$, the weights for all recipe arc $(i, i') \in A_1$: the minimal processing time for i

eS-graph

E is the set of events

$\mathcal{SP} \subseteq \wp(E)$ is a set of subprocesses

D is the set of dependencies between events

J is the set of units/resources to be scheduled

$\mathscr{O} \in \mathcal{SP} \to \wp(\wp(J))$ is a set of assignment options

$\mathscr{W}$ is a weight function for dependencies based on scheduling decisions.

$(\mathscr{A}, \mathscr{S})$ schedule

$\mathscr{A} \subseteq J \times \mathcal{SP}$ is a set of resource-subprocess assignements made so far.

$\mathscr{S} \subseteq \mathcal{SP} \times J \times \mathcal{SP}$ is a set of sequencing decisions made so far.

$\mathscr{A}(j) = \{S \mid (j, S) \in \mathscr{A}\} \quad \forall j \in J$

$J_{\mathscr{A}}(S) = \{j \mid (j, S) \in \mathscr{A}\} \quad \forall S \in \mathcal{SP}$

$S_1 \xrightarrow{j} S_2$ is true if $(S_1, j, S_2) \in \mathscr{S}$, false otherwise

$\overline{\mathscr{S}}$ is the transitive closure of $\mathscr{S}$,i.e., if $S_1 \xrightarrow{j} S_2$ and $S_2 \xrightarrow{j} S_3$ then $S_1 \xrightarrow{j} S_3$.

Appendix C

Full tables of test results

C.1 Throughput maximization

This section contains all the empirical results for the 18 throughput maximization. The columns are:

first Configuration selection strategy

second Update subroutine

third Feasibility subroutine

rest time horizon, and the CPU times in seconds

The time limit for the solver was 3600 seconds.

C.1.1 Pharmaceutical case study

			24	25	26	27	28	29	30
LEX	U	FT	0.1	0.12	0.12	0.11	0.16	4.56	13.42
		MM	3.5	3.98	3.98	3.98	5.36	21.49	3600
	F	FT	0.1	0.11	0.12	0.11	0.16	7.83	14.73
		MM	3.98	3.98	3.99	3.97	5.37	21.82	30.77
	E	FT	0.09	0.12	0.12	0.12	0.13	8.15	15.71
		MM	0.16	0.20	0.19	0.19	0.26	21.02	30.61
BFS	U	FT	0.13	0.13	0.13	0.13	0.16	5.57	15.23
		MM	6.91	4.30	4.30	4.27	5.36	30.95	3600
	F	FT	0.12	0.12	0.12	0.12	0.16	8.05	15.41
		MM	5.75	3.98	3.98	4.00	5.35	24.17	33.41
	E	FT	0.1	0.12	0.12	0.12	0.15	8.29	17.62
		MM	0.17	0.21	0.21	0.21	0.26	21.72	36.87
DFS	U	FT	0.21	0.27	0.27	0.27	0.5	7.01	14.78
		MM	7.30	11.93	11.81	11.90	19.82	35.96	52.41
	F	FT	0.2	0.27	0.27	0.27	0.51	9.16	18.63
		MM	7.05	11.87	11.96	11.88	19.98	36.17	53.55
	E	FT	0.1	0.12	0.12	0.12	0.13	8.58	17.09
		MM	0.17	0.21	0.20	0.21	0.26	27.99	39.49

			31	32	33	34	35	36	37
LEX	U	FT	53.56	129.79	96.83	446.93	448.2	1296.92	1306.86
		MM	3600	3600					
	F	FT	20	34.26	40.97	101.43	101.28	687.4	690.63
		MM	67.82	85.77					
	E	FT	23.12	36.29	52.5	55.52	54.48	71.36	78.82
		MM	47.00	67.74					
BFS	U	FT	63.21	177.01	267.69	534.91	538.71	1385.11	1386.55
		MM	3600	3600					
	F	FT	23.87	44.11	47.99	107.27	107.22	685.34	689.15
		MM	77.18	107					
	E	FT	29.38	48.93	63	57.58	56.8	74.85	80.81
		MM	72.77	99.26					
DFS	U	FT	64.67	139.17	209.1	404.51	406.47	2960.98	2966.28
		MM	3600	3600					
	F	FT	38.44	55.6	65.52	284.84	284.34	2978.23	2949.65
		MM	256	272					
	E	FT	27.37	42.97	56.66	59.26	58.38	75.81	82.37
		MM	69.86	88.19					

C.1.2 Agrochemical example

			13	14	15	16
LEX	U	FT	3.06	339.45	453.11	3600
		MM	3600	3600	3600	3600
	F	FT	3.07	340.04	451.78	3600
		MM	3600	3600	3600	3600
	E	FT	3.05	205.05	453.93	3600
		MM	3600	3600	3600	3600
BFS	U	FT	3.08	405.79	583.83	3600
		MM	3600	3600	3600	3600
	F	FT	3.04	405.01	582.28	3600
		MM	3600	3600	3600	3600
	E	FT	3.05	414.03	591.55	3600
		MM	3600	3600	3600	3600
DFS	U	FT	3.03	405.13	589.11	3600
		MM	3600	3600	3600	3600
	F	FT	3.08	402.99	583.25	3600
		MM	3600	3600	3600	3600
	E	FT	3.07	269.34	581.33	3600
		MM	3600	3600	3600	3600

C.1.3 Literature example

			14	15	16	17
LEX	U	FT	16.88	179.21	1260.14	3600
		MM	276.5	2063.53	3600	3600
	F	FT	15.84	134.7	1159.3	3600
		MM	272.52	848.64	3600	3600
	E	FT	15.21	113.79	984.94	3600
		MM	30.07	484.25	3600	3600
BFS	U	FT	14.95	208.14	964.05	3600
		MM	3600	3600	3600	3600
	F	FT	18.52	144.16	1496.75	3600
		MM	768.76	929.27	3600	3600
	E	FT	21.07	133.26	1512.81	3600
		MM	236.08	689.51	3600	3600
DFS	U	FT	22.63	437.39	1594.05	3600
		MM	1313.48	3600	3600	3600
	F	FT	22.53	279.89	1629.85	3600
		MM	1308.07	3600	3600	3600
	E	FT	21.01	225.55	1403.72	3600
		MM	459.21	2053.05	3600	3600

C.2 Zero-wait test results

CPU times of the ZW tests

Conf.	sLP	aLP	aLP'	Neg	Rec	Rec+
11111	0.035	0.096	0.061	0.008	0.009	0.005
11112	0.099	0.139	0.225	0.007	0.017	0.009
11121	0.047	0.178	0.157	0.008	0.016	0.007
11211	0.102	0.398	0.666	0.02	0.049	0.027
12111	2.959	0.691	1.397	0.054	0.049	0.055
21111	0.099	0.47	0.264	0.019	0.026	0.014
22111	22.71	4.457	10.636	0.21	0.323	0.34
22211	188.55	40.695	83.453	1.592	3.014	4.618
22221	437.386	71.556	225.756	2.689	5.785	8.523
22222	1000	125.454	403.037	6.964	14.361	13.592
32222	1000	1000	1000	57.545	133.474	102.853
33333	1000	1000	1000	1000	1000	1000
44444	1000	1000	1000	1000	1000	1000
55555	1000	1000	1000	1000	1000	1000

CPU times of the ZW tests

Conf.	sLP	aLP	aLP'	Neg	Rec	Rec+
11111	18.2	18.2	18.2	18.2	18.2	18.2
11112	18.2	18.2	18.2	18.2	18.2	18.2
11121	18.2	18.2	18.2	18.2	18.2	18.2
11211	22.7	22.7	22.7	22.7	22.7	22.7
12111	22	22	22	22	22	22
21111	18.2	18.2	18.2	18.2	18.2	18.2
22111	22	22	22	22	22	22
22211	26.5	26.5	26.5	26.5	26.5	26.5
22221	26.5	26.5	26.5	26.5	26.5	26.5
22222	47.5	27.1	27.1	27.1	27.1	27.1
32222	49.1	27.9	27.9	27.9	27.9	27.9
33333	69.8	42.9	58.7	37.9	41	37.9
44444	57.5	60.4	inf	57.2	62.2	57.2
55555	87.1	98.8	inf	79	79	79

C.3 Expected profits of feasible configurations for the stochastic tests

Batch number configurations						Expected profits		
						Two-stage	Flexible	Fixed
1	0	3	0	0	1	2689.87	2475.31	2451.15

Batch number configurations						Expected profits		
						Two-stage	Flexible	Fixed
1	0	3	0	1	0	2661.76	2474.58	2474.58
0	0	0	0	0	3	2643.56	2465.39	2465.39
1	0	1	0	0	2	2643.56	2465.39	2465.39
2	0	2	0	0	1	2643.56	2465.39	2465.39
0	1	1	0	1	1	2625.29	2456.09	2456.09
1	1	0	0	0	2	2607.03	2446.78	2446.78
2	1	1	0	0	1	2607.03	2446.78	2446.78
0	0	2	1	0	1	2597.9	2442.13	2442.13
1	0	3	1	0	0	2597.9	2442.13	2442.13
0	0	3	0	0	1	2586.99	2435.7	2435.7
1	0	4	0	0	0	2586.99	2435.7	2435.7
1	0	0	1	1	1	2586.97	2435.68	2435.68
0	0	0	0	1	2	2574.73	2427.92	2427.92
1	0	1	0	1	1	2574.73	2427.92	2427.92
2	0	2	0	1	0	2574.73	2427.92	2427.92
2	2	0	0	0	1	2562.47	2420.14	2420.14
2	0	0	0	0	2	2550.26	2412.41	2412.41
3	0	1	0	0	1	2550.26	2412.41	2412.41
0	1	1	1	0	1	2550.24	2412.39	2412.39
1	1	2	1	0	0	2550.24	2412.39	2412.39
0	1	1	0	2	0	2550.21	2412.36	2412.36
0	1	2	0	0	1	2538	2404.63	2404.63
1	1	3	0	0	0	2538	2404.63	2404.63
1	1	0	0	1	1	2525.74	2396.85	2396.85
2	1	1	0	1	0	2525.74	2396.85	2396.85
0	0	1	2	1	0	2525.74	2396.85	2396.85
0	0	2	1	1	0	2513.51	2389.09	2389.09
0	0	3	0	1	0	2501.28	2381.34	2381.34
1	0	0	2	0	1	2501.28	2381.34	2381.34
3	1	0	0	0	1	2501.28	2381.34	2381.34
1	0	0	1	2	0	2501.24	2381.32	2381.32
0	0	0	1	0	2	2489.04	2373.58	2373.58
1	0	1	1	0	1	2489.04	2373.58	2373.58
2	0	2	1	0	0	2489.04	2373.58	2373.58
0	2	1	0	0	1	2489.01	2373.56	2373.56
1	2	2	0	0	0	2489.01	2373.56	2373.56
0	0	0	0	2	1	2489.01	2373.56	2373.56
1	0	1	0	2	0	2489.01	2373.56	2373.56
0	0	1	0	0	2	2476.81	2365.82	2365.82
1	0	2	0	0	1	2476.81	2365.82	2365.82
2	0	3	0	0	0	2476.81	2365.82	2365.82

Batch number configurations						Expected profits		
						Two-stage	Flexible	Fixed
2	2	0	0	1	0	2476.75	2365.78	2365.78
2	0	0	0	1	1	2464.55	2358.05	2358.05
3	0	1	0	1	0	2464.55	2358.05	2358.05
0	1	1	1	1	0	2464.52	2358.02	2358.02
0	1	2	0	1	0	2452.2	2350.15	2350.15
1	1	0	1	0	1	2436.21	2337.14	2337.14
2	1	1	1	0	0	2436.21	2337.14	2337.14
0	0	1	3	0	0	2436.21	2337.14	2337.14
1	1	0	0	2	0	2436.18	2337.12	2337.12
0	1	0	0	0	2	2420.22	2324.14	2324.14
1	1	1	0	0	1	2420.22	2324.14	2324.14
2	1	2	0	0	0	2420.22	2324.14	2324.14
0	0	2	2	0	0	2420.22	2324.14	2324.14
0	0	3	1	0	0	2404.23	2311.13	2311.13
3	1	0	0	1	0	2404.2	2311.11	2311.11
1	0	0	2	1	0	2404.2	2311.11	2311.11
0	0	4	0	0	0	2388.24	2298.12	2298.12
0	0	0	1	1	1	2388.21	2298.1	2298.1
1	0	1	1	1	0	2388.21	2298.1	2298.1
0	2	1	0	1	0	2388.18	2298.08	2298.08
0	0	0	0	3	0	2388.18	2298.08	2298.08
0	0	1	0	1	1	2372.22	2285.1	2285.1
1	0	2	0	1	0	2372.22	2285.1	2285.1
2	2	0	1	0	0	2372.19	2285.07	2285.07
2	0	0	1	0	1	2356.23	2272.09	2272.09
3	0	1	1	0	0	2356.23	2272.09	2272.09
1	2	0	0	0	1	2356.2	2272.07	2272.07
2	2	1	0	0	0	2356.2	2272.07	2272.07
0	1	1	2	0	0	2356.2	2272.07	2272.07
2	0	0	0	2	0	2356.2	2272.07	2272.07
1	0	0	0	0	2	2340.24	2259.08	2259.08
2	0	1	0	0	1	2340.24	2259.08	2259.08
3	0	2	0	0	0	2340.24	2259.08	2259.08
0	1	2	1	0	0	2340.21	2259.06	2259.06
0	1	3	0	0	0	2324.22	2246.05	2246.05
1	1	0	1	1	0	2324.19	2246.03	2246.03
0	1	0	0	1	1	2308.2	2233.03	2233.03
1	1	1	0	1	0	2308.2	2233.03	2233.03
2	3	0	0	0	0	2292.18	2220	2220
2	1	0	0	0	1	2276.22	2207.01	2207.01
3	1	1	0	0	0	2276.22	2207.01	2207.01

Batch number configurations						Expected profits		
						Two-stage	Flexible	Fixed
0	0	0	2	0	1	2276.22	2207.01	2207.01
1	0	1	2	0	0	2276.22	2207.01	2207.01
0	2	1	1	0	0	2276.19	2206.99	2206.99
0	0	0	1	2	0	2276.19	2206.99	2206.99
0	0	1	1	0	1	2260.23	2194.01	2194.01
1	0	2	1	0	0	2260.23	2194.01	2194.01
0	2	2	0	0	0	2260.2	2193.98	2193.98
0	0	1	0	2	0	2260.2	2193.98	2193.98
0	0	2	0	0	1	2244.24	2181	2181
1	0	3	0	0	0	2244.24	2181	2181
2	0	0	1	1	0	2244.21	2180.98	2180.98
1	2	0	0	1	0	2244.18	2180.96	2180.96
1	0	0	0	1	1	2228.22	2167.97	2167.97
2	0	1	0	1	0	2228.22	2167.97	2167.97
3	2	0	0	0	0	2212.2	2154.94	2154.94
1	1	0	2	0	0	2212.2	2154.94	2154.94
3	0	0	0	0	1	2196.24	2141.96	2141.96
4	0	1	0	0	0	2196.24	2141.96	2141.96
0	1	0	1	0	1	2196.21	2141.94	2141.94
1	1	1	1	0	0	2196.21	2141.94	2141.94
0	3	1	0	0	0	2196.18	2141.91	2141.91
0	1	0	0	2	0	2196.18	2141.91	2141.91
0	1	1	0	0	1	2180.22	2128.93	2128.93
1	1	2	0	0	0	2180.22	2128.93	2128.93
2	1	0	0	1	0	2164.2	2115.9	2115.9
0	0	0	2	1	0	2164.2	2115.9	2115.9
0	0	1	1	1	0	2148.21	2102.89	2102.89
4	1	0	0	0	0	2132.22	2089.89	2089.89
0	0	2	0	1	0	2132.22	2089.89	2089.89
2	0	0	2	0	0	2132.22	2089.89	2089.89
1	2	0	1	0	0	2132.19	2089.87	2089.87
1	0	0	1	0	1	2116.23	2076.88	2076.88
2	0	1	1	0	0	2116.23	2076.88	2076.88
0	2	0	0	0	1	2116.2	2076.86	2076.86
1	2	1	0	0	0	2116.2	2076.86	2076.86
1	0	0	0	2	0	2116.2	2076.86	2076.86
0	0	0	0	0	2	2100.24	2063.87	2063.87
1	0	1	0	0	1	2100.24	2063.87	2063.87
2	0	2	0	0	0	2100.24	2063.87	2063.87
3	0	0	0	1	0	2081.68	2047.3	2047.3
0	1	0	1	1	0	2081.65	2047.28	2047.28

Batch number configurations						Expected profits		
						Two-stage	Flexible	Fixed
0	1	1	0	1	0	2059.66	2026.78	2026.78
2	1	0	1	0	0	2036.81	2005.42	2005.42
0	0	0	3	0	0	2036.81	2005.42	2005.42
1	3	0	0	0	0	2036.76	2005.38	2005.38
1	1	0	0	0	1	2013.96	1984.06	1984.06
2	1	1	0	0	0	2013.96	1984.06	1984.06
0	0	1	2	0	0	2013.96	1984.06	1984.06
0	0	2	1	0	0	1991.11	1962.7	1962.7
0	0	3	0	0	0	1968.26	1941.34	1941.34
1	0	0	1	1	0	1968.21	1941.3	1941.3
0	2	0	0	1	0	1968.15	1941.25	1941.25
0	0	0	0	1	1	1945.36	1919.94	1919.94
1	0	1	0	1	0	1945.36	1919.94	1919.94
3	0	0	1	0	0	1922.51	1898.58	1898.58
0	1	0	2	0	0	1922.45	1898.53	1898.53
2	2	0	0	0	0	1922.45	1898.53	1898.53
2	0	0	0	0	1	1899.66	1877.22	1877.22
3	0	1	0	0	0	1899.66	1877.22	1877.22
0	1	1	1	0	0	1899.6	1877.17	1877.17
0	1	2	0	0	0	1876.75	1855.81	1855.81
1	1	0	0	1	0	1853.85	1834.41	1834.41
3	1	0	0	0	0	1808.15	1791.69	1791.69
1	0	0	2	0	0	1808.15	1791.69	1791.69
0	2	0	1	0	0	1808.09	1791.64	1791.64
0	0	0	1	0	1	1785.3	1770.33	1770.33
1	0	1	1	0	0	1785.3	1770.33	1770.33
0	2	1	0	0	0	1785.24	1770.28	1770.28
0	0	0	0	2	0	1785.24	1770.28	1770.28
0	0	1	0	0	1	1762.45	1748.97	1748.97
1	0	2	0	0	0	1762.45	1748.97	1748.97
2	0	0	0	1	0	1739.54	1727.56	1727.56
4	0	0	0	0	0	1693.84	1684.84	1684.84
1	1	0	1	0	0	1693.79	1684.8	1684.8
0	3	0	0	0	0	1693.73	1684.75	1684.75
0	1	0	0	0	1	1670.94	1663.44	1663.44
1	1	1	0	0	0	1670.94	1663.44	1663.44
0	0	0	1	1	0	1625.18	1620.67	1620.67
0	0	1	0	1	0	1597.22	1594.2	1594.2
2	0	0	1	0	0	1569.13	1567.59	1567.59
1	2	0	0	0	0	1569.07	1567.55	1567.55
1	0	0	0	0	1	1541.03	1540.98	1540.98

Batch number configurations						Expected profits		
						Two-stage	Flexible	Fixed
2	0	1	0	0	0	1541.03	1540.98	1540.98
0	1	0	0	1	0	1481.51	1481.51	1481.51
0	0	0	2	0	0	1422.08	1422.08	1422.08
2	1	0	0	0	0	1422.08	1422.08	1422.08
0	0	1	1	0	0	1392.37	1392.37	1392.37
0	0	2	0	0	0	1360.07	1360.07	1360.07
1	0	0	0	1	0	1325.04	1325.04	1325.04
3	0	0	0	0	0	1255.11	1255.11	1255.11
0	1	0	1	0	0	1255.04	1255.04	1255.04
0	1	1	0	0	0	1220.08	1220.08	1220.08
1	0	0	1	0	0	1077.29	1077.29	1077.29
0	2	0	0	0	0	1077.19	1077.19	1077.19
0	0	0	0	0	1	1039.22	1039.22	1039.22
1	0	1	0	0	0	1039.22	1039.22	1039.22
1	1	0	0	0	0	886.77	886.77	886.77
0	0	0	0	1	0	772.48	772.48	772.48
2	0	0	0	0	0	696.34	696.34	696.34
0	0	0	1	0	0	505.83	505.83	505.83
0	0	1	0	0	0	467.76	467.76	467.76
0	1	0	0	0	0	315.31	315.31	315.31
1	0	0	0	0	0	124.89	124.89	124.89

Bibliography

[1] R. Adonyi, G. Biros, T. Holczinger, and F. Friedler. "Effective scheduling of a large-scale paint production system". In: *Journal of Cleaner Production* 16 (2) (2008), pp. 225–232.

[2] R. Adonyi, J. Romero, L. Puigjaner, and F. Friedler. "Incorporating heat integration in batch process scheduling". In: *Applied Thermal Engineering* 23 (2003), pp. 1743–1762.

[3] R. Adonyi. "Batch process scheduling with the extensions of the S-graph framework". PhD thesis. Doctoral School of Information Science and Technology - University of Veszprem, 2008.

[4] R. Adonyi, I. Heckl, A. Szalamin, and F. Olti. "Routing of Railway Systems with the S-graph Framework for Effective Scheduling". In: *Chemical Engineering Transactions* 21, Jiri Klemes and Hon Loong Lam and Petar Varbanov (2010), pp. 913–918.

[5] A. M. Aguirre, C. A. Méndez, and P. M. Castro. "A novel optimization method to automated wet-etch station scheduling in semiconductor manufacturing systems". In: *Computers & Chemical Engineering* 35.12 (2011), pp. 2960–2972.

[6] A. M. Aguirre, C. A. Méndez, Á. García-Sánchez, M. Ortega-Mier, and P. M. Castro. "General Framework for Automated Manufacturing Systems: Multiple Hoists Scheduling Solution". In: *Chemical Engineering Transactions* 32 (2013), pp. 1381–1386.

[7] A. M. Aguirre, C. A. Méndez, G. Gutierrez, and C. D. Prada. "An improvement-based MILP optimization approach to complex AWS scheduling". In: *Computers & Chemical Engineering* 47 (2012), pp. 217–226.

[8] R. Alur. "Timed Automata". In: *Theoretical Computer Science* 126 (1999), pp. 183–235.

[9] R. Alur and D. L. Dill. "A theory of timed automata". In: *Theoretical Computer Science* 126.2 (Apr. 1994), pp. 183–235.

[10] K. Arrow, T. Harris, and J. Marshak. "Optimal Inventory Policy". In: *Econometrica* 19.3 (1951), pp. 250–272.

[11] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, and J. Romijn. "Efficient Guiding Towards Cost-Optimality in UPPAAL". In: *7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01). Genova, Italy, April 2 to 6, 2001. LNCS 2031.* 2001, pp. 174–188.

[12] S. Bhushan and I. A. Karimi. "An MILP Approach to Automated Wet-Etch Station Scheduling". In: *Industrial & Engineering Chemistry Research* 42.7 (2003), pp. 1391–1399.

[13] S. Bhushan and I. Karimi. "Heuristic algorithms for scheduling an automated wet-etch station". In: *Computers & Chemical Engineering* 28.3 (2004), pp. 363–379.

[14] A. Bonfill, A. Espuna, and L. Puigjaner. "Proactive approach to address the uncertainty in short-term scheduling". In: *Computers & Chemical Engineering* 32.8 (Aug. 2008), pp. 1689–1706.

[15] E. Capón-García, M. Moreno-Benito, and A. Espuna. "Improved Short-Term Batch Scheduling Flexibility Using Variable Recipes". In: *Industrial & Engineering Chemistry Research* 50.9 (2011), pp. 4983–4992.

[16] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Springer-Link Engineering. Springer, 2008. ISBN: 9780387333328.

[17] P. M. Castro, A. P. Barbosa-Povoa, H. A. Matos, and A. Q. Novais. "Simple Continuous-Time Formulation for Short-Term Scheduling of Batch and Continuous Processes". In: *Industrial & Engineering Chemistry Research* 43 (2004), pp. 105–118.

[18] P. M. Castro, L. J. Zeballos, and C. A. Méndez. "Hybrid time slots sequencing model for a class of scheduling problems". In: *AIChE Journal* 58.3 (2012), pp. 789–800.

[19] C.-L. Chen and C.-Y. Chang. "A resource-task network approach for optimal short-term/periodic scheduling and heat integration in multipurpose batch plants". In: *Applied Thermal Engineering* 29.5-6 (Apr. 2009), pp. 1195–1208.

[20] C.-L. Chen, C.-Y. Chang, and J.-Y. Lee. "Resource-Task Network Approach to Simultaneous Scheduling and Water Minimization of Batch Plants". In: *Industrial & Engineering Chemistry Research* (Oct. 2011).

[21] F. Corman, A. D'Ariano, D. Pacciarelli, and M. Pranzo. "Optimal inter-area coordination of train rescheduling decisions". In: *Transportation Research Part E: Logistics and Transportation Review* (2011), pages.

[22] F. Corman, A. D'Ariano, D. Pacciarelli, and M. Pranzo. "Optimal inter-area coordination of train rescheduling decisions". In: *Procedia - Social and Behavioral Sciences* 17 (2011), pp. 58–81.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* 3rd. MIT Press, 2009.

[24] A. D'Ariano, D. Pacciarelli, and M. Pranzo. "A branch and bound algorithm for scheduling trains in a railway network". In: *European Journal of Operational Research* 183.2 (Dec. 2007), pp. 643–657.

[25] A. D'Ariano, D. Pacciarelli, and M. Pranzo. "Assessment of flexible timetables in real-time traffic management of a railway bottleneck". In: *Transportation Research Part C: Emerging Technologies* 16.2 (Apr. 2008), pp. 232–245.

[26] C. Delort and O. Spanjaard. "Using bound sets in multiobjective optimization: Application to the biobjective binary knapsack problem". In: *9th International Symposium on Experimental Algorithms (SEA 2010)*. Vol. 6049. Lecture Notes in Computer Science. 2010, pp. 253–265.

[27] D. Dill. "Timing assumptions and verification of finite-state concurrent systems". In: *Proc. Automatic Verification Methods for Finite State Systems*. Ed. by J. Sifakis. Vol. 407. LNCS. Springer, 1990, pp. 197–212.

[28] E.-L. Dogaru and V. Lavric. "Dynamic Water Network Topology Optimization of Batch Processes". In: *Industrial & Engineering Chemistry Research* (Aug. 2011).

[29] F. Y. Edgeworth. "The Mathematical Theory of Banking". In: *Journal of the Royal Statistical Society* 51 (1) (1888), pp. 113–127.

[30] A. Éles. "Decision supporting software for the scheduling of batch processes (In Hungarian, original title: Ütemezési feladatmegoldó módszer választását támogató szoftver)". Bachelor thesis. Faculty of Information Technology, University of Pannonia, Jan. 2014.

[31] S. Ferrer-Nadal, E. Capón-García, C. A. Méndez, and L. Puigjaner. "Material transfer operations in batch scheduling. A critical modeling issue". In: *Industrial & Engineering Chemistry Research* 47 (2008), pp. 7721–7732.

[32] C. A. Floudas and X. Lin. "Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review". In: *Computers & Chemical Engineering* 28.11 (Oct. 2004), pp. 2109–2129.

[33] F. Friedler, K. Tarjan, Y. W. Huang, and L. T. Fan. "Graph-Theoretic Approach to Process Synthesis: Axioms and Theorems". In: *Chem. Engng Sci.* 47 (1992), pp. 1973–1988.

[34] F. Friedler, K. Tarjan, Y. Huang, and L. Fan. "Graph-theoretic approach to process synthesis: Polynomial algorithm for maximal structure generation". In: *Computers & Chemical Engineering* 17.9 (Sept. 1993), pp. 929–942.

[35] F. Friedler, J. B. Varga, E. Feher, and L. T. Fan. "Nonconvex Optimization and Its Applications, State of the Art in Global Optimization, Computational Methods and Applications". In: ed. by C. A. Floudas and P. M. Pardalos. Kluwer Academic Publishers, 1996. Chap. Combinatorially Accelerated Branch-and-Bound Method for Solving the MIP Model of Process Network Synthesis, pp. 609–626.

[36] F. Friedler, J. Varga, and L. Fan. "Decision-mapping: A tool for consistent and complete decisions in process synthesis". In: *Chemical Engineering Science* 50.11 (June 1995), pp. 1755–1768.

[37] C. D. Geiger, K. G. Kempf, and R. Uzsoy. "A Tabu search approach to scheduling an automated wet etch station". In: *Journal of Manufacturing Systems* 16.2 (1997), pp. 102–116.

[38] M. C. Georgiadis and L. G. Papageorgiou. "Optimal scheduling of heat-integrated multipurpose plants under fouling conditions". In: *Applied Thermal Engineering* 21.16 (Nov. 2001), pp. 1675–1697.

[39] M. Ghaeli, P. A. Bahri, P. Lee, and T. Gu. "Petri-net based formulation and algorithm for short-term scheduling of batch plants". In: *Computers & Chemical Engineering* 29.2 (Jan. 2005), pp. 249–259.

[40] J. F. Gouws and T. Majozi. "Usage of inherent storage for minimisation of wastewater in multipurpose batch plants". In: *Chemical Engineering Science* 64.16 (Aug. 2009), pp. 3545–3554.

[41] R. Grau, M. Graells, J. Corominas, A. Espuna, and L. Puigjaner. "Global strategy for energy and waste analysis in scheduling and planning of multiproduct batch chemical processes". In: *Computers & Chemical Engineering* 20.6-7 (1996), pp. 853–868.

[42] I. E. Grossmann. "Review of Nonlinear Mixed-Integer and Disjunctive Programming Techniques". In: *Optimization and Engineering* 3 (2002), pp. 227–252.

[43] I. Halim and R. Srinivasan. "Sequential methodology for integrated optimization of energy and water use during batch process scheduling". In: *Computers & Chemical Engineering* 35.8 (Aug. 2011), pp. 1575–1597.

[44] I. Halim and R. Srinivasan. "Sequential Methodology for Simultaneous Batch Process Scheduling and Water Reuse Optimization". In: *Chemical Engineering Transactions* 21 (2010), pp. 727–732.

[45] M. Hegyháti, T. Majozi, T. Holczinger, and F. Friedler. "Practical infeasibility of cross-transfer in batch plants with complex recipes: S-graph vs MILP methods". In: *Chemical Engineering Science* 64.3 (2009), pp. 605–610.

[46] M. Hegyhati and F. Friedler. "Combinatorial Algorithms of the S-Graph Framework for Batch Scheduling". In: *Industrial & Engineering Chemistry Research* 50.9 (May 2011), pp. 5169–5174.

[47] M. Hegyhati and F. Friedler. "In-Depth Study and Comparison of S-Graph Framework and Precedence Based MILP Formulations for Batch Process Scheduling". In: *AIChE Annual Meeting*. Oct. 2011.

[48] M. Hegyhati and F. Friedler. "Overview of Industrial Batch Process Scheduling". In: *Chemical Engineering Transactions* 21, Jiri Klemes and Hon Loong Lam and Petar Varbanov (2010), pp. 895–900.

[49] M. Hegyhati and F. Friedler. "Tools of Discrete Event Systems for Batch Process Scheduling". In: *CAPE forum*. Veszprem, Hungary, Mar. 2013.

[50] M. Hegyhati, T. Holczinger, A. Szoldatics, and F. Friedler. "Combinatorial Approach to Address Batch Scheduling Problems with Limited Storage Time". In: *Chemical Engineering Transactions* 25 (2011), pp. 495–500.

[51] T. Holczinger, J. Romero, L. Puigjaner, and F. Friedler. "Scheduling of Multipurpose Batch Processes with Multiple Batches of the Products". In: *Hung. J. Ind. Chem.* 30 (2002), pp. 305–312.

[52] T. Holczinger. "Method for scheduling non-intermediate storage batch process systems". PhD thesis. Doctoral School of Information Science and Technology - University of Veszprem, 2004.

[53] T. Holczinger. *OWI ZALA igényfelmérés*. Tech. rep. University of Pannonia, 2008.

[54] T. Holczinger, M. Hegyhati, and F. Friedler. "Simultaneous Heat Integration and Batch Process Scheduling". In: *Chemical Engineering Transactions* 29 (2012), pp. 337–342.

[55] T. Holczinger, T. Majozi, M. Hegyhati, and F. Friedler. "An automated algorithm for throughput maximization under fixed time horizon in multipurpose batch plants: S-Graph approach". In: *Computer Aided Chemical Engineering*. Computer Aided Chemical Engineering 24 (2007). Ed. by V. Plesu and P. S. Agachi, pp. 649–654.

[56] M. G. Ierapetritou and C. A. Floudas. "Effective continuous-time formulation for short-term scheduling. Part 1. Multipurpose batch processes". In: *Industrial & Engineering Chemistry Research* 37 (1998), pp. 4341–4359.

[57] M. G. Ierapetritou and C. A. Floudas. "Short Term Scheduling: New Mathematical Models vs Algorithmic Improvements". In: *Computers & Chemical Engineering* 22 (1998), S419–S426.

[58] J. Jackson. "An Extension of Johnson's Results on Job Lot Scheduling". In: *Naval Research Logistics Quarterly* 3 (1956), pp. 201–203.

[59] Z. Jia and M. G. Ierapetritou. "Generate Pareto optimal solutions of scheduling problems using normal boundary intersection technique". In: *Computers & Chemical Engineering* 31.4 (Feb. 2007), pp. 268–280.

[60] D. B. Johnson. "Finding all the elementary circuits of a directed graph". In: *SIAM Journal of Computing* 4.1 (Mar. 1975), pp. 77–84.

[61] S. Johnson. "Optimal Two and Three-Stage Production Schedules with Setup Times Included". In: *Naval Research Logistics Quarterly* 1 (1954), pp. 61–67.

[62] I. Karimi, Z. Y. Tan, and S. Bhushan. "An improved formulation for scheduling an automated wet-etch station". In: *Computers & Chemical Engineering* 29.1 (2004), pp. 217–224.

[63] O. A. Kilic, D. P. van Donk, and J. Wijngaard. "A discrete time formulation for batch processes with storage capacity and storage time limitations". In: *Computers & Chemical Engineering* 35 (2011), pp. 622–629.

[64] S. B. Kim, H.-K. Lee, I.-B. Lee, E. S. Lee, and B. Lee. "Scheduling of non-sequential multipurpose batch processes under finite intermediate storage policy". In: *Computers & Chemical Engineering* 24.2-7 (July 2000), pp. 1603–1610.

[65] J. Klemes, F. Friedler, I. Bulatov, and P. Varbanov. *Sustainability in the Process Industry: Integration and Optimization: Integration and Optimization*. Green manufacturing & systems engineering. McGraw-Hill Education, 2010. ISBN: 9780071605557.

[66] E. Kondili, C. Pantelides, and R. Sargent. "A general algorithm for short-term scheduling of batch operations–I. MILP formulation". In: *Computers & Chemical Engineering* 17.2 (Feb. 1993), pp. 211–227.

[67] G. M. Kopanos, J. M. Lainez, and L. Puigjaner. "An Efficient Mixed-Integer Linear Programming Scheduling Framework for Addressing Sequence-Dependent Setup Issues in Batch Plants". In: *Industrial & Engineering Chemistry Research* 48.13 (July 2009), pp. 6346–6357.

[68] G. M. Kopanos, C. A. Mendez, and L. Puigjaner. "Solving Scheduling Problems in a Multi-stage Multi-product Batch Pharmaceutical Industry". In: *Chemical Engineering Transactions* 21 (2010), pp. 511–516.

[69] G. M. Kopanos and L. Puigjaner. "Simultaneous Batching and Scheduling in Multi-product Multi-stage Batch Plants through Mixed-Integer Linear Programming". In: *Chemical Engineering Transactions* 21 (2010), pp. 505–510.

[70] G. M. Kopanos, L. Puigjaner, and C. T. Maravelias. "Production Planning and Scheduling of Parallel Continuous Processes with Product Families". In: *Industrial & Engineering Chemistry Research* (2011).

[71] B. Kovács. "Extension of the S-graph solver for AWS problems (In Hungarian, original title: S-gráf megoldó továbbfejlesztése az automated wet-etch station feladat kezelésére)". Bachelor thesis. Faculty of Information Technology, University of Pannonia, 2014.

[72] J. M. Laínez, M. Hegyháti, F. Friedler, and L. Puigjaner. "Using S-graph to address uncertainty in batch plants". In: *Clean Technologies and Environmental Policy* 12.2 (2010), pp. 105–115.

[73] B. Lee and G. Reklaitis. "Optimal scheduling of cyclic batch processes for heat integration–I. Basic formulation". In: *Computers & Chemical Engineering* 19.8 (Aug. 1995), pp. 883–905.

[74] B. Lee and G. Reklaitis. "Optimal scheduling of cyclic batch processes for heat integration–II. Extended problems". In: *Computers & Chemical Engineering* 19.8 (Aug. 1995), pp. 907–931.

[75] J. Li and C. A. Floudas. "Optimal Event Point Determination for Short-Term Scheduling of Multipurpose Batch Plants via Unit-Specific Event-Based Continuous-Time Approaches". In: *Industrial & Engineering Chemistry Research* 49.16 (Aug. 2010), pp. 7446–7469.

[76] Z. Li and M. Ierapetritou. "Process scheduling under uncertainty: Review and challenges". In: *Computers & Chemical Engineering* 32.4-5 (Apr. 2008), pp. 715–727.

[77] B. Linnhoff and J. R. Flower. "Synthesis of heat exchanger networks: I. Systematic generation of energy optimal networks". In: *AIChE Journal* 24.4 (1978), pp. 633–642.

[78] B. Linnhoff and J. R. Flower. "Synthesis of heat exchanger networks: II. Evolutionary generation of networks with various criteria of optimality". In: *AIChE Journal* 24.4 (1978), pp. 642–654.

[79] H. Liu and J. Wang. "A new way to enumerate cycles in graph". In: 2006, p. 57.

[80] T. Majozi. "Heat integration of multipurpose batch plants using a continuous-time framework". In: *Applied Thermal Engineering* 26.13 (Sept. 2006), pp. 1369–1377.

[81] T. Majozi. "Minimization of energy use in multipurpose batch plants using heat storage: an aspect of cleaner production". In: *Journal of Cleaner Production* 17.10 (July 2009), pp. 945–950.

[82] T. Majozi and F. Friedler. "Maximization of throughput in a multipurpose batch plant under fixed time horizon: S-graph approach". In: *Industrial & Engineering Chemistry Research* 45 (2006), pp. 6713–6720.

[83] T. Majozi and J. F. Gouws. "A mathematical optimisation approach for wastewater minimisation in multipurpose batch plants: Multiple contaminants". In: *Computers & Chemical Engineering* 33.11 (Nov. 2009), pp. 1826–1840.

[84] T. Majozi and X. X. Zhu. "A novel continuous-time MILP formulation for multipurpose batch plants. 1. Short-term scheduling". In: *Industrial & Engineering Chemistry Research* 40(25) (2001), pp. 5935–5949.

[85] C. T. Maravelias and I. E. Grossmann. "A New Continous-Time State Task Network Formulation for Short Term Scheduling of Multipurpose Batch Plants". In: *Computer Aided Chemical Engineering* 14 (2003), pp. 215–220.

[86] C. T. Maravelias. "On the combinatorial structure of discrete-time MIP formulations for chemical production scheduling". In: *Computers & Chemical Engineering* 38 (Mar. 2012), pp. 204–212.

[87] C. T. Maravelias and I. E. Grossmann. "A hybrid MILP/CP decomposition approach for the continuous time scheduling of multipurpose batch plants". In: *Computers & Chemical Engineering* 28.10 (Sept. 2004), pp. 1921–1949.

[88] A. Mascis and D. Pacciarelli. "Job-shop scheduling with blocking and no-wait constraints". In: *European Journal of Operational Research* 143.3 (Dec. 2002), pp. 498–517.

[89] P. Mateti and N. Deo. "On Algorithms for Enumerating all Circuits of Graph". In: *SIAM* 5.1 (Mar. 1976), pp. 90–99.

[90] C. A. Mendez and J. Cerda. "An MILP Continuous-Time Framework for Short-Term Scheduling of Multipurpose Batch Processes Under Different Operation Strategies". In: *Optimization and Engineering* 4 (2003), pp. 7–22.

[91] C. A. Mendez, G. P. Henning, and J. Cerda. "An MILP continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities". In: *Computers & Chemical Engineering* 25.4-6 (May 2001), pp. 701–711.

[92] C. A. Mendez and J. Cerda. "An MILP framework for batch reactive scheduling with limited discrete resources". In: *Computers & Chemical Engineering* 28.6-7 (June 2004), pp. 1059–1068.

[93] C. A. Mendez, J. Cerda, I. E. Grossmann, I. Harjunkoski, and M. Fahl. "State-of-the-art review of optimization methods for short-term scheduling of batch processes". In: *Computers & Chemical Engineering* 30.6-7 (May 2006), pp. 913–946.

[94] K. Nolde and M. Morari. "Electrical load tracking scheduling of a steel plant". In: *Computers & Chemical Engineering* 34.11 (2010), pp. 1899–1903.

[95] J. M. Novas and G. P. Henning. "A comprehensive constraint programming approach for the rolling horizon-based scheduling of automated wet-etch stations". In: *Computers & Chemical Engineering* 42 (2012), pp. 189–205.

[96] Á. Orosz. "Analysis and acceleration of S-graph based throughput maximization algorithms (In Hungarian, original title: S-gráf alapú throughput-maximalizálási algoritmusok vizsgálata és továbbfejlesztése)". Bachelor thesis. Faculty of Information Technology, University of Pannonia, 2012.

[97] O. Ősz. "Novel S-graph based algorithms for the scheduling of automated wet-etch stations (In Hungarian, original title: Új S-gráf alapú algoritmusok kifejlesztése automated wet-etch station ütemezésére)". Bachelor thesis. Faculty of Information Technology, University of Pannonia, 2014.

[98] D. Pacciarelli and M. Pranzo. "Production scheduling in a steelmaking-continuous casting plant". In: *Computers & Chemical Engineering* 28.12 (Nov. 2004), pp. 2823–2835.

[99] S. Panek, S. Engell, S. Subbiah, and O. Stursberg. "Scheduling of multi-product batch plants based upon timed automata models". In: *Computers & Chemical Engineering* 32.1-2 (Jan. 2008), pp. 275–291.

[100] C. C. Pantelides. "Unified frameworks for optimal process planning and scheduling". In: *Proceedings of the second international conference on foundations of computer-aided process operations*. Ed. by R.-p. D. W. T., H. J. C., and D. J. 1993, pp. 253–274.

[101] M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Ed. by -. Third. Prentice Hall, July 2008, 678 pages.

[102] J. M. Pinto and I. E. Grossmann. "A Continuous Time Mixed Integer Linear Programming Model for Short Term Scheduling of Multistage Batch Plants". In: *Ind. Eng. Chem. Res.* 34.9 (Sept. 1995), pp. 3037–3051.

[103] J. M. Pinto and I. E. Grossmann. "A logic-based approach to scheduling problems with resource constraints". In: *Computers & Chemical Engineering* 21.8 (Apr. 1997), pp. 801–818.

[104] E. Pistikopoulos. "Uncertainty in process design and operations". In: *Computers & Chemical Engineering* 19, Supplement 1 (1995), pp. 553–563.

[105] Y. Pochet and F. Warichet. "A tighter continuous time formulation for the cyclic scheduling of a mixed plant". In: *Computers & Chemical Engineering* 32.11 (Nov. 2008), pp. 2723–2744.

[106] G. Robertson, A. Palazoglu, and J. Romagnoli. "A multi-level simulation approach for the crude oil loading/unloading scheduling problem". In: *Computers & Chemical Engineering* 35.5 (2011), pp. 817–827.

[107] B. Roe, L. G. Papageorgiou, and N. Shah. "A hybrid MILP/CLP algorithm for multipurpose batch process scheduling". In: *Computers & Chemical Engineering* 29.6 (May 2005), pp. 1277–1291.

[108] J. Romero, A. Espuna, F. Friedler, and L. Puigjaner. "A New Framework for Batch Process Optimization Using the Flexible Recipe". In: *Industrial and Engineering Chemistry Research* 42(2) (2003), pp. 370–379.

[109] J. Romero, L. Puigjaner, T. Holczinger, and F. Friedler. "Scheduling Intermediate Storage Multipurpose Batch Plants Using the S-Graph". In: *AIChE Journal* 50(2) (2004), pp. 403–417.

[110] A. J. Ruiz-Torres, J. C. Ho, and F. J. López. "Generating Pareto schedules with outsource and internal parallel resources". In: *International Journal of Production Economics* 103.2 (Oct. 2006), pp. 810–825.

[111] K. Sankar and A. Sarad. "A time and memory efficient way to enumerate cycles in a graph". In: *Intelligent and Advanced Systems, 2007. ICIAS 2007.* 25-28 2007, pp. 498–500.

[112] E. Sanmarti, F. Friedler, and L. Puigjaner. "Combinatorial Technique for Short Term cheduling of Multipurpose Batch Plants Based on Schedule-Graph Representation". In: *Computer Aided Chemical Engineering* 22 (1998), S847–850.

[113] E. Sanmarti, T. Holczinger, L. Puigjaner, and F. Friedler. "Combinatorial framework for effective scheduling of multipurpose batch plants". In: *AIChE Journal* 48(11) (2002), pp. 2557–2570.

[114] R. Seid and T. Majozi. "A novel technique for prediction of time points for scheduling of multipurpose batch plants". In: *Chemical Engineering Science* 68.1 (Jan. 2012), pp. 54–71.

[115] R. Seid and T. Majozi. "A robust mathematical formulation for multipurpose batch plants". In: *Chemical Engineering Science* 68.1 (Jan. 2012), pp. 36–53.

[116] N. K. Shah and M. G. Ierapetritou. "Integrated production planning and scheduling optimization of multisite, multiproduct process industry". In: *Computers & Chemical Engineering* 37 (2012), pp. 214–226.

[117] M. A. Shaik and C. A. Floudas. "Novel unified modeling approach for short-term scheduling." In: *Industrial & Engineering Chemistry Research* 48(6) (2009), pp. 2947–2964.

[118] M. A. Shaik, C. A. Floudas, J. Kallrath, and H.-J. Pitz. "Production scheduling of a large-scale industrial continuous plant: Short-term and medium-term scheduling". In: *Computers & Chemical Engineering* 33.3 (Mar. 2009), pp. 670–686.

[119] J. Smidla and I. Heckl. "S-graph based parallel algorithm to the scheduling of multipurpose batch plants". In: *Chemical Engineering Transactions* 21 (2010), pp. 937–942.

[120] M. d. S. Soares, S. Julia, and J. Vrancken. "Real-time scheduling of batch systems using Petri nets and linear logic". In: *Journal of Systems and Software* 81.11 (Nov. 2008), pp. 1983–1996.

[121] F. Sourd and O. Spanjaard. "A multi-objective branch-and-bound framework. Application to the bi-objective spanning tree problem". In: *INFORMS Journal of Computing* 20.3 (2008), pp. 472–484.

[122] S. Subbiah, T. Tometzki, S. Panek, and S. Engell. "Multi-product batch scheduling with intermediate due dates using priced timed automata models". In: *Computers & Chemical Engineering* 33.10 (Oct. 2009), pp. 1661–1676.

[123] A. Sundaramoorthy and I. Karimi. "A simpler better slot-based continuous-time formulation for short-term scheduling in multipurpose batch plants". In: *Chemical Engineering Science* 60.10 (May 2005), pp. 2679–2702.

[124] N. Susarla, J. Li, and I. A. Karimi. "A novel approach to scheduling multipurpose batch plants using unit-slots". In: *AIChE Journal* 56.7 (2010), pp. 1859–1879.

[125] R. Uma, J. Wein, and D. P. Williamson. "On the relationship between combinatorial and LP-based lower bounds for NP-hard scheduling problems". In: *Theoretical Computer Science* 361.2-3 (Sept. 2006), pp. 241–256.

[126] V. T. Voudouris and I. E. Grossmann. "MILP model for scheduling and design of a special class of multipurpose batch plants". In: *Computers & Chemical Engineering* 20.11 (1996), pp. 1335–1360.

[127] L. Wolsey. *Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, 1998. ISBN: 9780471283669.

[128] L. J. Zeballos, P. M. Castro, and C. A. Meendez. "Integrated Constraint Programming Scheduling Approach for Automated Wet-Etch Stations in Semiconductor Manufacturing". In: *Industrial & Engineering Chemistry Research* 50.3 (Feb. 2011), pp. 1705–1715.

[129] C. Zhao, J. Fu, and Q. Xu. "Production-ratio oriented optimization for multi-recipe material handling via simultaneous hoist scheduling and production line arrangement". In: *Computers & Chemical Engineering* 50 (2013), pp. 28–38.

[130] X. Zhao, B. O'neill, J. Roach, and R. Wood. "Heat Integration For Batch Processes: Part 1: Process Scheduling Based on Cascade Analysis". In: *Chemical Engineering Research and Design* 76.6 (Sept. 1998), pp. 685–699.

Index