Modular Scheduling of Tightly Coupled Production Lines

Joan Marcè i Igual[1*], Marc Geilen[1], Mitra Nasri[1], Twan Basten[1]

[1*]Eindhoven University of Technology, The Netherlands.

*Corresponding author(s). E-mail(s): J.Marce.i.Igual@tue.nl;
Contributing authors: m.c.w.geilen@tue.nl; m.nasri@tue.nl;
a.a.basten@tue.nl;

Abstract

Optimising productivity of tightly coupled production lines in, for instance, the production printing or semiconductor industry is difficult due to the diversity of products resulting in different product flows, the variety of constraints, and the precise timing required to coordinate multiple tightly coupled machines. A modular setup provides flexibility and cost reduction through reuse of machinery and schedulers. However, the scheduling of a modular production line is challenging as it leads to a distributed decision process where each module has a limited view of the entire system, while local scheduling decisions have a global impact on schedule feasibility. This work proposes a distributed scheduling method for tightly coupled modular sequential production lines where products cannot overtake each other during production. We develop a multi-agent framework in which a system agent propagates timing constraints of the schedulers of different modules. The system agent aims to reach consensus about the handover times of jobs between modules to converge to a globally feasible schedule. The system agent interacts with local agents that interface with local schedulers, allowing the use of existing local schedulers without modification. We illustrate the approach on production lines with multiple re-entrant flow-shop instances with setup times and due dates, which results in a challenging scheduling problem. The performance of the proposed distributed scheduling method is assessed using a monolithic exact scheduler (implemented as a constraint program, not applicable to modular problems) as a reference. It is found to deliver good quality schedules, having only a 1.15 larger makespan on average than the hypothetical optimum provided by the exact scheduler.

Keywords: Scheduling, Distributed decision making, Flow shops, Modular production lines, Consensus, Constraint graphs, Collaborative manufacturing

1 Introduction

1.1 Opportunity

High quality, productive production lines, such as the ones found in the printing or semiconductor industry, require complex processes and advanced machinery (Pan, Zhou, Qiao, & Wu, 2018; Traganos et al., 2020). Due to diversity in products and different flows, the scheduling problem is very challenging (J. Zhang & Wang, 2016). Given the complexity of these production lines, their integration capabilities can be improved with a modular setup. Modularity allows to flexibly build the desired production line by combining the right modules, possibly of different manufacturers. Reusing modules and schedulers can also reduce costs and design effort because it does not require a specific monolithic design for a production line. Furthermore, each manufacturer can focus on optimising their modules, including their local schedulers.

Production lines may be built from loosely coupled or tightly coupled machines. In loosely coupled production lines, the product flow between machines is decoupled through buffers. In *tightly coupled* production lines, multiple machines are integrated with limited or no buffering in between them. This reduces the overall cost of the manufacturing equipment and potentially increases productivity. However, productive scheduling of tightly-coupled production lines is more difficult than scheduling loosely coupled production lines. In tightly coupled lines, the schedule of each module must account for product handover times and constraints of neighbouring modules. Local scheduling decisions have a *global impact* on schedule feasibility and productivity.

Realizing *modularity and reusability of machines* for tightly coupled production lines is challenging. Since modules must be able to operate in different production lines and production-line configurations, scheduling must follow a distributed decision-making process built upon the local schedulers of the individual modules (J. Zhang & Wang, 2016). Distributed scheduling is an essential enabler for flexible, customizable, multi-vendor setups, with reusable manufacturing equipment. The key challenge in distributed scheduling for tightly coupled production lines built from reusable machines is to find productive globally feasible schedules, coordinating precise handovers between machines, without having a global view of and control over the local scheduling of the individual modules. Multi-agent technology provides a potential solution to address this challenge of distribution in tightly coupled production lines.

Figure 1 shows a tightly coupled production line consisting of multiple modules, each with its own local scheduler and consisting in turn of one or more machines. Such a production line requires a method to synchronise the modules. In this work, we propose a distributed scheduling method that allows modules to find a consensus on their local schedules that leads to a feasible global schedule.

1.2 Contributions

Our work is the first to provide a modular scheduling approach for tightly coupled modular production lines with modules, possibly from different vendors, where each module has it own local scheduler. It applies to non-cyclic sequential module patterns processing jobs with a fixed output order. Our approach supports the integration

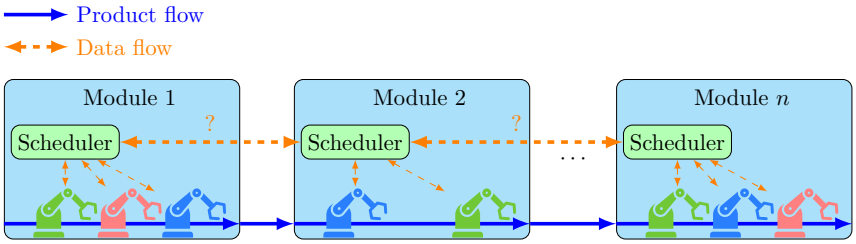

Figure 1: Production line composed of different modules.

and reuse of the modules and their local schedulers with minimal or no buffering in between modules, under minimal assumptions. Our method (explained in Section 5) provides an abstraction of a module's output behaviour, in the form of timing bounds, the translation and propagation mechanism of constraints to other modules, as well as the necessary steps to converge to a globally feasible schedule. We implement this mechanism as a cooperative communicating multi-agent system (MAS) using a hierarchical architecture (Glavic, 2006) with local agents (LAs) for the modules and a single system agent (SA). Each LA interacts with a single module and its scheduler while the SA forwards and translates module constraints between local agents. Our design aims to minimise the assumptions on the behaviour of local schedulers, so that it can be flexibly integrated with existing local schedulers. By following an iterative approach and a translation of constraints, our method guarantees that if a consensus between modules is found, then the local schedules together form a global schedule that is feasible.

The approach is illustrated on a production line with multiple 2-re-entrant machines, where products are processed two times in the same machine. Re-entrant production lines can be found in semiconductor lithography or production printing, for instance (J. Zhang & Wang, 2016). We model the machines as re-entrant flow shops with setup times, due dates, no overtaking between machines, and fixed output order of jobs. The setup times and due dates allow modelling of travelling times and limited buffering by expressing lower and upper bounds between operations, respectively. This makes the approach suitable for tightly coupled modules. The scheduling goal is makespan minimization. The essential (local) scheduling decisions are made by choosing the order of operations of each re-entrant machine. We integrate our MAS with two different existing local schedulers for 2-re-entrant machines (Jeong & Kim, 2014; van Pinxten, Waqas, Geilen, Basten, & Somers, 2017) for experimental analysis, showing feasibility of reusing existing schedulers, also in combination within a single production line.

We evaluate the quality of our distributed scheduling solution using as a reference a monolithic (all-knowing) scheduler implemented as a constraint program in a constraint solver. We observe a good trade-off between schedule quality and scheduling speed by generating schedules for large systems with 10 modules and 300 jobs at an average of 1.5 seconds per job with a makespan only 1.15 times larger than the makespan produced by the monolithic scheduler (that cannot be used in modular setups).

The paper is structured as follows, Section 2 gives an overview of the related work. Section 3 defines the flow-shop problem and our system model. Section 4 explains how modules are composed into a production line. Section 5 presents our multi-agent solution to the distributed scheduling problem. Section 6 explains the reasoning behind the design choices. Section 7 has the experimental evaluation. Section 8 concludes.

2 Related work

Our work focuses on tightly coupled production lines, as a combination of multiple re-entrant flow-shops and their schedulers, with no or limited (work-in-progress) buffering in-between modules. We start discussing approaches to distributed manufacturing and satisfying handover constraints. Then, we give details on existing approaches for re-entrant flow-shop scheduling with added constraints, including work-in-progress buffering constraints. Finally, we look at MAS scheduling as a way to handle different schedulers (or agents) in a production line. Thus, motivating our decision to use a coordinating MAS for this problem.

2.1 Distributed systems

Distributed manufacturing involves producing different parts of a product in different factories or on different machines, usually distributed geographically or physically. An important advantage of distribution is to create multiple possible product flows, allowing more customisation and personalisation (Sadiku, Ajayi-Majebi, & Adebo, 2023). Typically, manufacturing of products has been modelled with variants of the job-shop or flow-shop scheduling problems (JSSP, FSSP) (Emmons & Vairaktarakis, 2013). For distributed manufacturing, there are two categories of problems: the distributed job-shop/flow-shop scheduling problem (DJSP, DFSP) (Bagheri Rad & Behnamian, 2022) and the job-shop scheduling problem with transportation resources (JSPT) (Nouri, Driss, & Ghédira, 2016).

In the basic form of the DJSP and DFSP, multiple factories are available to process a job. But the entire job must be processed in a single factory. Thus, the scheduling problem consists of choosing the factory where a job will be made and the ordering of the jobs inside the factory (Jia, Fuh, Nee, & Zhang, 2002; Naderi & Ruiz, 2010). Although the basic DJSP/DFSP consider distribution across factories, the techniques also apply to manufacturing systems where jobs are distributed over multiple machines in a single factory, as long as jobs should be allocated to a single machine in such a factory. These techniques are therefore not applicable to modular setups where jobs need to be scheduled on multiple machines.

Work on transportation between factories in DJSP/DFSP is rare given the assumption that jobs are processed entirely in a single factory (Lohmer & Lasch, 2021). Chan, Prakash, Ma, and Wong (2012) considered transportation using a centralised agent that uses Tabu search and simulated annealing to generate a schedule. More recently, Ziaee (2017) proposed a centralised heuristic approach to solve the DJSP with transport times as well as transfer of work in progress (WIP) between factories. These approaches address *loosely* coupled production systems. Moreover, given that

the approaches are *centralized*, they do not apply to modular setups as they assume full information about and control over the entire production process.

The JSPT defines a job-shop problem with the transportation resources required to transport elements (robots, vehicles...) from one machine to another and their availability. There are both heuristics and exact approaches for JSPT (Nouri et al., 2016). However, these approaches cannot be applied to modular production lines due to their central nature that assumes full information and control.

The systems considered in this work, tightly coupled modular production lines, are another form of distributed manufacturing, focused on a single manufacturing facility. These production lines have multiple modules, each with their own local schedules of operations to be performed, requiring coordination of work-in-progress handover times. A common assumption in DFSP and DJSP is the availability of unlimited (work-in-progress) buffering between modules (Chan et al., 2012; Ziaee, 2017). However, in tightly coupled production lines with only limited buffering between modules, the handover times between modules are critical. Handovers between machines are bounded in duration from below by transportation time and from above by limited buffering. In *tightly coupled* setups, the modules need to achieve *consensus* about the handovers. For *modular* setups, this consensus needs to be achieved in a *distributed* manner, because it cannot be assumed that the production-line scheduler has control over scheduling decisions of individual machines.

2.2 Flow-shop problems

Inspired by the FSSPs in wafer fabrication, TFT-LCD manufacturing, or production-printing problems (Rifai, Nguyen, & Dawal, 2015; Waqas et al., 2015), in this paper, we focus on production lines with fixed job completion order (Lin & Hwang, 2011), precedence constraints between operations (Balas, Simonetti, & Vazacopoulos, 2008), and without overtaking in-between machines (Gupta & Stafford, 2006). We illustrate our framework in a production line with re-entrant flow shops (Riezebos, Gaalman, & Gupta, 1995), where a job is processed multiple times by a machine. Because overtaking is not allowed and the output order is fixed, without re-entrance, the scheduling problem is trivial as there are no decisions to make.

Online schedulers take decisions in little time to allow for a continuous production. There exists an online scheduler with two different variations for 2-re-entrant flow-shop problems: the bounded heuristic constraint-based scheduler (BHCS) and the multi-dimensional bounded heuristic constraint-based scheduler (MD-BHCS) (van Pinxten et al., 2017). They are based on the heuristic constraint-based scheduler (HCS) (Waqas et al., 2015), that uses a constraint-graph model to check for feasibility of candidate schedules and to determine earliest start times of operations. These heuristics can also solve variable re-entrant problems, where jobs can have a different number of re-entrances in each machine (van der Tempel, van Pinxten, Geilen, & Waqas, 2018).

Offline scheduling solutions for the same problem are the modified FL (MFL) or modified NEH (MNEH) heuristics (Jeong & Kim, 2014), which use a branch-and-bound algorithm to schedule 2-re-entrant flow-shops. They cannot be used for online scheduling, as the execution time increases significantly with the number of jobs.

All these solutions are designed to work in a monolithic production line with a single re-entrant machine and a single scheduler. This work aims to schedule a production line composed of multiple re-entrant flow shops by coordinating existing schedulers for re-entrant flow shops. Thus, our work allows the scheduling of multiple re-entrant machines in a system of systems.

We assume that our production lines are tightly coupled. This means that there is limited or no buffering (also known as limited WIP) between the modules of the production line. There are many existing solutions that consider this type of constraints. The previously mentioned BHCS and MD-BHCS can handle them due to their constraint-graph-based modelling and analysis. S.Q. Liu, Kozan, Masoud, Zhang, and Chan (2018) designed two monolithic heuristics that can also handle the limited or no buffering scenarios. C. Zhang, Shi, Huang, Wu, and Shi (2017) present a heuristic for a two-stage production line combining batch processing and processing of individual jobs, with limited buffering between the two stages. Qin, Wang, Shi, Liu, and Shi (2021) designed a genetic algorithm for hybrid flow shops with limited waiting time. However, all these solutions focus on monolithic central approaches. Our work is the first to consider modular scheduling of tightly coupled systems.

2.3 Multi-agent approaches

A MAS is a collection of computerised agents. Each agent can implement different behaviours to solve coordination problems. For manufacturing, MAS can model different machines or modules and abstract part of their behaviour in a common framework. Seitz, Gehlhoff, Cruz Salazar, Fay, and Vogel-Heuser (2021) propose such a framework, where different manufacturing facilities are modelled as agents and place a bid to decide who produces a product. Their approach is suitable for reusing local schedulers, but not for tightly-coupled production lines, as the agents communicate with each other to allocate production of products based on the local module status but they do not consider handover constraints.

Different MAS architectures are available for scheduling (Glavic, 2006). Common approaches use one agent per operation or per constraint and resolve conflicts either through a bidding process or with a central agent until none are remaining (Fazlirad & Brennan, 2018; Maturana & Norrie, 1996). This approach is also applied to solve distributed constraint optimisation problems (Fioretto, Pontelli, & Yeoh, 2018; J. Liu & Sycara, 1993), like our problem. However, since we want to reuse existing schedulers, we cannot use approaches that actively decide the start time of the operations in different modules and we look at heterarchical MAS architectures (Toptal & Sabuncuoglu, 2010). Our architecture is a cooperative MAS with multiple LAs interacting with the local schedulers and a single SA propagating and translating their messages. Since this SA is not scheduling, we consider it a heterarchical architecture from the scheduling point of view and a hierarchical architecture from the organisation point of view.

Heterarchical MAS scheduling approaches usually use iterative refinement (Toptal & Sabuncuoglu, 2010). Kouider and Bouzouia (2011) propose a hierarchical architecture (heterarchical from the scheduling point of view) where the SA decomposes the problem into smaller problems, one per machine, and each machine has a LA. The LAs exchange completion times of operations and as soon as the dependencies of an

operation are satisfied, the LA schedules it. In contrast, our heterarchical architecture also employs iterative refinement but assumes that the decomposition is derived from the linking of multiple modules, each possibly composed of multiple machines and each with its own local scheduler. Another iterative approach is proposed by J. Liu and Sycara (1993). They create a heterarchical MAS for constraint satisfaction in job-shop scheduling with due dates. Their architecture has order agents, managing jobs, and resource agents, managing machines. Order agents only talk to resource agents and vice versa until there are no more conflicts. The information exchanged between agents is, primarily, intervals on the start times of the operations. This approach is similar to ours as we share bounds on the relative start times of consecutive input and output operations. A work that explicitly considers consensus in the handover times between machines is the one from Miyamoto, Umeda, and Takai (2020). They propose a consensus mechanism for the JSSP where they minimise the discrepancy in the start times of the handover operations, reaching consensus when the error reduces to 0 on all handovers. Their solution is a heterarchical architecture where the local agents have full control of the start times of the operations.

Our approach differs from Kouider and Bouzouia (2011), J. Liu and Sycara (1993) and Miyamoto et al. (2020) by enabling modularity and reuse of local schedulers. The three discussed methods rely on control of the local scheduler. In contrast, our approach focuses on reaching consensus on handovers only, leaving the scheduling of operations within modules to local schedulers. The mentioned approaches do not support modularity, as their agents rely on specifying the start times of all the operations.

As an alternative for heterarchical MAS scheduling approaches, J. Zhang and Wang (2016) propose a *hierarchical* MAS architecture with different agents for different types of machines and jobs. Their approach divides the architecture in a system layer and machine layer. The system layer models the capacity of each machine and generates a plan for the number of products processed in a period of time. The machine layer receives the number of products that must be produced in a period and generates a schedule with the exact start times required for the selected period. The design handles dynamic events in the machine layer but it is not suitable for tightly coupled production lines where the timing of one module greatly impacts the scheduling possibilities of other modules.

In summary, our method for scheduling jobs on production lines is the first that supports modularity in the presence of tight coupling between modules. This allows integration of modules, possibly from different original equipment manufacturers (OEMs), while reusing their local schedulers.

3 Flow shops with output order constraints

In this section, we define the flow-shop scheduling problem that we assume as an appropriate abstraction of the scheduling problem of every module and give the definition of a feasible schedule. We further explain the constraint-graph model used to check for feasibility and we introduce an illustrative example.

3.1 Flow-shop model

A flow shop with setup times, relative due dates, and a fixed output order is a tuple

$$\tau = (J, M, \mathsf{ops}, \mathsf{mch}, \mathsf{prc}, \mathsf{si}, \mathsf{sd}, \mathsf{dd})$$

$J = \{j_1, \ldots, j_{|J|}\}$ is the set of jobs to be processed. For every job j_u, the index u in $\mathbb{N}$ with $1 \leq u \leq |J|$ denotes its place in the output order. M is the set of machines on which the jobs are processed. $\mathsf{ops}(j_u) = \{o_{u,1}, \ldots, o_{u,|\mathsf{ops}(j_u)|}\}$ is the set of operations performed by a job j_u in J; these sets are disjoint for different jobs. For every operation $o_{u,k}$, the index k in $\mathbb{N}$ with $1 \leq k \leq |\mathsf{ops}(j_u)|$ determines its place in the execution order of the operations of job j_u.

The set O of all operations is defined as $O = \bigcup_{j \in J} \mathsf{ops}(j)$. The function $\mathsf{mch} : O \to M$ defines which machine performs each operation. We define the auxiliary functions $\mathsf{fst}(j_u) = o_{u,1}$ and $\mathsf{lst}(j_u) = o_{u,|\mathsf{ops}(j_u)|}$ to refer to the first and last operations of a job j_u, respectively. These functions help defining restrictions on the flow shop. Depending on the restrictions, different flavours of flow shop can be obtained. If a job can visit a machine multiple times, it is called a re-entrant flow shop and if it can only visit one machine at most two times, it is a 2-re-entrant flow shop.

The processing time of an operation on a machine is defined as $\mathsf{prc} : O \to \mathbb{R}_{>0}$. We define setup times as the delay between the completion of an operation and the start of another one. In practice, setup times often model product travel times or machine or product preparation times. We distinguish two types of setup times: sequence-independent and sequence-dependent setup times. Sequence-independent setup times $\mathsf{si} : O \times O \hookrightarrow \mathbb{R}_{\geq 0}$ are always-enforced constraints such as travel times between machines. If $\mathsf{si}(o_1, o_2)$ is undefined, no restriction exists between the completion of o_1 and the start of o_2. Sequence-dependent setup times $\mathsf{sd} : O \times O \hookrightarrow \mathbb{R}_{\geq 0}$ model preparation times, such as the time needed for an arm to reach a given position. If the arm is already at the desired position, then the preparation time is shorter. Thus, sd depends on the sequence of operations that a machine performs. The $\mathsf{sd}(o_1, o_2)$ constraint only applies if o_1 is scheduled immediately before o_2 on the same machine. We assume that it is defined between all operations mapped onto the same machine. The last element of the tuple is the relative deadline, or due date, $\mathsf{dd} : O \times O \hookrightarrow \mathbb{R}_{\geq 0}$, that defines the maximum time delay between the completion time of one operation and the start time of another one. If $\mathsf{dd}(o_1, o_2)$ is undefined, there is no due date between o_1 and o_2.

3.2 Feasible schedules

A schedule $\Omega : O \to \mathbb{R}_{\geq 0}$ describes the start times for all the operations in O and is a potential solution of the flow-shop problem. It is called **feasible** iff the following constraints are met:

C1) Every job must perform its operations in the order specified by the operations' indices:

$$\forall u, k \in \mathbb{N} \wedge 1 \leq u \leq |J| \wedge 1 \leq k < |\mathsf{ops}(j_u)| : \\ \Omega(o_{u,k}) + \mathsf{prc}(o_{u,k}) \leq \Omega(o_{u,k+1})$$

C2) Every machine can only perform one operation at a time:

$$\begin{aligned}&\forall o_1, o_2 \in O : \mathsf{mch}(o_1) = \mathsf{mch}(o_2) \wedge o_1 \neq o_2 \\ &\quad \Rightarrow \Omega(o_1) + \mathsf{prc}(o_1) \leq \Omega(o_2) \vee \Omega(o_2) + \mathsf{prc}(o_2) \leq \Omega(o_1)\end{aligned}$$

C3) If two operations are scheduled on the same machine and no other operation starts between the start of o_1 and the start of o_2, then the sequence-dependent setup times must be met:

$$\begin{aligned}\forall o_1, o_2 \in O \ : \Big(&\mathsf{mch}(o_1) = \mathsf{mch}(o_2) \wedge \Omega(o_1) < \Omega(o_2) \wedge \\ &\big(\nexists o \in O : \mathsf{mch}(o) = \mathsf{mch}(o_1) \wedge \Omega(o_1) < \Omega(o) < \Omega(o_2)\big)\Big) \\ &\Rightarrow \Omega(o_1) + \mathsf{prc}(o_1) + \mathsf{sd}(o_1, o_2) \leq \Omega(o_2)\end{aligned}$$

C4) If the sequence-independent setup time between two operations is defined, it must be respected[1]:

$$\forall (o_1, o_2) \in \mathrm{dom}(\mathsf{si}) \Rightarrow \Omega(o_1) + \mathsf{prc}(o_1) + \mathsf{si}(o_1, o_2) \leq \Omega(o_2)$$

C5) If a due date is defined between two operations o_1 and o_2, then o_2 cannot start later than allowed by the relative due date:

$$\forall (o_1, o_2) \in \mathrm{dom}(\mathsf{dd}) \Rightarrow \Omega(o_1) + \mathsf{prc}(o_1) + \mathsf{dd}(o_1, o_2) \geq \Omega(o_2)$$

C6) Jobs must complete in the order specified by their indices:

$$\forall u \in \mathbb{N} \wedge 1 \leq u < |J| : \ \Omega(\mathsf{lst}(j_u)) \leq \Omega(\mathsf{lst}(j_{u+1}))$$

C7) A job cannot overtake other jobs in-between machines:

$$\begin{aligned}&\forall u, v, k, l \in \mathbb{N} \wedge 1 \leq u, v \leq |J| \wedge u \neq v \wedge 1 \leq k < |\mathsf{ops}(j_u)| \wedge 1 \leq l < |\mathsf{ops}(j_v)| : \\ &\big(\mathsf{mch}(o_{u,k}) = \mathsf{mch}(o_{v,l}) \wedge \mathsf{mch}(o_{u,k+1}) = \mathsf{mch}(o_{v,l+1}) \\ &\ \wedge \Omega(o_{u,k}) < \Omega(o_{v,l})\big) \Rightarrow \Omega(o_{u,k+1}) \leq \Omega(o_{v,l+1})\end{aligned}$$

3.3 Constraint-graph model and feasibility check

We use, without loss of generality, the constraint-graph model (Waqas et al., 2015) to represent the constraints of a production line for a given order of operations on each of the machines, to test the feasibility of a schedule and to obtain the start times of its operations. We define sequences of operations on the machines $\mathsf{seq} : M \to O^+$, where $\mathsf{seq}(m)(i)$ denotes the i-th operation that a machine m must perform. We assume that machine sequences seq specify sequences for all machines in the flow shop, ordering all operations allocated to each of these machines. A constraint-graph model $\mathcal{G}_{\mathsf{seq}} = (V, E)$ is a labelled directed graph built from the problem τ and the machine sequences seq.

[1] We use the notation $\mathrm{dom}(f)$ to denote the domain of function f.

In the constraint graph $\mathcal{G}_{\mathsf{seq}}$, all operations are represented as vertices ($V = O$). An edge $(o_1, b, o_2) \in E$ labelled with a value b represents the constraint $\Omega(o_1) + b \leq \Omega(o_2)$ between o_1 and o_2. The graph is built by representing every constraint defined in Section 3.2 with a set of labelled edges. After the machine sequences are given, all constraints are fixed. In particular, the sequence-dependent setup-time constraints can be determined. Constraints are mapped as follows:

- Constraint C1: job operation order. For every pair $o_{u,k}$ and $o_{u,k+1}$ of operations in $\mathsf{ops}(j_u)$ of a job j_u in J, we add an edge with the value $\mathsf{prc}(o_{u,k})$ from $o_{u,k}$ to $o_{u,k+1}$.
- Constraints C2 and C3: one operation per machine at a time and sequence-dependent setup times. For each machine m in M and pair of consecutively scheduled operations $o_1 = \mathsf{seq}(m)(i)$ and $o_2 = \mathsf{seq}(m)(i+1)$, we add an edge with the value $\mathsf{prc}(o_1) + \mathsf{sd}(o_1, o_2)$ from o_1 to o_2. This edge represents both constraints C2 and C3.
- Constraint C4: sequence-independent setup times. For every pair (o_1, o_2) of operations in the domain of si, we add an edge with the value $\mathsf{prc}(o_1) + \mathsf{si}(o_1, o_2)$ from o_1 to o_2.
- Constraint C5: due date between operations. For every pair (o_1, o_2) of operations in the domain of dd, we add an edge with the value $-\mathsf{dd}(o_1, o_2) - \mathsf{prc}(o_1)$ from o_2 to o_1. The edge is negative and goes in the opposite direction because constraint C5 is in the form $\Omega(o_1) + \mathsf{prc}(o_1) + \mathsf{dd}(o_1, o_2) \geq \Omega(o_2)$ while the graph only represents constraints of the form $\Omega(o_x) + b \leq \Omega(o_y)$. Thus, here $o_x = o_2$, $o_y = o_1$ and $b = -\mathsf{prc}(o_1) - \mathsf{dd}(o_1, o_2)$.
- Constraint C6: job output order. For every pair of consecutive jobs j_u and j_{u+1} in J, we add an edge from $\mathsf{lst}(j_u)$ to $\mathsf{lst}(j_{u+1})$ with value 0.
- Constraint C7: no overtaking in-between machines. For each machine m in M and pair of consecutive operations $o_{u,k} = \mathsf{seq}(m)(i)$ and $o_{v,l} = \mathsf{seq}(m)(i+1)$, if they are not the last operations and their next operations are performed on the same machine, i.e., $\mathsf{mch}(o_{u,k+1}) = \mathsf{mch}(o_{v,l+1})$, we add an edge from $o_{u,k+1}$ to $o_{v,l+1}$ with the value 0.

When multiple constraints exist between two operations, these can be captured with a single edge; the label of the edge is determined by the largest value among those constraints.

Since the labelled edges of the graph represent timing constraints between operations, relative timing constraints between operations are captured by the length of the longest path that goes from one operation to the other. Thus, the feasible schedule Ω with the earliest start time of all operations is found by taking the lengths of the longest path between the first operation in the system and each of the other operations. Moreover, these longest paths, and hence a feasible schedule, only exist iff no positive cycles exist in the graph. Any such a positive cycle represents a chain of constraints that cannot be satisfied (van Pinxten et al., 2017).

Example 3.1 An example of a re-entrant flow-shop problem is a production printer. It has three machines, $M = \{m_1, m_2, m_3\}$. m_1 is the printer input machine taking in sheets of paper, m_2 is the image transfer machine, that transfers ink onto the paper, and m_3 is the

$o_{u,j}$	prc($o_{u,j}$)	mch($o_{u,j}$)
$o_{u,1}$	20	m_1
$o_{u,2}$	30	m_2
$o_{u,3}$	30	m_2
$o_{u,4}$	20	m_3

(a) Values for prc and mch

Between		si	dd
$o_{u,1}$	$o_{u,2}$	0	0
$o_{u,2}$	$o_{u,3}$	70	270
$o_{u,3}$	$o_{u,4}$	0	0

(b) Values for si and dd

Table 1: Values for Example 3.1

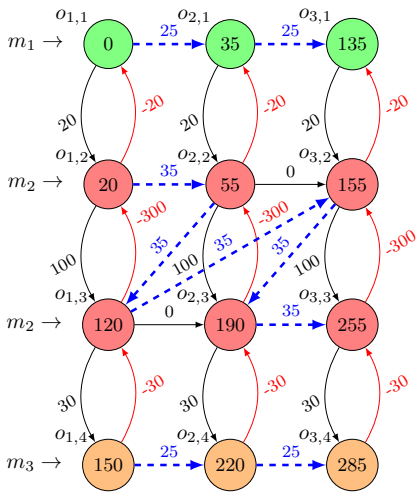

Figure 2: Representation of the sequences of Example 3.1 (blue dashed lines) in a constraint graph. The values inside the nodes are the start times of the earliest feasible schedule Ω.

printer output machine, releasing the printed sheets. We assume that there are three sheets to be printed, represented by jobs $J = \{j_1, j_2, j_3\}$. Every job has four operations ($\mathsf{ops}(j_u) = \{o_{u,1}, \ldots, o_{u,4}\}$). The first operation is the input operation, then there is one image transfer operation for each side of the paper and finally there is the output operation. The mapping mch of operations as well as the prc, si and dd values can be seen in Table 1. If both o_1 and o_2 are performed on the same machine ($\mathsf{mch}(o_1) = \mathsf{mch}(o_2)$), then $\mathsf{sd}(o_1, o_2) = 5$.

Figure 2 shows a constraint-graph representation for the given flow shop for the following operation sequences for the three machines: $\mathsf{seq}(m_1) = \langle o_{1,1}, o_{2,1}, o_{3,1}\rangle$, $\mathsf{seq}(m_2) = \langle o_{1,2}, o_{2,2}, o_{1,3}, o_{3,2}, o_{2,3}, o_{3,3}\rangle$, and $\mathsf{seq}(m_3) = \langle o_{1,4}, o_{2,4}, o_{3,4}\rangle$. As the operations $o_{u,2}$ and $o_{u,3}$ of every job $j_u \in J$ are performed on the same machine, we call these *re-entrant operations*. With the constraint graph, the Bellman-Ford longest path algorithm can be used to obtain the schedule with earliest starting times by determining the lengths of the longest paths between $o_{1,1}$ and each of the other operations. The length of this path is the value in the center of each node in Figure 2. The longest path from $o_{1,1}$ to $o_{2,1}$, for instance, has length 35 and goes via $o_{1,2}$ and $o_{2,2}$. The schedule with the earliest start times is also the schedule with the lowest makespan for the given sequences.

4 Production-line problem

In this section, we explain how we compose multiple flow shops defined in Section 3 into a production line, like the one shown in Figure 1, as a *production-line problem (PLP)*. Then, we define the feasibility conditions for its schedule. In the following sections, we explain how our MAS generates a schedule for a production line.

The way we define the composition allows also to model the whole production line as a single instance of the flow-shop problem defined in Section 3.1. This is useful

later in Section 7 to compare against a monolithic scheduler. We consider non-cyclic sequential module patterns with a fixed output order of jobs. These patterns can be found for instance in production printing or the semiconductor industry.

4.1 Composition

A production line $P = (N, T)$ is composed of a set N of modules and a set T of transfer points between modules. These transfer points model the transfer of products from one module to the next one. Every module $n_i \in N$ comes with an instance of the flow-shop problem $\tau_i = (J_i, M_i, \mathsf{ops}_i, \mathsf{mch}_i, \mathsf{prc}_i, \mathsf{si}_i, \mathsf{sd}_i, \mathsf{dd}_i)$ defined in Section 3.1. The index $i \in \mathbb{N}, 1 \leq i \leq |N|$ denotes the order of the modules. Because the jobs flow from one module to the next and all jobs are processed by all modules, we assume that the set of jobs is the same for all modules, that is, $J_i = J$ for all i, and that the sets of machines M_i are disjoint. Every transfer point $T_i = (\mathsf{ts}_i, \mathsf{td}_i)$ in T, with $1 \leq i < |N|$, represents a connection between modules where jobs flow from module n_i to module n_{i+1}. The setup-time function $\mathsf{ts}_i : J \to \mathbb{R}_{\geq 0}$ defines the minimum transfer time of a job between leaving module n_i and arriving at module n_{i+1}. Likewise, the due date $\mathsf{td}_i : J \hookrightarrow \mathbb{R}_{\geq 0}$ defines the maximum allowed transfer time of a job. If no due date is defined, then there is no maximum allowed transfer time for a job (e.g., in case of unlimited buffering where a job can stay in a buffer as much as needed).

4.2 Feasible schedules

A schedule Ω for a production line is a function that maps all operations of all modules to their start times, i.e., $\Omega : O \to \mathbb{R}_{\geq 0}$, with $O = \bigcup_{n_i \in N} O_i$. It can be composed from the local schedules of the modules $\Omega_i : O_i \to \mathbb{R}_{\geq 0}$. It is called feasible if all the local schedules Ω_i are feasible and the following constraints are respected for the transfer points:
CT1) Minimum transport times must be respected:

$$\forall T_i \in T, j \in J: \ \Omega_i(\mathsf{lst}_i(j)) + \mathsf{prc}_i(\mathsf{lst}_i(j)) + \mathsf{ts}_i(j) \leq \Omega_{i+1}(\mathsf{fst}_{i+1}(j))$$

CT2) Due dates must be respected, if defined:

$$\forall T_i \in T, j \in \mathrm{dom}(\mathsf{td}_i) : \Omega_i(\mathsf{lst}_i(j)) + \mathsf{prc}_i(\mathsf{lst}_i(j)) + \mathsf{td}_i(j) \geq \Omega_{i+1}(\mathsf{fst}_{i+1}(j))$$

4.3 Global constraint-graph model

The constraints of a full production-line problem and the chosen sequences seq of all modules can be represented as a global constraint graph $\mathcal{G}^{global}_{\mathsf{seq}} = (V, E)$, similar to the way flow-shop problems are represented in Section 3.3. All operations of all modules are represented as vertices ($V = O = \bigcup_{n_i \in N} O_i$). The edges between operations are defined in the same way as in Section 3.3. Additionally, we add edges to represent constraints CT1 and CT2:

- Constraint CT1: transfer setup times. For every T_i in T and job j in J, we add an edge from $\mathsf{lst}_i(j)$ to $\mathsf{fst}_{i+1}(j)$ with the value $\mathsf{prc}_i(\mathsf{lst}_i(j)) + \mathsf{ts}_i(j)$.

$o_{u,k}$	prc_i	mch_1	mch_2
$o_{u,1}$	20	m_1	m_4
$o_{u,2}$	30	m_2	m_5
$o_{u,3}$	30	m_2	m_5
$o_{u,4}$	20	m_3	m_6

(a) Values for prc_i, mch_1 and mch_2

Between		si_i	dd_i
$o_{u,1}$	$o_{u,2}$	0	0
$o_{u,2}$	$o_{u,3}$	70	270
$o_{u,3}$	$o_{u,4}$	0	0

(b) Values for si_i and dd_i

Table 2: Values for Example 4.1.

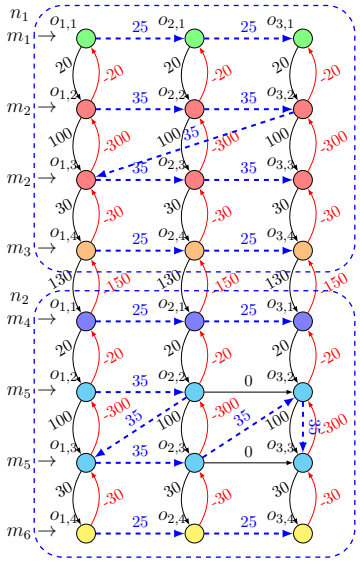

Figure 3: Constraint graph of Example 4.1. Columns and rows denote jobs and operations. Machines are indicated on the left, modules are represented by ▪▪▪ rectangles, and sequences seq_i by ▪▶ arrows.

- Constraint CT2: transfer due dates. For every T_i in T and job j in the domain of td_i, we add an edge from $\mathsf{fst}_{i+1}(j)$ to $\mathsf{lst}_i(j)$ with the value $-\mathsf{prc}_i(\mathsf{lst}_i(j)) - \mathsf{td}_i(j)$.

Example 4.1 We define a production line with two modules n_1 and n_2 each with three machines, $M_1 = \{m_1, m_2, m_3\}$ and $M_2 = \{m_4, m_5, m_6\}$. Both modules are instances of the flow shop of Example 3.1. We schedule three jobs $J = \{j_1, j_2, j_3\}$, with four operations per job per module, $\mathsf{ops}_1(j_u) = \{o_{u,1}, \ldots, o_{u,4}\}$, $\mathsf{ops}_2(j_u) = \{o_{u,1}, \ldots, o_{u,4}\}$ (for $u \in \{1, 2, 3\}$). The mapping mch of operations as well as the values for processing times prc_i, sequence-independent setup times si_i, and due dates dd_i (for $i \in \{1, 2\}$) can be found in Table 2. If both o_1 and o_2 are performed on the same machine, then $\mathsf{sd}_i(o_1, o_2) = 5$ (for $i \in \{1, 2\}$). All these values align with the values in Example 3.1. There is a transfer point $T_1 = (\mathsf{ts}_1, \mathsf{td}_1)$. For every job j_u, the transfer setup time is $\mathsf{ts}_1(j_u) = 110$ and the transfer due date is $\mathsf{td}_1(j_u) = 130$. Figure 3 shows a constraint graph for the production line with the given sequences $\mathsf{seq}_1(m_1) = \langle o_{1,1}, o_{2,1}, o_{3,1}\rangle$, $\mathsf{seq}_1(m_2) = \langle o_{1,2}, o_{2,2}, o_{3,2}, o_{1,3}, o_{2,3}, o_{3,3}\rangle$, $\mathsf{seq}_1(m_3) = \langle o_{1,4}, o_{2,4}, o_{3,4}\rangle$, $\mathsf{seq}_2(m_4) = \langle o_{1,1}, o_{2,1}, o_{3,1}\rangle$, $\mathsf{seq}_2(m_5) = \langle o_{1,2}, o_{2,2}, o_{1,3}, o_{2,3}, o_{3,2}, o_{3,3}\rangle$, and $\mathsf{seq}_2(m_6) = \langle o_{1,4}, o_{2,4}, o_{3,4}\rangle$ indicated by the ▪▶ arrows.

5 Distributed scheduler framework

In this section, we explain our method to generate a schedule for a modular, tightly coupled production line. We propose a MAS architecture (Section 5.1) that propagates

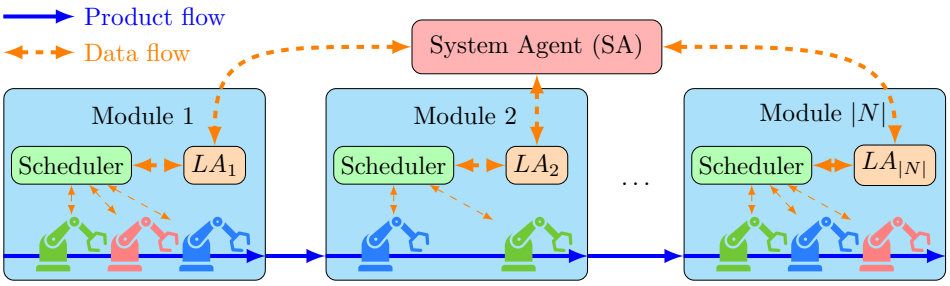

Figure 4: Production line assembly and agent architecture. The system agent (SA) interacts with the local agents (LAs) and the LAs interact with the module's scheduler.

constraints between modules until all modules have reached a consensus on mutually consistent schedules. Figure 4 shows an updated version of Figure 1 with the added MAS coordinating between schedulers. We define the (system and local) agents and the information that they exchange to produce a schedule for a production line in Sections 5.2 and 5.3. Section 5.4 discusses the requirements on the local schedulers and the local agents for the proposed approach to work. In Section 5.5, we prove soundness of the approach. Section 5.7 concludes.

5.1 Multi-agent system architecture

Our MAS architecture is composed of a system agent (SA) and multiple local agents (LAs). Each local agent LA_i is in charge of module n_i, contains the local problem τ_i, and interacts with its scheduler. This architecture allows the modules to only share a small portion of their inner workings (for reasons of reusability and/or because they are produced by different manufacturers) by only sharing timing bounds on their input or output operations.

The LAs communicate with the local schedulers and model their space of feasible solutions by representing the relative start resp. completion times of the input and output operations of all pairs of jobs in the system with a lower and upper bound. The bounds are sent to the SA that forwards them to the other LAs in the system. Each LA_i models the received bounds as constraints of their local re-entrant problem. Due to transfer-point constraints CT1 and CT2, output constraints of a module cannot be mapped directly to input constraints of the next one. Hence, since the SA knows all the transfer points, it translates the constraints between modules.

Example 5.1 Suppose we schedule Example 4.1 from Section 4. Figure 5 shows a general overview of the workings of the constraint propagation mechanism, of which the details are elaborated in the upcoming subsections. First, each LA collects constraints in the form of minimal setup times and due dates for its last resp. first operations resulting from the chosen local machine sequences. These bounds are included in the local constraint-graph representation of the scheduling problem at hand, as indicated in Figure 5a (→ arrows), and sent to the SA. Then, the SA translates and propagates these constraints. Thus, the bound between $o_{2,1}$ and $o_{3,1}$ with value 135 from module n_2, obtained from the length of

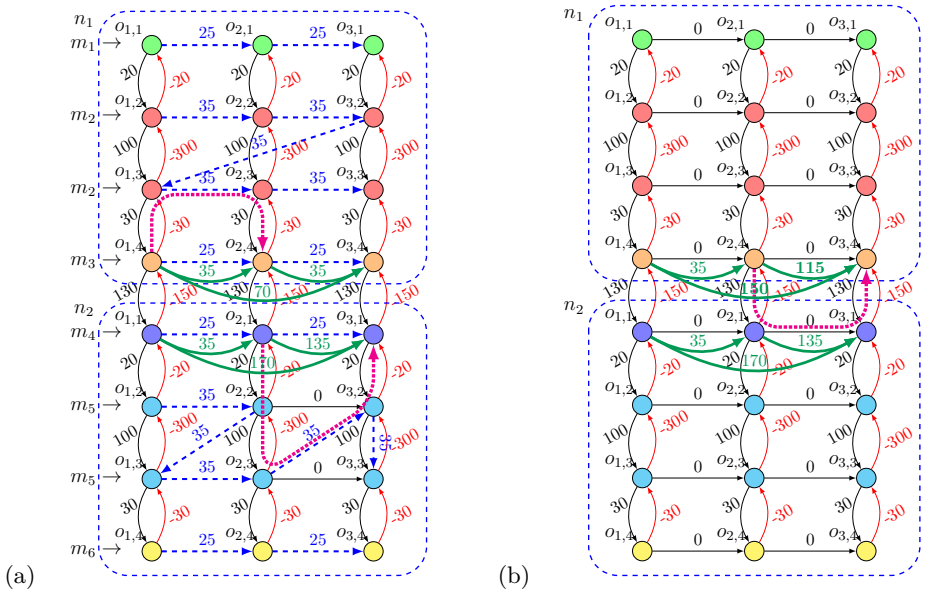

Figure 5: Propagation of constraints between modules n_1 and n_2 of Example 4.1. Fig. (a) shows an extension of Figure 3 with the border constraints, computed by the LAs, drawn as ➔ arrows. The ⇢ arrows show some of the longest paths used to compute the value of these constraints. Fig. (b) shows a constraint-graph result of propagating the constraints between modules n_1 and n_2. The graph is shown without the sequence edges for simplicity. Thus, now in module n_1 the value of the constraint from $o_{2,4}$ to $o_{3,4}$ is updated to 115 from the previous value 35 as a result of the longest path highlighted with the ⇢ arrow. This propagation is done by the SA.

the longest path highlighted by the ⇢ arrow in Figure 5a, is propagated to module n_1 as a bound between $o_{2,4}$ and $o_{3,4}$ with value 115. To do so, the SA computes the longest path in the transfer point. This path is highlighted by the ⇢ arrow in Figure 5b.

5.2 System agent

The SA uses Algorithm 1 to try to find a consensus between the schedules of the LAs and forwards the necessary constraints such that each LA generates a schedule that, together, provide a globally feasible schedule.

5.2.1 Finding consensus

To find a consensus between the LAs, the SA follows an iterative approach in lines 5 to 21. During this step, the LAs generate constraints that the SA stores in MB^{in} and MB^{out} (lines 7 to 9). After collecting all constraints, the SA translates and forwards them to the appropriate destination according to the defined transfer points (lines 10 to 14).

Algorithm 1: System agent

input : Production line $P = (N, T)$
output : Feasible schedule Ω for P or `null` if no feasible schedule is found

1 **function** `ModularScheduling`$(P = (N, T))$
2 $\quad \mathsf{MB}^{\text{in}} \leftarrow \mathsf{MB}^{\text{out}} \leftarrow \mathsf{MB}_{init}$ ▷ function mapping bounds of each module
3 $\quad k \leftarrow 0$
4 $\quad$ propagate_ub $\leftarrow$ `false` ▷ if `true` then also propagate upper bounds
5 $\quad$ **loop**
6 $\quad\quad \mathsf{MB}^{\text{in}}_{prev}, \mathsf{MB}^{\text{out}}_{prev} \leftarrow \mathsf{MB}^{\text{in}}, \mathsf{MB}^{\text{out}}$ ▷ save bounds of previous iteration
7 $\quad\quad$ **foreach** $i \in \{1, ..., |N|\}$ **do** ▷ collect constraints
$\quad\quad\quad$ ▷ ask LA_i for bounds of $\tau_{i,k}$ and generate $\tau_{i,k+1}$ (Section 5.3.1)
8 $\quad\quad\quad \mathsf{MB}^{\text{in}}(i)$, $\mathsf{MB}^{\text{out}}(i)$, result $\leftarrow LA_i$`.LAObtainBounds(`k, propagate_ub`)`
9 $\quad\quad\quad$ **if** result $=$ *not OK* **then return** *null* ▷ no solution found
10 $\quad\quad$ **foreach** $T_i \in T$ **do** ▷ propagate constraints
11 $\quad\quad\quad$ Translate $\mathsf{MB}^{\text{out}}(i)$ to $\mathsf{MB}^{\text{in}}_{*}(i+1)$ using Equations 1 and 2
12 $\quad\quad\quad LA_{i+1}$`.LAAddIn(`k, $\mathsf{MB}^{\text{in}}_{*}(i+1)$`)` ▷ updates $\tau_{i+1,k+1}$ (Equations 5 and 6)
13 $\quad\quad\quad$ Translate $\mathsf{MB}^{\text{in}}(i+1)$ to $\mathsf{MB}^{\text{out}}_{*}(i)$ using Equations 3 and 4
14 $\quad\quad\quad LA_i$`.LAAddOut(`k, $\mathsf{MB}^{\text{out}}_{*}(i)$`)` ▷ updates $\tau_{i,k+1}$ (Equations 7 and 8)
$\quad\quad$ ▷ check termination conditions
15 $\quad\quad$ **if** $\mathsf{MB}^{\text{in}}_{prev} = \mathsf{MB}^{\text{in}} \wedge \mathsf{MB}^{\text{out}}_{prev} = \mathsf{MB}^{\text{out}}$ **then**
16 $\quad\quad\quad$ **if** propagate_ub **then**
17 $\quad\quad\quad\quad$ **exit loop** ▷ convergence achieved
18 $\quad\quad\quad$ **else**
$\quad\quad\quad\quad$ ▷ lower bounds converged, propagate upper bounds
19 $\quad\quad\quad\quad$ propagate_ub $\leftarrow$ `true`
20 $\quad\quad$ **if** $k \geq k^{\max}$ **then return** *null* ▷ limit on iterations reached
21 $\quad\quad k \leftarrow k + 1$
$\quad$ ▷ convergence achieved, generate schedule
22 $\quad \forall j \in J : \mathsf{s}(j) \leftarrow 0$ ▷ map of job start times
23 $\quad \Omega_1 \leftarrow LA_1$`.LASchedule(s)` ▷ LA_1 generates schedule of first module
24 $\quad$ **for** $i = 1$ **to** $|N| - 1$ **do**
25 $\quad\quad \forall j \in J : \mathsf{s}(j) \leftarrow \Omega_i(\mathsf{lst}_i(j)) + \mathsf{prc}_i(\mathsf{lst}_i(j)) + \mathsf{ts}_i(j)$
26 $\quad\quad \Omega_{i+1} \leftarrow LA_{i+1}$`.LASchedule(s)` ▷ LA_{i+1} generates schedule
27 $\quad \Omega \leftarrow \bigcup_{i \in \{1,...,|N|\}} \Omega_i$
28 $\quad$ **return** Ω

At every iteration k, the LAs use the `LAObtainBounds` function of line 8 (described in Algorithm 2) to obtain the bounds abstracting the space of feasible solutions of their local problems $\tau_{i,k}$. Initially, the problem tuple is the one provided by the module, i.e., $\tau_{i,0} = \tau_i$. The local problem to be solved is updated in every iteration of the propagation step with the bounds received from neighbouring modules. The `LAObtainBounds` method provides a pair $(t^{\text{LB}}, t^{\text{UB}})$ with a lower and upper bound for each pair of jobs in the system. Section 5.2.3 precisely defines these bounds. The LA obtains the bounds by evaluating the sequences of operations per machine obtained from the local scheduler. The bounds correspond to the lengths of the longest paths between boundary operations in a constraint-graph representation of the local

scheduling problem (⇢ arrows in Figure 5a). After producing the bounds, the LA also creates a new problem tuple $\tau_{i,k+1} = (J_i, M_i, \mathsf{ops}_i, \mathsf{mch}_i, \mathsf{prc}_i, \mathsf{si}_{i,k+1}, \mathsf{sd}_i, \mathsf{dd}_{i,k+1})$ and adds the bounds into $\tau_{i,k+1}$ as constraints (Section 5.3.2, the → arrows in the constraint-graph representation in Figure 5a). This problem tuple will be updated in the constraint propagation phase with the constraints of other modules. The details are further explained in Section 5.3.1.

The bounds of input operations are collected in MB^{in} and the bounds of output operations in MB^{out}. In our initialisation step, lines 2 to 4, we define the MB_{init} function as a function returning the tuple $(0, \perp)$ for each module and pair of jobs from production line P. This captures the fact that, initially, no bounds are known on the relative input and output times of jobs in a module (the symbol $\perp$ represents the absence of an upper bound).

There might be instances where a local scheduler fails to find a solution for a given problem. This could be due to the problem being unsolvable from the beginning or because the propagated bounds have shrunk the feasible solution space of the local problem to such an extent that no viable solutions are left. When this happens, the scheduling process is halted (line 9). A back-up solution should be considered for such cases. Depending on the local schedulers used, it may be possible to use back-up schedules (see Section 7); alternatively, some of the problem constraints can be relaxed.

After the constraints are collected, for each transfer point T_i, the SA propagates the constraints of the output operations of module n_i to module n_{i+1} (line 12) and propagates the constraints of input operations of module n_{i+1} to n_i (line 14). Because the constraints cannot be mapped directly to an adjacent module, due to transfer-point-constraints CT1 and CT2, the SA translates them to $\mathsf{MB}^{\text{in}}_*(i+1)$ and $\mathsf{MB}^{\text{out}}_*(i)$, respectively, before sending them (lines 11 and 13). This translation is explained in Section 5.2.4. The extra constraints are included in the sequence-independent setup times, $\mathsf{si}_{i,k+1}$, and the due dates, $\mathsf{dd}_{i,k+1}$, respectively. This process is explained in Section 5.3.2 and illustrated in Figure 5b (the → arrows).

This iterative process stops when convergence is achieved. That is, when all the bounds MB^{in} and MB^{out} are the same for two consecutive iterations (line 17). This means that the modules have reached a consensus on their constraints and that the constraint requirements of one module match the constraint requirements of its neighbours for each module in the system. To avoid reducing the space of feasible schedules of modules too fast, Algorithm 1 defers the propagation of upper bounds. Until line 19 is reached, only lower bounds are being propagated. Function `LAObtainBounds` only outputs lower bounds when propagate_ub is false (see Section 5.3.1). After line 19 is reached, propagate_ub is true and upper bounds are propagated.

In some cases, if the local schedulers repeatedly alternate between the same schedules, the modules do not reach a consensus. This problem cannot be avoided without imposing additional assumptions on the local schedulers, as explained in Section 6.2. That is why there is a limit of $k^{\max}$ iterations in line 20. When this limit is reached, Algorithm 1 terminates without providing a schedule.

Example 5.2 Continuing Example 5.1, assume that the local schedulers always choose the same sequences. Table 3 shows the values of the bounds that the SA receives, translates and propagates across iterations. $\mathsf{MB}^{\text{in}}(1)$ and $\mathsf{MB}^{\text{out}}(2)$ are omitted because modules n_1 and n_2 do not have a predecessor and successor respectively.

In Iteration 1, only lower bounds are propagated. Thus, the lower diagonal parts of the boxes in Table 3 are empty. LA_2 obtains the value 135 from the length of the longest path ⇢ in Figure 5a. The SA translates the value to 115 by finding the length of the longest path from $o_{2,4}$ to $o_{3,4}$ in the transfer point (⇢ in Figure 5b), see Section 5.2.4. In Iteration 2, module n_1 has incorporated the constraints of $\mathsf{MB}^{\text{out}}_*(1)$ of the previous iteration so that the bounds that it sends to the SA satisfy these constraints. That is why the bound from job j_2 to j_3 in $\mathsf{MB}^{\text{out}}(1)$ in the second iteration is 115.

Iteration 3 reaches lower bounds convergence and hence is identical to Iteration 2; it is omitted from Table 3. The SA continues iterating using also upper bounds (setting `propagate_ub` to `true` in the algorithm). The upper bounds are represented as negative values starting from Iteration 4. In Iteration 5, module n_2 has incorporated the upper bounds of module n_1 and its schedule satisfies them. Finally, in Iteration 6, not shown in Table 3, all LAs produce the same bounds as in Iteration 5 and consensus has been reached. The schedule generation that then follows is explained in the running example in Section 5.2.2.

	From \ To	$\mathsf{MB}^{\text{out}}(1)$			$\mathsf{MB}^{\text{in}}_*(2)$			$\mathsf{MB}^{\text{in}}(2)$			$\mathsf{MB}^{\text{out}}_*(1)$		
		j_1	j_2	j_3	j_1	j_2	j_3	j_1	j_2	j_3	j_1	j_2	j_3
Iter. 1	j_1	0	35	70	0	15	50	0	35	170	0	15	150
	j_2		0	35		0	15		0	135		0	115
	j_3			0			0			0			0
Iter. 2	j_1	0	35	**150**	0	15	**130**	0	35	170	0	15	150
	j_2		0	**115**		0	**95**		0	135		0	115
	j_3			0			0			0			0
Iter. 4	j_1	0	35	150	0	15	130	0	35	170	0	15	150
	j_2	**-150**	0	115	**-170**	0	95	**-265**	0	135	**-285**	0	115
	j_3	**-265**	**-230**	0	**-285**	**-250**	0			0			0
Iter. 5	j_1	0	35	150	0	15	130	0	35	170	0	15	150
	j_2	-150	0	115	-170	0	95	**-150**	0	135	**-170**	0	115
	j_3	-265	-230	0	-285	-250	0	**-285**	**-250**	0	**-305**	**-270**	0

Table 3: Values of the bounds and their translation across iterations of the running example. Changes between iterations are highlighted in bold face. Each value from j_u to j_v with $u < v$ represents a lower bound while each negative value from j_v to j_u represents an upper bound (i.e., as a negative lower bound in the opposite direction). Each row represents a single iteration. Repeated iterations (3 and 6) have been omitted.

Algorithm 1 detects convergence by comparing the obtained bounds in two consecutive iterations. This can be implemented more efficiently by evaluating the bounds (MB^{in}, MB^{out}) that the modules send and checking whether the translated bounds ($\mathsf{MB}^{\text{in}}_*$, $\mathsf{MB}^{\text{out}}_*$) would really update the local problems of the modules. If, for instance,

the bounds $\mathsf{MB}(i)$ that a module n_i sends are "stronger" than the bounds $\mathsf{MB}_*(i)$ that will be incorporated in the module, then the local problem will not be affected. That is, if, for all pairs of jobs, all the lower bounds of $\mathsf{MB}(i)$ are at least equal to the lower bounds of $\mathsf{MB}_*(i)$ and the upper bounds of $\mathsf{MB}(i)$ are at most equal to the upper bounds of $\mathsf{MB}_*(i)$, then convergence has occurred. In the above example, for instance, this means that already in Iteration 2, it is possible to detect convergence of lower bounds, avoiding the explicit computation of Iteration 3.

5.2.2 Production-line schedule generation

If a consensus between modules is found, then the input and/or output constraints of a module satisfy the constraints of its neighbours for each module in the system. Thus, it is then possible to generate a feasible schedule for the production line. We generate a schedule in a distributed manner in lines 22 to 27 of Algorithm 1 by iteratively providing the start times of the input operations of a module (lines 23 and 26) and obtaining the completion times of its output operations.

Definition 1 (ASAP schedule) We define the as-soon-as-possible (ASAP) schedule with the lower bounds function $\mathsf{s} : J \hookrightarrow \mathbb{R}$ for a flow shop τ and the sequences seq as a feasible schedule Ω where $\Omega(\mathsf{fst}(j)) \geq \mathsf{s}(j)$ for all j in the domain of s, where the start times of the operations of each machine m in M follow the order of the sequence $\mathsf{seq}(m)$ (i.e., $\Omega(\mathsf{seq}(m)(i)) < \Omega(\mathsf{seq}(m)(j))$ if and only if $i < j$), and where $\Omega(o)$ is minimal for all o in O.

In lines 23 and 26, the `LASchedule` method of the LAs (defined in Algorithm 5 of Section 5.3.3) is called. It generates the ASAP schedule from the lower bounds function s and the sequences seq_i that the local schedulers used to reach consensus. The ASAP schedule is required for the global schedule to be valid, as elaborated in Section 5.5. The values of the lower bounds function s for each next module are determined by adding the transfer times of the output operations of the previous module to the scheduled completion time of these output operations. This ensures that a job in n_{i+1} cannot start until it is finished in module n_i and has arrived at module n_{i+1}. We prove that the global schedule generated in these steps is feasible in Section 5.5. We provide the precise requirements for the local schedulers in Section 5.4. Note that, even though line 25 refers to start and processing times of an operation, in a practical implementation, it may not be feasible or desirable that operation processing times and/or start times of internal operations of modules are made available to the SA. In such a case, it suffices to communicate only completion times of the local schedules from the LAs to the SA. Algorithm 1 provides a global schedule containing all operations so we may reason about feasibility of this global schedule.

Example 5.3 Recall Example 5.2. After Iteration 6, the SA asks LA_1 to generate a schedule. Figure 6a shows a constraint graph with schedule Ω_1. The SA receives the output values (155, 190, and 305) of Ω_1, adds the minimum transport times (130), and passes these values (285, 320, and 435) to LA_2 as lower bounds on the start times of its first operations. LA_2 generates the ASAP schedule Ω_2 using these values as lower bounds and produces the schedule of Figure 6b. Notice how the start time of operation $o_{3,1}$, 455, is greater than the received

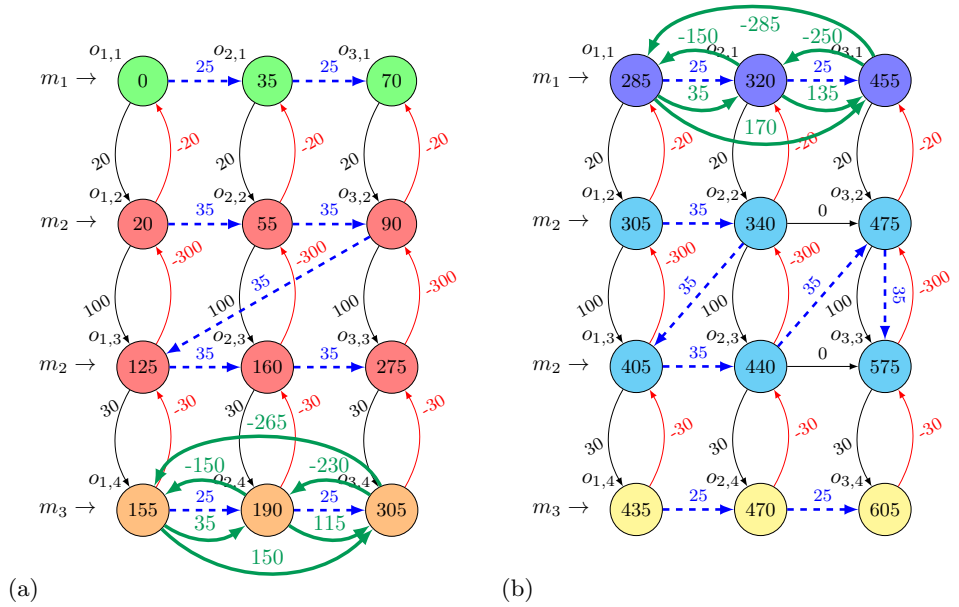

Figure 6: Constraint graph of modules n_1 (a) and n_2 (b) of the running example with schedule and final bounds (➞ arrows). The value inside each node is the start time of that operation according to global schedule Ω. Schedule Ω is built from local schedules Ω_1 and Ω_2 that are globally feasible in the production line defined in Example 4.1.

lower bound 435, but that this value still satisfies the transfer due date constraint of 150 (see Figure 3).

5.2.3 Constraints as bounds

The SA and the LAs exchange bounds as an abstraction of the local problem constraints. We first formally define them in Definition 3 and then explain how the SA gathers them. Section 5.3.1 explains how an LA obtains these bounds from a local problem tuple.

We model the space of feasible solutions of a module by representing, for each pair of jobs j_u and j_v with $u < v$, the lower and upper bound on the difference between the start times of two input operations and on the difference between completion times of two output operations. Each local agent LA_i asks the local scheduler to generate the sequences of operations seq_i for the machines of the module and produces the bounds by evaluating these sequences.

Definition 2 (Feasible schedules given machine sequences) We define $S_{\mathsf{seq}_{i,k}}$ as the set of all feasible schedules $\Omega_{i,k}$ of problem $\tau_{i,k}$ at iteration k that have the machine sequences $\mathsf{seq}_{i,k}$.

Definition 3 (Bounds) For the input operations of $\tau_{i,k}$, we define $\boldsymbol{t^{\text{in,LB}}_{i,k,u,v}}$ and $\boldsymbol{t^{\text{in,UB}}_{i,k,u,v}}$ as the minimum and maximum difference $\Omega_{i,k}(\mathsf{fst}_i(j_v)) - \Omega_{i,k}(\mathsf{fst}_i(j_u))$, respectively, over all $\Omega_{i,k}$ in $S_{\mathsf{seq}_{i,k}}$, and, for the output operations, we define $\boldsymbol{t^{\text{out,LB}}_{i,k,u,v}}$ and $\boldsymbol{t^{\text{out,UB}}_{i,k,u,v}}$ as the minimum and maximum difference $\Omega_{i,k}(\mathsf{lst}_i(j_v)) + \mathsf{prc}_i(\mathsf{lst}_i(j_v)) - \big(\Omega_{i,k}(\mathsf{lst}_i(j_u)) + \mathsf{prc}_i(\mathsf{lst}_i(j_u))\big)$, respectively, over all $\Omega_{i,k}$ in $S_{\mathsf{seq}_{i,k}}$.

It is possible that the upper bounds $t^{\text{in,UB}}_{i,k,u,v}$ or $t^{\text{out,UB}}_{i,k,u,v}$ do not exist, as it may be possible for operations to be delayed indefinitely. For simplicity, in the remainder, we drop super- and subscripts that are not relevant or can be deduced from the context.

Because the SA and the LAs exchange bounds for all pairs of jobs, we define the mapping function $B : \{u \in \mathbb{N} \mid 1 \leq u \leq |J|\}^2 \hookrightarrow (\mathbb{R} \cup \{\perp\})^2$ that returns a tuple $(t^{\text{LB}}_{u,v}, t^{\text{UB}}_{u,v})$ representing the value of the lower $(t^{\text{LB}}_{u,v})$ and upper $(t^{\text{UB}}_{u,v})$ bound for each pair of jobs j_u and j_v with $u < v$. Each element of the returned tuple can either be a real value or the symbol $\perp$ meaning the absence of a bound. For consistency, we require the second element of the pair to be at least the value of the first if both exist. So, we have, for instance, that $B(u,v) = (t^{\text{in,LB}}_{i,k,u,v}, t^{\text{in,UB}}_{i,k,u,v})$. We define the mappings MB^{in} and MB^{out} used in Algorithm 1 as functions with domain $\{i \in \mathbb{N} \mid 1 \leq i \leq |N|\}$ mapping a module index to its mapping of bounds (i.e., a function B as defined above); $\mathsf{MB}^{\text{in}}(i)$ and $\mathsf{MB}^{\text{out}}(i)$ contain all the input and output bounds for all job pairs of module n_i, respectively. Recall Example 5.2 that illustrates the use of these mappings in the iterative consensus process.

5.2.4 Constraint propagation

The maps of bounds B^{in} and B^{out} generated by the local agents in each iteration of the main loop of Algorithm 1 are collected by the SA in the mappings MB^{in} and MB^{out}, respectively. Since they represent constraints of the sending module, the SA translates them to valid constraints of the module that receives them. For the transfer point T_i, the SA translates the maps $\mathsf{MB}^{\text{out}}(i)$ and $\mathsf{MB}^{\text{in}}(i+1)$ into the maps $\mathsf{MB}^{\text{in}}_*(i+1)$ and $\mathsf{MB}^{\text{out}}_*(i)$, respectively. LA_i and LA_{i+1} then receive $\mathsf{MB}^{\text{out}}_*(i)$ and $\mathsf{MB}^{\text{in}}_*(i+1)$, respectively, which they use to update the problems $\tau_{i,k+1}$ and $\tau_{i+1,k+1}$.

The translation process accounts for all the constraints of the transfer point T_i. Figure 7 shows a constraint graph of the possible transfer-point constraints between two jobs j_u and j_v of two consecutive modules n_i and n_{i+1}. The bounds $\mathsf{MB}^{\text{out}}(i)(u,v) = \big(t^{\text{out,LB}}_{i,k,u,v}, t^{\text{out,UB}}_{i,k,u,v}\big)$ are translated into $\mathsf{MB}^{\text{in}}_*(i+1)(u,v) = \big(t^{*\text{in,LB}}_{i+1,k,u,v}, t^{*\text{in,UB}}_{i+1,k,u,v}\big)$ and $\mathsf{MB}^{\text{in}}(i+1)(u,v) = \big(t^{\text{in,LB}}_{i+1,k,u,v}, t^{\text{in,UB}}_{i+1,k,u,v}\big)$ into $\mathsf{MB}^{\text{out}}_*(i)(u,v) = \big(t^{*\text{out,LB}}_{i,k,u,v}, t^{*\text{out,UB}}_{i,k,u,v}\big)$. If the upper bounds $t^{\text{in,UB}}_{i+1,k,u,v}$ or $t^{\text{out,UB}}_{i,k,u,v}$ do not exist, they are not propagated.

$t^{*\text{in,LB}}_{i+1,k,u,v}$ is the propagation of the output constraint $t^{\text{out,LB}}_{i,k,u,v}$ of module n_i in iteration k when applied to the input of module n_{i+1} in iteration $k+1$, accounting for the transfer constraints. In the graph from Figure 7, $t^{\text{out,LB}}_{i,k,u,v}$ is represented as part of the weight of the edge from $\mathsf{lst}_i(j_u)$ to $\mathsf{lst}_i(j_v)$. Because $t^{\text{out,LB}}_{i,k,u,v}$ is a constraint on the relative completion time between the operations, whereas the edges in a constraint graph represent constraints on start times, the constraint in the graph is corrected

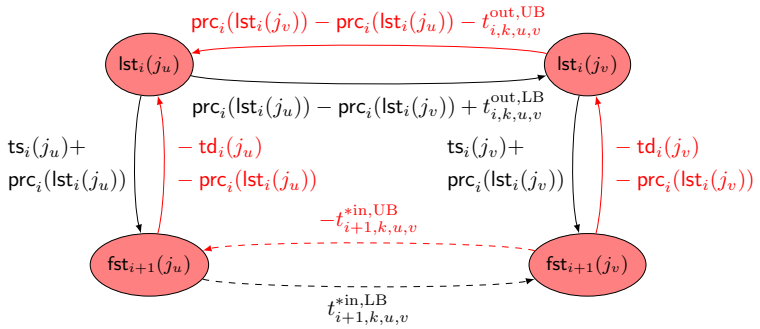

Figure 7: All possible constraints between jobs j_u and j_v at transfer point T_i.

with the processing times $\mathsf{prc}_i(\mathsf{lst}_i(j_v))$ and $\mathsf{prc}_i(\mathsf{lst}_i(j_u))$. Bound $t^{*\text{in,LB}}_{i+1,k,u,v}$ is represented by the dashed edge from $\mathsf{fst}_{i+1}(j_u)$ to $\mathsf{fst}_{i+1}(j_v)$. The latter is to be added to $\tau_{i+1,k+1}$. To properly model all the constraints of the transfer point, the value of this edge needs to be equal to the length of the longest path that goes from $\mathsf{fst}_{i+1}(j_u)$ to $\mathsf{fst}_{i+1}(j_v)$. This path is the sequence of nodes $\langle \mathsf{fst}_{i+1}(j_u), \mathsf{lst}_i(j_u), \mathsf{lst}_i(j_v), \mathsf{fst}_{i+1}(j_v)\rangle$. Putting this together, we obtain:

$$\begin{aligned} t^{*\text{in,LB}}_{i+1,k,u,v} &= -\mathsf{td}_i(j_u) - \mathsf{prc}_i(\mathsf{lst}_i(j_u)) + \mathsf{prc}_i(\mathsf{lst}_i(j_u)) - \mathsf{prc}_i(\mathsf{lst}_i(j_v)) \\ &\quad + t^{\text{out,LB}}_{i,k,u,v} + \mathsf{prc}_i(\mathsf{lst}_i(j_v)) + \mathsf{ts}_i(j_v) \\ &= -\mathsf{td}_i(j_u) + \mathsf{ts}_i(j_v) + t^{\text{out,LB}}_{i,k,u,v} \end{aligned} \tag{1}$$

For $t^{*\text{in,UB}}_{i+1,k,u,v}$, we use the path that goes from $\mathsf{fst}_{i+1}(j_v)$ to $\mathsf{fst}_{i+1}(j_u)$ with negated length. We use the same method for the output bounds from $\mathsf{MB}^{\text{in}}(i+1)(u,v) = \left(t^{\text{in,LB}}_{i+1,k,u,v}, t^{\text{in,UB}}_{i+1,k,u,v}\right)$ to $\mathsf{MB}^{\text{out}}_{*}(i)(u,v) = \left(t^{*\text{out,LB}}_{i,k,u,v}, t^{*\text{out,UB}}_{i,k,u,v}\right)$. As a result, we obtain:

$$t^{*\text{in,UB}}_{i+1,k,u,v} = -\mathsf{ts}_i(j_u) + t^{\text{out,UB}}_{i,k,u,v} + \mathsf{td}_i(j_v) \tag{2}$$

$$t^{*\text{out,LB}}_{i,k,u,v} = \mathsf{ts}_i(j_u) + t^{\text{in,LB}}_{i+1,k,u,v} - \mathsf{td}_i(j_v) \tag{3}$$

$$t^{*\text{out,UB}}_{i,k,u,v} = -\mathsf{ts}_i(j_v) + t^{\text{in,UB}}_{i+1,k,u,v} + \mathsf{td}_i(j_u) \tag{4}$$

Example 5.4 Recall Example 5.1. Figure 5b illustrates how the bound of module n_1 between operations $o_{2,4}$ and $o_{3,4}$ is obtained. Recall that $t_{i,k,u,v}$ represents a bound from the first/last operation of j_u to the first/last operation of j_v of module n_i at iteration k. The superscript "in" denotes input bounds (first operations of jobs), "out" denotes output bounds (last operations of jobs), "LB" denotes lower bounds and "UB" denotes upper bounds. LA_2 produces the bound $t^{\text{in,LB}}_{2,1,2,3} = 135$ at iteration $k = 1$ ($\mathsf{MB}^{\text{in}}(2)(2,3)$ in Example 5.2, highlighted in red in Table 3). To obtain the translated output lower bound of module n_2, $t^{*\text{out,LB}}_{1,1,2,3}$, we use Equation 3. The transfer setup time is $\mathsf{ts}_1(j_2) = 110$ and the due date is $\mathsf{td}_1(j_3) = 130$.

Putting it all together:

$$t^{*\text{out,LB}}_{1,1,2,3} = \underbrace{110}_{\mathsf{ts}_1(j_2)} + \underbrace{135}_{t^{\text{in,LB}}_{2,1,2,3}} - \underbrace{130}_{\mathsf{td}_1(j_3)} = 115$$

5.3 Local agent

The LA bridges the local scheduler and the SA to produce a globally feasible schedule. To do so, it (i) produces bounds abstracting the local schedules (`LAObtainBounds`, Section 5.3.1), (ii) consolidates internal bounds and bounds received from the SA as constraints in the local problem (`LAAddIn`, `LAAddOut`, Section 5.3.2), and (iii) generates the local schedule and forwards the times of output operations (`LASchedule`, Section 5.3.3).

5.3.1 Obtaining bounds

The LA is responsible for producing the bounds of Definition 3 needed by the SA. The LA uses Algorithm 2 to compute these bounds at every iteration k. Initially, it requests the local scheduler to generate the sequences of operations per machine (line 2). It then generates a constraint graph from these sequences (line 4) and evaluates the sequences (lines 5 to 17) to produce the bounds. In our implementation, we reuse the graph across iterations, updating only the constraints that have changed. For simplicity of presentation, Algorithm 2 derives the full graph for each iteration.

Because the lower and upper bounds $t^{\text{LB}}_{i,k}$ and $t^{\text{UB}}_{i,k}$ are related to the minimum or maximum delays between two operations, the values of these bounds are obtained by evaluating the longest path between the operations that the bounds represent. The `LongestPath` function is used to compute the longest path from the node `from` to the node `to` in the graph passed as the first parameter. The value of $t^{\text{in,LB}}_{i,k,u,v}$ is the length of the longest path that goes from $\mathsf{fst}_i(j_u)$ to $\mathsf{fst}_i(j_v)$ (line 6). This length is the same as the minimum value of $\Omega_{i,k}(\mathsf{fst}_i(j_v)) - \Omega_{i,k}(\mathsf{fst}_i(j_u))$ for all schedules $\Omega_{i,k}$ in $S_{\mathsf{seq}_{i,k}}$. The value of bound $t^{\text{in,UB}}_{i,k,u,v}$ is determined in a similar way (line 7). The path with length w from $\mathsf{fst}_i(j_v)$ to $\mathsf{fst}_i(j_u)$ imposes the constraint $\Omega_{i,k}(\mathsf{fst}_i(j_v)) + w \leq \Omega_{i,k}(\mathsf{fst}_i(j_u))$. Since $t^{\text{in,UB}}_{i,k,u,v}$ represents the constraint $\Omega_{i,k}(\mathsf{fst}_i(j_u)) + t^{\text{in,UB}}_{i,k,u,v} \geq \Omega_{i,k}(\mathsf{fst}_i(j_v))$, the value of $t^{\text{in,UB}}_{i,k,u,v}$ is $-w$. Similarly, the values for $t^{\text{out,LB}}_{i,k,u,v}$ and $t^{\text{out,UB}}_{i,k,u,v}$ are computed from longest paths in the constraint graph in lines 8 and 9, respectively. But, because these bounds are defined in terms of completion times instead of start times, the computed values are updated in lines 10 and 11. If the constraint graph has no path for a given bound, `LongestPath` returns the symbol $\perp$ representing the absence of a bound.

Algorithm 2 only returns the upper bounds if $\mathsf{propagate\_ub}$ is true. Otherwise, it returns $\perp$ as value for the upper bounds. Thus, in lines 12 to 17 the right value is set. This is used in Algorithm 1 to avoid reducing the space of feasible solutions too fast. It is further explained in Section 5.2.1.

In line 18, the LA generates the new problem tuple $\tau_{i,k+1}$ that will be updated with the bounds from its neighbours. As mentioned, our implementation maintains a constraint graph that is incrementally updated in every iteration. The explicit problem

tuple update is included to clarify that the local scheduling problem to be solved changes in every iteration and to support the soundness proof later on.

Finally, to expedite convergence, the LA incorporates the bounds that it transmits to the SA as constraints for the problem tuple of the next iteration $\tau_{i,k+1}$ in lines 19 to 20. This method accelerates convergence as the bounds are disseminated to the neighbours during the propagation phase, and in turn, the neighbours send these values (or tighter constraints) back in their propagation step of the next iterations.

Example 5.5 As already introduced, in Figure 5a, the ⇢ arrow of module n_1 is the longest path used to compute the value of the bound, $t^{\text{out,LB}}_{1,1,1,2}$, from $o_{1,4}$ to $o_{2,4}$ in iteration 1. The paths of the other bounds of module n_1 are not shown. Similarly, for module n_2, the ⇢ arrow is the longest path used to compute the value of the bound $t^{\text{in,LB}}_{2,1,2,3}$ from $o_{2,1}$ to $o_{3,1}$.

Algorithm 2: Obtaining $t^{\text{in,LB}}_{i,k,u,v}$, $t^{\text{in,UB}}_{i,k,u,v}$, $t^{\text{out,LB}}_{i,k,u,v}$, $t^{\text{out,UB}}_{i,k,u,v}$

input : Iteration k, Boolean propagate_ub
output : Maps of bounds B^{in} and B^{out}

1 **function** `LAObtainBounds`(k, propagate_ub)
  ▷ $\tau_{i,k}$ is the current problem tuple for LA_i
2 $\mathsf{seq}_{i,k}$, result ← `Scheduler`($\tau_{i,k}$) ▷ obtain scheduling sequences
3 **if** result = *not OK* **then return** $\emptyset, \emptyset$, *not OK* ▷ no solution found
4 Create graph $\mathcal{G}_{\mathsf{seq}_{i,k}}$ from $\tau_{i,k}$ and $\mathsf{seq}_{i,k}$ ▷ Section 3.3
5 **foreach** $1 \leq u < v \leq |J|$ **do**
6 $t^{\text{in,LB}}_{i,k,u,v} \leftarrow$ `LongestPath`($\mathcal{G}_{\mathsf{seq}_{i,k}}$, *from*=$\mathsf{fst}_i(j_u)$, *to*=$\mathsf{fst}_i(j_v)$)
7 $t^{\text{in,UB}}_{i,k,u,v} \leftarrow$ `-LongestPath`($\mathcal{G}_{\mathsf{seq}_{i,k}}$, *from*=$\mathsf{fst}_i(j_v)$, *to*=$\mathsf{fst}_i(j_u)$)
8 $d^{\text{LB}} \leftarrow$ `LongestPath`($\mathcal{G}_{\mathsf{seq}_{i,k}}$, *from*=$\mathsf{lst}_i(j_u)$, *to*=$\mathsf{lst}_i(j_v)$)
9 $d^{\text{UB}} \leftarrow$ `-LongestPath`($\mathcal{G}_{\mathsf{seq}_{i,k}}$, *from*=$\mathsf{lst}_i(j_v)$, *to*=$\mathsf{lst}_i(j_u)$)
  ▷ adjust for completion times instead of start times
10 $t^{\text{out,LB}}_{i,k,u,v} \leftarrow d^{\text{LB}} - \mathsf{prc}_i(\mathsf{lst}_i(j_u)) + \mathsf{prc}_i(\mathsf{lst}_i(j_v))$
11 $t^{\text{out,UB}}_{i,k,u,v} \leftarrow d^{\text{UB}} - \mathsf{prc}_i(\mathsf{lst}_i(j_u)) + \mathsf{prc}_i(\mathsf{lst}_i(j_v))$
12 **if** propagate_ub **then**
13 $B^{\text{in}}(u,v) \leftarrow (t^{\text{in,LB}}_{i,k,u,v}, t^{\text{in,UB}}_{i,k,u,v})$
14 $B^{\text{out}}(u,v) \leftarrow (t^{\text{out,LB}}_{i,k,u,v}, t^{\text{out,UB}}_{i,k,u,v})$
15 **else**
16 $B^{\text{in}}(u,v) \leftarrow (t^{\text{in,LB}}_{i,k,u,v}, \perp)$
17 $B^{\text{out}}(u,v) \leftarrow (t^{\text{out,LB}}_{i,k,u,v}, \perp)$
18 $\tau_{i,k+1} \leftarrow \tau_{i,k}$ ▷ create new tuple, to be updated when receiving bounds
19 `LAAddIn`(k, B^{in}) ▷ consolidate input bounds
20 `LAAddOut`(k, B^{out}) ▷ consolidate output bounds
21 **return** B^{in}, B^{out}, *OK*

5.3.2 Incorporating constraints

LA_i receives the maps $\mathsf{MB}_*^{\text{in}}(i)$ and $\mathsf{MB}_*^{\text{out}}(i)$ from the SA (lines 10 to 14 of Algorithm 1). The LA uses the constraints of $\mathsf{MB}_*^{\text{in}}(i)$ and $\mathsf{MB}_*^{\text{out}}(i)$ to update the problem $\tau_{i,k+1}$ derived from the problem of the current iteration $\tau_{i,k}$. The bounds $t_{i,k,u,v}^{*\text{in,LB}}$ and $t_{i,k,u,v}^{*\text{in,UB}}$ in the range of $\mathsf{MB}_*^{\text{in}}(i)$ and the bounds $t_{i,k,u,v}^{*\text{out,LB}}$ and $t_{i,k,u,v}^{*\text{out,UB}}$ in the range of $\mathsf{MB}_*^{\text{out}}(i)$ are added as constraints in $\tau_{i,k+1}$. As explained earlier, LA_i incorporates the internal bounds that it generated already in Algorithm 2.

For every input lower bound $t_{i,u,v}^{\text{in,LB}}$ either internal $\left(t_{i,k,u,v}^{\text{in,LB}}\right)$ or external $\left(t_{i,k,u,v}^{*\text{in,LB}}\right)$, i.e., $t_{i,u,v}^{\text{in,LB}}$ in $\left\{t_{i,k,u,v}^{*\text{in,LB}}, t_{i,k,u,v}^{\text{in,LB}}\right\}$, the LA includes the constraint $\Omega_{i,k+1}(\mathsf{fst}_i(j_u))+t_{i,u,v}^{\text{in,LB}} \leq \Omega_{i,k+1}(\mathsf{fst}_i(j_v))$ as a setup constraint $\mathsf{si}_{i,k+1}$ between operations $\mathsf{fst}_i(j_u)$ and $\mathsf{fst}_i(j_v)$. For every input upper bound $t_{i,u,v}^{\text{in,UB}}$ with $t_{i,u,v}^{\text{in,UB}}$ in $\left\{t_{i,k,u,v}^{*\text{in,UB}}, t_{i,k,u,v}^{\text{in,UB}}\right\}$, the LA includes the constraint $\Omega_{i,k+1}(\mathsf{fst}_i(j_u)) + t_{i,u,v}^{\text{in,UB}} \geq \Omega_{i,k+1}(\mathsf{fst}_i(j_v))$ as a due date constraint $\mathsf{dd}_{i,k+1}$ between operations $\mathsf{fst}_i(j_u)$ and $\mathsf{fst}_i(j_v)$. Similarly, for output lower bounds $t_{i,u,v}^{\text{out,LB}}$ in $\left\{t_{i,k,u,v}^{*\text{out,LB}}, t_{i,k,u,v}^{\text{out,LB}}\right\}$, and $t_{i,u,v}^{\text{out,UB}}$ in $\left\{t_{i,k,u,v}^{*\text{out,UB}}, t_{i,k,u,v}^{\text{out,UB}}\right\}$, the LA includes the constraints $\Omega_{i,k+1}(\mathsf{lst}_i(j_u))+t_{i,u,v}^{\text{out,LB}} \leq \Omega_{i,k+1}(\mathsf{lst}_i(j_v))$ and $\Omega_{i,k+1}(\mathsf{lst}_i(j_u))+t_{i,u,v}^{\text{out,UB}} \geq \Omega_{i,k+1}(\mathsf{lst}_i(j_v))$ as a setup constraint $\mathsf{si}_{i,k+1}$ and a due date constraint $\mathsf{dd}_{i,k+1}$ respectively between operations $\mathsf{lst}_i(j_u)$ and $\mathsf{lst}_i(j_v)$. If a setup constraint $\mathsf{si}_{i,k+1}$ already exists, the value is updated with the maximum among the previous one and the new one and if a deadline $\mathsf{dd}_{i,k+1}$ already exists, the value is updated with the minimum among the previous value and the new one. Note that the received bounds need to be adjusted with the processing time of $\mathsf{fst}_i(j_u)$. The constraints are added as follows[2]:

$$\mathsf{si}_{i,k+1}(\mathsf{fst}_i(j_u),\mathsf{fst}_i(j_v)) = \max\left(t_{i,u,v}^{\text{in,LB}} - \mathsf{prc}_i(\mathsf{fst}_i(j_u)),\ \mathsf{si}_{i,k+1}(\mathsf{fst}_i(j_u),\mathsf{fst}_i(j_v))\right) \quad (5)$$

$$\mathsf{dd}_{i,k+1}(\mathsf{fst}_i(j_u),\mathsf{fst}_i(j_v)) = \min\left(t_{i,u,v}^{\text{in,UB}} - \mathsf{prc}_i(\mathsf{fst}_i(j_u)),\mathsf{dd}_{i,k+1}(\mathsf{fst}_i(j_u),\mathsf{fst}_i(j_v))\right) \quad (6)$$

To add the constraints implied by the bounds $\mathsf{MB}_*^{\text{out}}(i)(u,v) = \left(t_{i,k,u,v}^{*\text{out,LB}}, t_{i,k,u,v}^{*\text{out,UB}}\right)$ for the output operations, we use similar equations but account for the fact that the bounds relate to completion times of the last operations of jobs j_u and j_v instead:

$$\mathsf{si}_{i,k+1}(\mathsf{lst}_i(j_u),\mathsf{lst}_i(j_v)) = \max\left(t_{i,u,v}^{\text{out,LB}} - \mathsf{prc}_i(\mathsf{lst}_i(j_v)),\ \mathsf{si}_{i,k+1}(\mathsf{lst}_i(j_u),\mathsf{lst}_i(j_v))\right) \quad (7)$$

$$\mathsf{dd}_{i,k+1}(\mathsf{lst}_i(j_u),\mathsf{lst}_i(j_v)) = \min\left(t_{i,u,v}^{\text{out,UB}} - \mathsf{prc}_i(\mathsf{lst}_i(j_v)),\mathsf{dd}_{i,k+1}(\mathsf{lst}_i(j_u),\mathsf{lst}_i(j_v))\right) \quad (8)$$

To update the problem tuple $\tau_{i,k+1}$ with the module-internal and received constraints, the LA uses Algorithm 3 for the input constraints and Algorithm 4 for the output constraints. Note that the problem tuple $\tau_{i,k+1}$ is initialised with the problem tuple $\tau_{i,k}$ of the previous iteration in function `LAObtainBounds`. Also module-internal bounds are already incorporated in `LAObtainBounds`.

[2]We assume that $\mathsf{si}_{i,k}$ and $\mathsf{dd}_{i,k}$ return 0 resp. ∞ if they are undefined.

Algorithm 3: Updating problem $\tau_{i,k+1}$ with additional input constraints B^{in}

input : Iteration k, input bounds B^{in}
function `LAAddIn`(k, B^{in})
  foreach $\left(t^{\text{in,LB}}_{i,u,v}, t^{\text{in,UB}}_{i,u,v}\right) \in B^{\text{in}}$ **do**
    Update $\mathsf{si}_{i,k+1}$ with $t^{\text{in,LB}}_{i,u,v}$ ▷ Equation 5
    Update $\mathsf{dd}_{i,k+1}$ with $t^{\text{in,UB}}_{i,u,v}$ ▷ Equation 6

Algorithm 4: Updating problem $\tau_{i,k+1}$ with additional output constraints B^{out}

input : Iteration k, output bounds B^{out}
function `LAAddOut`(k, B^{out})
  foreach $\left(t^{\text{out,LB}}_{i,u,v}, t^{\text{out,UB}}_{i,u,v}\right) \in B^{\text{out}}$ **do**
    Update $\mathsf{si}_{i,k+1}$ with $t^{\text{out,LB}}_{i,u,v}$ ▷ Equation 7
    Update $\mathsf{dd}_{i,k+1}$ with $t^{\text{out,UB}}_{i,u,v}$ ▷ Equation 8

Algorithm 5: Generating local Ω_i with lower bounds s

input : Local lower bounds function s
output : Local schedule Ω_i
function `LASchedule`(s)
  Recall last sequences $\mathsf{seq}_{i,k}$ returned by the local scheduler `Scheduler` in the last iteration k
  Build graph $\mathcal{G}_{\mathsf{seq}_{i,k}}$ from $\tau_{i,k}$ and $\mathsf{seq}_{i,k}$ ▷ Section 3.3
  Initialise vector v of node distances to 0
  foreach $j \in \text{dom}(\mathsf{s})$ **do**
    $v[\mathsf{fst}_i(j)] \leftarrow \mathsf{s}(j)$
  $\Omega_i \leftarrow$ `LongestPathLB`($\mathcal{G}_{\mathsf{seq}_{i,k}}$, v) ▷ Longest path with given distances
  return Ω_i

Example 5.6 Recall Example 5.1. Figure 5b showed the propagation of the bound $t^{*\text{out,LB}}_{1,1,2,3}$ with value 115 to module n_1. This bound is added as a sequence-independent setup time of 95 from $o_{2,4}$ to $o_{3,4}$ using Equation 7:

$$\mathsf{si}_{1,2}\left(o_{2,4}, o_{3,4}\right) = \max(\underbrace{115}_{t^{*\text{out,LB}}_{1,1,2,3}} - \underbrace{20}_{\mathsf{prc}_1(o_{3,4})}, \underbrace{0}_{\mathsf{si}_{1,2}(o_{2,4},o_{3,4})}) = 95$$

5.3.3 Schedule generation

The LA uses Algorithm 5 to generate the local schedule. The (partial) function s, defined as $\mathsf{s} : J \hookrightarrow \mathbb{R}$, is a parameter, setting a lower bound on the start time of the first operation for each job for which it is defined.

The ASAP schedule is constructed using the longest path algorithm. The sequences that the local scheduler returns during the final iteration of the convergence step of Algorithm 1 form the graph $\mathcal{G}_{\mathsf{seq}_{i,k}}$ in lines 2 to 3. Then, Algorithm 5 sets the distances of the longest path to 0 for all nodes (line 4), excluding the distances for the nodes of the first operation of every job j in the domain of s that are set to the value of $\mathsf{s}(j)$ (lines 5 to 6).

Finally, the `LongestPathLB` function takes the graph $\mathcal{G}_{\mathsf{seq}_{i,k}}$ and the lower bounds v and determines, for each operation o_1 in O_i, the maximum over all o_2 in O_i of $v[o_2]$ plus the length of the longest path from o_2 to o_1. This provides the ASAP schedule of Definition 1 for the given machine sequences, as any operation starting time smaller than the computed longest paths will violate some constraints. The longest paths provide the smallest values that can satisfy the constraints. This computation is similar to the one used by van Pinxten et al. (2017).

Example 5.7 Recall Example 5.3. Figure 6b shows the schedule Ω_2 of module n_2 as the values inside the nodes in the graph. This schedule is generated using Algorithm 5 and using as start-time lower bounds $\mathsf{s}(1) = 285$, $\mathsf{s}(2) = 320$, and $\mathsf{s}(3) = 435$. Inside Algorithm 5, the start-time lower bounds are used to initialise the longest-path algorithm that finds the longest distances from the source nodes ($o_{1,1}$, $o_{2,1}$, and $o_{3,1}$) to the other nodes in the graph $\mathcal{G}_{\mathsf{seq}_{2,6}}$ built from the problem $\tau_{2,6}$ and the sequences $\mathsf{seq}_{2,6}$ from Iteration 6 (the last iteration before schedule generation).

5.4 Requirements on the LAs and local schedulers

Because the LAs and local schedulers are typically implemented by manufacturers of the modules, we state the requirements they need to adhere to so that the SA can properly coordinate the schedules of all modules.

The local scheduler must provide a schedule either as start times Ω or through the sequences of operations per machine in the module seq. The sequences seq are needed by the LA in Algorithm 2 to produce the bounds that the LAs provides to the SA. If the local scheduler provides start times Ω, the LA extracts the sequences seq from these start times. This ensures that the approach is compatible with many existing schedulers.

After the SA finds consensus between the modules in Algorithm 1, the decision on the final schedule of the module can be done either by the LA by accepting the schedule generated in Algorithm 5 or by the local scheduler. In the second case, the local scheduler must take into account the lower bound on start times s that the SA passes as a parameter to the LAs in lines 22 to 27 of Algorithm 1 and it must provide an ASAP schedule Ω. Thus, to implement the LA, a manufacturer must always implement Algorithms 2 to 4 and, additionally, Algorithm 5 if the LA generates the final schedule. In this paper, we fully described the scenario where the local scheduler provides the sequences seq and the final schedule is decided by the LA.

Note that the SA generates a global schedule for the full production line in the final part of Algorithm 1. This is primarily done to enable the proof of soundness given in the next subsection, Section 5.5. The SA does not need to be aware of the full global schedule. The only information that the SA needs are the completion times

of the last operations of each job processed by a module. This suffices to compute the lower bounds s in line 25 of Algorithm 1.

5.5 Proof of soundness

We prove that if Algorithm 1 terminates at line 28, the production-line schedule Ω returned is feasible. After convergence (at line 22), all the modules are aware of their neighbours' constraints. Consequently, it becomes possible to create mutually compatible schedules. A monolithic system can take all the local sequences seq_i, build the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$ (Section 4.3), and obtain its ASAP schedule. However, since we work on a distributed system, we need to generate the schedules of the modules one by one. Therefore, we prove that the schedule obtained by following lines 22 to 27 of Algorithm 1 is the same as the one that is obtained by a monolithic system.

To prove this, we first show that, after convergence, there are no positive cycles in the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$ (Corollary 1). Then, we prove that the lengths of the longest paths are the same between the local graphs $\mathcal{G}_{\mathsf{seq}_i}$ and the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$ for all pairs of nodes in each local graph $\mathcal{G}_{\mathsf{seq}_i}$ (Lemma 2). Finally, we prove that, with these properties, the algorithm generates the same ASAP schedule using the start times s and the local sequences seq_i as the global ASAP schedule of all the sequences seq_i (Lemma 3). Together, these results lead to the main result stated in Theorem 1, namely that the production-line schedule returned by Algorithm 1 is feasible.

Lemma 1 *If, after convergence of Algorithm 1, there is a cycle in the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$ that spans more than one module, where seq refers to all the sequences after the last iteration of the convergence step of Algorithm 1, there exists also a cycle with a length at least as large as the original cycle that spans one module less.*

Proof Let there be such a cycle spanning more than one module. Let $(n_i, n_{i+1}, \ldots, n_{i+j})$ be the modules that the cycle spans. Then, all paths part of the cycle that go through the first module, n_i, have edges (o_1, b, o_2) in the second module n_{i+1}, whose weights are at least the lengths of the longest paths from o_1 to o_2 because of the propagation of bounds explained in Sections 5.2.4 and 5.3.2. By replacing these paths with the corresponding edges in n_{i+1}, an equivalent cycle is obtained with a weight at least that of the original cycle but spanning one module less $(n_{i+1}, \ldots, n_{i+j})$. □

It follows then from (repeatedly using) Lemma 1 that if a positive cycle exists in the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$, then there exists also a positive cycle in the local graph $\mathcal{G}_{\mathsf{seq}_i}$ of some module i, implying that the local scheduling problem for module i is infeasible. Thus, if a positive cycle exists in the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$, Algorithm 1 terminates at line 9, without converging.

Corollary 1 *If Algorithm 1 converges at line 17, then the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$ does not contain a positive cycle.*

A feasible schedule of a re-entrant flow-shop problem τ_i with sequence seq_i exists if and only if there are no positive cycles in the graph $\mathcal{G}_{\mathsf{seq}_i}$ built from the sequences seq_i and the tuple τ_i. If a feasible schedule Ω_i exists, then the ASAP schedule exists and is unique. The ASAP schedule with the lower bound function s used in lines 22 to 27 assigns a starting time $\Omega_i(o)$ to all operations o in O_i taking into account the lower bounds in s. Since both the local and global ASAP schedules are obtained by finding the lengths of longest paths, we prove that these lengths are the same between the local graphs $\mathcal{G}_{\mathsf{seq}_i}$ and the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$.

Lemma 2 *After convergence of Algorithm 1, the length of the longest path between two nodes in each local graph $\mathcal{G}_{\mathsf{seq}_i}$ is the same as the length of the longest path between two equivalent nodes (that is, the nodes representing the same operations of the same module) in the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$.*

Proof We prove the lemma by contradiction. Assume that the length l_{seq_i} of the longest path from o_1 to o_2 in the local graph $\mathcal{G}_{\mathsf{seq}_i}$ is not equal to the length $l^{global}_{\mathsf{seq}}$ of the longest path between the same pair of operations in the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$. If this is the case, there are two possible options:

$\boldsymbol{l_{\mathsf{seq}_i} > l^{global}_{\mathsf{seq}}}$. In this case, because the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$ is built from all the local graphs $\mathcal{G}_{\mathsf{seq}_i}$ and the transfer constraints CT1 and CT2 (Section 4.3), for each edge in $\mathcal{G}_{\mathsf{seq}_i}$ there exists an edge in $\mathcal{G}^{global}_{\mathsf{seq}}$ between the same pair of operations with the same weight. Thus, the longest path from o_1 to o_2 in the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$ is at least as long as the longest path from o_1 to o_2 in the local graph $\mathcal{G}_{\mathsf{seq}_i}$. This contradicts the fact that $l_{\mathsf{seq}_i} > l^{global}_{\mathsf{seq}}$.

$\boldsymbol{l_{\mathsf{seq}_i} < l^{global}_{\mathsf{seq}}}$. If this is the case, there must exist a path with length $l^{global}_{\mathsf{seq}}$ that starts in o_1, leaves n_i and enters it again, possibly multiple times, and finally ends in o_2. At least one of the detours leaving n_i must be longer than the corresponding longest path inside the local graph $\mathcal{G}_{\mathsf{seq}_i}$. Without loss of generality, let operations o'_1 and o'_2 and job indices u and v be such that the detour from o'_1 via $\mathsf{fst}_i(j_u)$ through module n_{i-1} via $\mathsf{fst}_i(j_v)$ to o'_2 is longer than the local longest path from o'_1 to o'_2. The length of the path from $\mathsf{fst}_i(j_u)$ through module n_{i-1} to $\mathsf{fst}_i(j_v)$ is the translated bound $t^{*\text{in,LB}}_{i,k,u,v}$, where k is the last iteration of the main loop of Algorithm 1. Since we assume convergence of Algorithm 1, this translated bound must be equal to the length of the local longest path from $\mathsf{fst}_i(j_u)$ to $\mathsf{fst}_i(j_v)$, $t^{\text{in,LB}}_{i,k,u,v}$. But from the above reasoning, it follows that $t^{*\text{in,LB}}_{i,k,u,v} > t^{\text{in,LB}}_{i,k,u,v}$, giving a contradiction. Considering the output side of the module instead of the input side, following a similar reasoning, we obtain that $t^{*\text{out,LB}}_{i+1,k,u,v} > t^{\text{out,LB}}_{i+1,k,u,v}$, also leading to a contradiction. So, we may conclude that Algorithm 1 has not converged if $l_{\mathsf{seq}_i} < l^{global}_{\mathsf{seq}}$.

Concluding, we reached a contradiction in both scenarios. Thus, proving the lemma. □

We now define the ASAP schedule of the production-line problem P and in Lemma 3 we prove that the schedule obtained after line 27 of Algorithm 1 is the ASAP schedule of the derived production-line problem $P_{\mathsf{MB}^{\text{in}},\mathsf{MB}^{\text{out}}}$ taking into account the bounds MB^{in} and MB^{out} computed in the last iteration k of the main loop of Algorithm 1. $P_{\mathsf{MB}^{\text{in}},\mathsf{MB}^{\text{out}}}$ has the same modules and transfer points as the production

line P, but each module $n_i \in N$ is equipped with flow-shop problem $\tau_{i,k}$ from the last iteration of Algorithm 1 instead of τ_i.

Definition 4 The ASAP schedule of a production-line problem P and the sequences seq_i for all n_i in N is defined as a feasible schedule Ω where the start times of the operations of each machine m in M_i follow the order of the sequence $\mathsf{seq}_i(m)$ (i.e. $\Omega(\mathsf{seq}_i(m)(j_1)) < \Omega(\mathsf{seq}_i(m)(j_2))$ if and only if $j_1 < j_2$), and $\Omega(o)$ is minimal for all o in O.

Lemma 3 *The schedule Ω obtained after lines 22 to 27 of Algorithm 1 is the ASAP schedule for the production line $P_{\mathsf{MB}^{\text{in}},\mathsf{MB}^{\text{out}}}$ using the sequences seq_i and taking into account bounds MB^{in} and MB^{out} for all n_i in N of the last iteration of the main loop of Algorithm 1.*

Proof Because the process of lines 22 to 27 goes through an iterative process from the first module to the last, we prove it by induction. It is important to note that all of the modules have feasible schedules, because of Corollary 1. Furthermore, recall from Sections 3.3 and 4.3 that for every operation o in the production line, the start time $\Omega(o)$ in the ASAP schedule Ω corresponds to the longest path in constraint graph $\mathcal{G}^{global}_{\mathsf{seq}}$ from the first operation of the first job in module n_1 to o.

As a base case, consider the first module n_1 in the production line. Because the lower bounds defined by $\mathtt{s}$ are 0 for all first operations of all jobs performed by module n_1, and because of Lemma 2, the ASAP schedule for all module n_1-operations in the production line $P_{\mathsf{MB}^{\text{in}},\mathsf{MB}^{\text{out}}}$ is determined by the lengths of the longest paths from the first operation of the first job to all the others in the local $\mathcal{G}_{\mathsf{seq}_i}$. This is precisely what method call LA_1`.LASchedule(s)` in line 23 of Algorithm 1 provides. Hence, for all n_1-operations, Ω, as computed in lines 22 to 27 of Algorithm 1, is the ASAP schedule.

For the inductive step, considering module n_i with $i > 1$, assume that the start time of an operation o of the previous module n_{i-1} in de global ASAP schedule, $\Omega(o)$, equals the length of the longest path from the first operation of the first job in the first module to o in the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$. Observe that in line 26 of Algorithm 1 the parameter $\mathtt{s}$ passed to `LASchedule` contains, for the first operation $\mathsf{fst}_i(j)$ of each job j, the length of the longest path from the first operation in the system to that operation $\mathsf{fst}_i(j)$. For all the other operations o in the module n_i, the longest path can be divided into (1) the path that goes from the first operation in the system to a first operation $\mathsf{fst}_i(j)$ in the module, for the first time, and (2) the path that goes from $\mathsf{fst}_i(j)$ to the operation o of module n_i whose longest path we are searching. The length of path (1) is, by induction, the value stored in the parameter $\mathtt{s}$. Thus, the length of the longest path from the first operation of the system to o in the global graph $\mathcal{G}^{global}_{\mathsf{seq}}$ is, using Lemma 2 again, found by finding the largest value among all j of the sum of $\mathtt{s}(\mathsf{fst}_i(j))$ and the length of all the longest path from $\mathsf{fst}_i(j)$ to o in the local graph $\mathcal{G}_{\mathsf{seq}_i}$. This is precisely what the method `LASchedule()` in line 26 of Algorithm 1 does. Hence, also for module n_i, we may conclude that the start time $\Omega(o)$ in schedule Ω corresponds to the longest path in $\mathcal{G}^{global}_{\mathsf{seq}}$ from the first operation in the system to o. Thus, also for module n_i we may conclude that Ω, as computed in lines 22 to 27 of Algorithm 1, provides the ASAP schedule for production line $P_{\mathsf{MB}^{\text{in}},\mathsf{MB}^{\text{out}}}$. □

Theorem 1 *The schedule Ω obtained after lines 22 to 27 of Algorithm 1 is a feasible schedule for the production-line problem P.*

Proof The production-line problem $P_{\mathsf{MB}^{\text{in}},\mathsf{MB}^{\text{out}}}$ is derived from the original problem P by adding constraints. Thus, any feasible solution of the derived problem is a feasible solution of the original problem. Thus, the desired result follows immediately from Lemma 3. □

5.6 Quantifying the performance of modular schedulers

In the following, we assess the performance of our distributed scheduling framework. In this subsection, we establish the metrics used for evaluation and comparison of modular scheduling solutions. Our focus is on three metrics: relative makespan, schedulability ratio, and execution time.

Definition 5 ((Relative) makespan) Given a production line P and a feasible schedule Ω, the makespan of the schedule is the time between the start of the first operation in the system and the completion of the last operation in the system:

$$\mathsf{makespan}(P,\Omega) = \max_{o\in O}(\Omega(o) + \mathsf{prc}(o)) - \min_{o\in O}\Omega(o)$$

Given a base algorithm and a comparison algorithm, with makespans m_B and m_C respectively, the relative makespan between the base and the comparison is $\mathrm{m}_C/\mathrm{m}_B$.

Definition 6 (Schedulability ratio) Given a set of problems $\mathcal{P}$ to evaluate, the schedulability ratio is the ratio between the number of problems for which a feasible solution has been found and the total number of problems to solve.

$$\mathsf{schedulability\_ratio}(\mathcal{P}) = \frac{|\mathsf{solved}(\mathcal{P})|}{|\mathcal{P}|}$$

Definition 7 (Scheduler execution time) The execution time of a scheduler on a given scheduling problem is the total amount of CPU time that the scheduler requires to execute. For a modular scheduler, this includes the CPU time that each local scheduler requires to solve all the local problems.

5.7 Some concluding remarks

We designed a distributed scheduling method that propagates constraints between multiple schedulers in a production line. The structure is a hierarchical MAS architecture with the SA acting as a coordinator between LAs. We explained how the constraints are obtained, translated, propagated and incorporated in each module. We have shown that convergence of the distributed algorithm provides a globally feasible schedule for the full production line.

The quality of the global schedules generated depends both on the MAS design and on the quality of the local schedules. As long as it is possible to obtain bounds for subsequent jobs and sequences of operations, each module can use schedulers with different heuristics, such as for instance BHCS or MNEH discussed in Section 2, and combine their strengths, allowing scheduling flexibility and design freedom in the design of production lines. We experimentally investigate schedule quality in Section 7. The next section first provides some background on two important design decisions

in the proposed MAS, namely the bounds-propagation strategy and the decision to include a limit on the number of iterations of the main loop of Algorithm 1.

The presented approach can be applied to non-cyclic sequential module patterns with fixed-output order of jobs across modules. One could consider more general topologies, such as parallel modules. The SA would need to be aware of the destination of each job at every transfer point, and of the order of the jobs in the source and destination modules. This would allow the SA to send the bounds to the appropriate modules. Relaxing the fixed-output-order constraint (C6) would require updating the consensus mechanism. In such an updated mechanism, each module could propose a job order, and the SA could heuristically choose one that is globally near-optimal. Such extensions to other topologies and relaxing the ordering constraint are interesting topics for further work.

6 Design decisions

6.1 Bounds-propagation strategy

In Algorithm 1, the SA gathers and propagates bounds to other modules using a broadcast propagation strategy, where in each iteration of the propagation all constraints are communicated to all other modules. Moreover, the SA initially only propagates lower bounds; upper bounds are only propagated after initial convergence on the lower bounds. These choices impact the speed of convergence and the quality of the obtained solutions. It is important to find a good balance between convergence speed and solution quality. Related to this is the choice for the iteration limit $k^{\max}$. The following subsections discuss the propagation of lower and upper bounds, the propagation strategy itself, and the setting of $k^{\max}$ in some more detail.

6.1.1 Lower and upper bounds propagation

As previously explained, the convergence of Algorithm 1 follows a two-step process: (1) propagate lower bounds only and (2) propagate both lower and upper bounds. Propagating both lower and upper bounds immediately reduces the space of feasible solutions for the involved LAs too fast, as every LA is limited to the space defined by those bounds. Thus, we defer the propagation of upper bounds until lower-bound convergence has been achieved. One may in fact wonder whether it is needed to propagate upper bounds at all? We are interested in ASAP schedules, so it is interesting to consider the option to only propagate lower bounds. But it turns out that propagation of upper bounds is required as without convergence on upper bounds as well, it is possible for Algorithm 1 to produce an infeasible schedule. This is shown in Example 6.1.

Example 6.1 In this example, we show that without propagating upper bounds, it is possible for Algorithm 1 to terminate but produce local schedules that violate the transfer constraints.

We define a production line with three modules n_1, n_2, and n_3, and machines $M_1 = \{m_1\}$, $M_2 = \{m_2, m_3\}$, and $M_3 = \{m_4\}$. We schedule three jobs $J = \{j_1, j_2, j_3\}$. Each job j_u has the following operations in each module: $\mathsf{ops}_1(j_u) = \{o_{u,1}\}$, $\mathsf{ops}_2(j_u) = \{o_{u,1}, o_{u,2}\}$, and $\mathsf{ops}_3(j_u) = \{o_{u,1}\}$. The mapping of operations for each job j_u is $\mathsf{mch}_1\big(o_{u,1}\big) = m_1$,

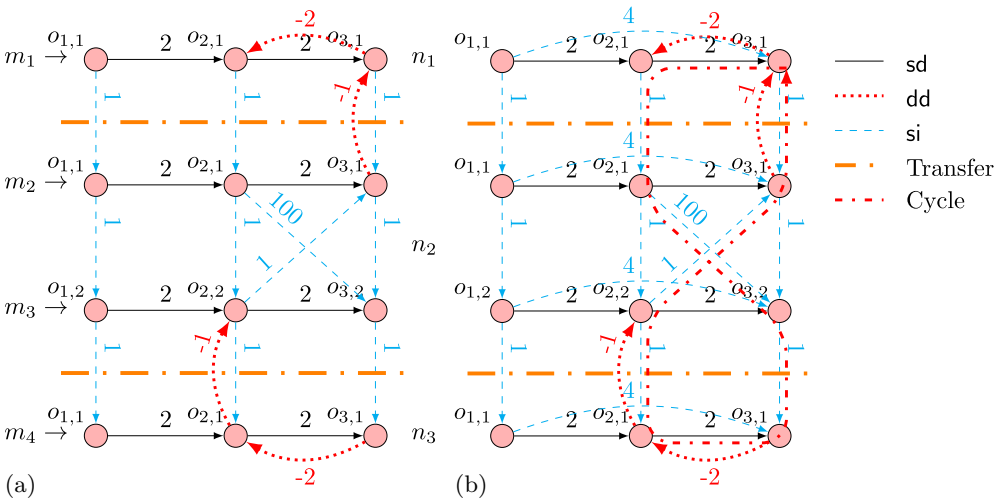

Figure 8: Constraint graphs for Example 6.1. (a) shows a constraint graph of the input problem with trivial sequences and (b) shows the graph after convergence. The red dashed and dotted line in (b) show the positive cycle that exists after termination and the horizontal edges from job j_1 to job j_3 are the added bounds.

$\mathsf{mch}_2\big(o_{u,1}\big) = m_2$, $\mathsf{mch}_2\big(o_{u,2}\big) = m_3$, and $\mathsf{mch}_3\big(o_{u,1}\big) = m_4$. The processing time of all operations is 1, i.e., for all o in O_1, O_2 and O_3, $\mathsf{prc}(o) = 1$. The sequence-dependent setup time for all machines is 1, i.e., for every pair of operations o_1 and o_2 such that $\mathsf{mch}_i(o_1) = \mathsf{mch}_i(o_2)$, we have $\mathsf{sd}_i(o_1, o_2) = 1$.

We add the sequence-independent setup times $\mathsf{si}_2\big(o_{2,1}, o_{3,2}\big) = 99$ and $\mathsf{si}_2\big(o_{2,2}, o_{3,1}\big) = 0$, and the due dates $\mathsf{dd}_1\big(o_{2,1}, o_{3,1}\big) = \mathsf{dd}_3\big(o_{2,1}, o_{3,1}\big) = 1$. For the transfer points T_1 and T_2, we set the transfer setup time to $\mathsf{ts}_1(j_u) = \mathsf{ts}_2(j_u) = 0$ for all j_u in J and the transfer due dates $\mathsf{td}_1(j_3) = \mathsf{td}_2(j_2) = 0$. A constraint-graph representation of these constraints can be seen in Figure 8a, assuming machine sequences for all machines that process all operations of all jobs in order of the job index. These are in fact the (trivial) initial machine sequences that the LAs come up with when executing Algorithm 1, using BHCS (van Pinxten et al., 2017) as a local scheduler.

When scheduling the full production line of this example with Algorithm 1 *without propagating upper bounds*, the algorithm terminates after two iterations of the main loop. The machine sequences do not change in the process. The only extra constraints added after termination are sequence-independent setup times $\mathsf{si}_i\big(o_{1,i}, o_{3,i}\big) = 3$ from each operation of j_1 to the same operation of j_3, shown in Figure 8b.

However, a positive cycle exists in the global graph of Figure 8b, as indicated by the red dashed and dotted line. Given that the graph has a positive cycle, no feasible global schedule for the full production line exists. The positive cycle includes the sequence-independent setup constraint between operations $o_{2,1}$ and $o_{3,2}$. This constraint requires a big gap between the start of processing job j_2 and completing job j_3 in module n_2. But such a gap is not possible in the other two modules, because of the due-date constraints in those modules in combination with the transfer due dates. This large setup constraint between $o_{2,1}$ and $o_{3,2}$

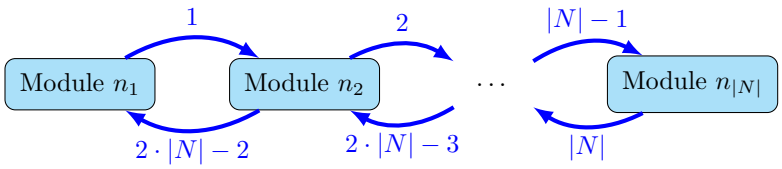

Figure 9: Cocktail-shaker propagation strategy.

is never propagated from module n_2 to the other modules, because it is not part of any path between any pair of input operations or any pair of output operations of n_2.

Returning to Algorithm 1 (without upper bound propagation), this means that generating the full schedule in lines 22 to 27 fails. The last module is not able to generate a schedule, because, given the transfer due date for operation $o_{2,1}$, it is forced to separate operations $o_{2,1}$ and $o_{3,1}$ further than allowed by its local due date.

This issue can be resolved by propagating upper bounds. When running Algorithm 1 as proposed (so, including upper bound propagation), two extra edges $(o_{3,1}, -2, o_{2,1})$, and $(o_{3,2}, -2, o_{2,2})$ are included in the local constraint graph of module n_2. This creates a positive cycle inside module n_2. As a result, Algorithm 1 terminates at line 9 without returning a schedule, in line with the earlier observation that no feasible schedule exists.

6.1.2 Propagation strategy

As explained, Algorithm 1 propagates bounds to other modules using a broadcast propagation strategy. A potential risk of this broadcast strategy is that the bounds of modules further apart in the production line might not be compatible with each other. It is interesting to compare the broadcast strategy with a strategy that strives for global compatibility of constraints during constraint propagation in each iteration. We designed such an alternative strategy, dubbed the *cocktail-shaker* propagation strategy, depicted in Figure 9. Here, the SA collects the bounds of module n_1 and propagates them to module n_2. Module n_2 updates its local scheduling problem with the received bounds and produces its own bounds, that the SA then propagates to module n_3, and so on. This process continues until the last module. Then, the process is reversed, starting propagation from the last module, propagating constraints one module at a time till the first module is reached again. The complete propagation process from the first to the last module and back constitutes one iteration in Algorithm 1. This cocktail-shaker propagation, replacing line 7 to line 14 in Algorithm 1, repeats until consensus is reached, i.e., as before, until the bounds are the same as the bounds of the previous iteration. We compare the broadcast and cocktail-shaker strategies in our experimental evaluation.

6.1.3 Setting the iteration limit $k^{\max}$

The purpose of the $k^{\max}$ limit is to stop the scheduling when non-convergence is suspected. A value too small may prematurely stop Algorithm 1, reducing the overall quality. A lower bound on the minimum number of expected iterations (for the broadcast strategy) is $4(|N|-1)$ as it requires $2(|N|-1)$ iterations for information to

propagate from the first to the last module and back, and there are two convergence processes. For the cocktail-shaker propagation strategy, the lower bound for $k^{\max}$ is 2, since one iteration already propagates constraints back and forth through the full production line. The actual value must be above the minimum number of expected iterations but it must also fit within the time budget of the application. Given that the time per iteration is relatively stable (from our experimental evaluation), a good estimate can be made from experiments with application-specific problem instances, observing the needed number of iterations. The value can also be adjusted online if all (or most) instances converge within an observed number of iterations.

6.2 Termination analysis

Given that Algorithm 1 propagates both lower and upper bounds between modules, it may seem that eventually the algorithm must converge, either because a schedule is found or because the bounds cross each other. In the latter case, the algorithm must terminate because a lower bound that is higher than an upper bound creates an infeasible problem. However, without the upper limit $k^{\max}$ on the main loop of Algorithm 1, it is possible to craft a counterexample that does not terminate by alternating between different sequences in the main loop of the algorithm. Example 6.2 below shows this counterexample. Note that, since the number of possible scheduling sequences per module is finite, such a non-terminating scenario could be mitigated by disallowing the reuse of already considered and discarded sequences in each of the LAs. However, that would require considerably more analysis effort to be performed in the LAs. We therefore opted not to do this.

Example 6.2 In this example, we show how it is possible that Algorithm 1 does not terminate if there is no hard limit on the number of iterations.

We define a production line with three modules n_1, n_2, and n_3 and machines $M_1 = \{m_1\}$, $M_2 = \{m_2\}$, and $M_3 = \{m_3\}$. We schedule three jobs $J = \{j_1, j_2, j_3\}$. Each job j_u has the following operations in each module: $\mathsf{ops}_1(j_u) = \{o_{u,1}\}$, $\mathsf{ops}_2(j_u) = \{o_{u,1}, o_{u,2}\}$, and $\mathsf{ops}_3(j_u) = \{o_{u,1}\}$. The mapping of operations for each job j_u is the trivial one, $\mathsf{mch}_1\big(o_{u,1}\big) = m_1$, $\mathsf{mch}_2\big(o_{u,j}\big) = m_2$, and $\mathsf{mch}_3\big(o_{u,1}\big) = m_3$. The processing time of all operations is 1. The sequence-dependent setup times for all machines are $\mathsf{sd}_i(o_1, o_2) = 1$ for all o_1 and o_2 with $\mathsf{mch}_i(o_1) = \mathsf{mch}_i(o_2)$ except $\mathsf{sd}_2\big(o_{3,1}, o_{2,2}\big) = 99$. We add the due dates $\mathsf{dd}_2\big(o_{1,1}, o_{1,2}\big) = 1$, and $\mathsf{dd}_2\big(o_{2,1}, o_{3,1}\big) = 3$. For the transfer points T_1 and T_2, we set all transfer setup times and due dates to 0, i.e., $\mathsf{ts}_1(j_u) = \mathsf{ts}_2(j_u) = 0$ and $\mathsf{td}_1(j_u) = \mathsf{td}_2(j_u) = 0$ for all j_u in J. Figure 10 shows a constraint-graph representation of the production line, assuming the trivial machine sequences for modules n_1 and n_3, and two possible sequences for module n_2.

In this setup, module n_2 keeps alternating between the two sequences and thus the bounds exchanged between modules keep changing. The values of these bounds are shown in Table 4. Assume, towards a counterexample, that, in the first iteration, module n_2 chooses scheduling sequence A, shown in Figure 10a. Note that sequence A is not optimal. Sequence B shown in Figure 10b is the optimal sequence. So, a normal scheduler would likely not select sequence A. We chose this sequence for illustration purposes, to obtain a small counterexample. Similar behaviours can be achieved with larger examples and schedulers looking for optimality.

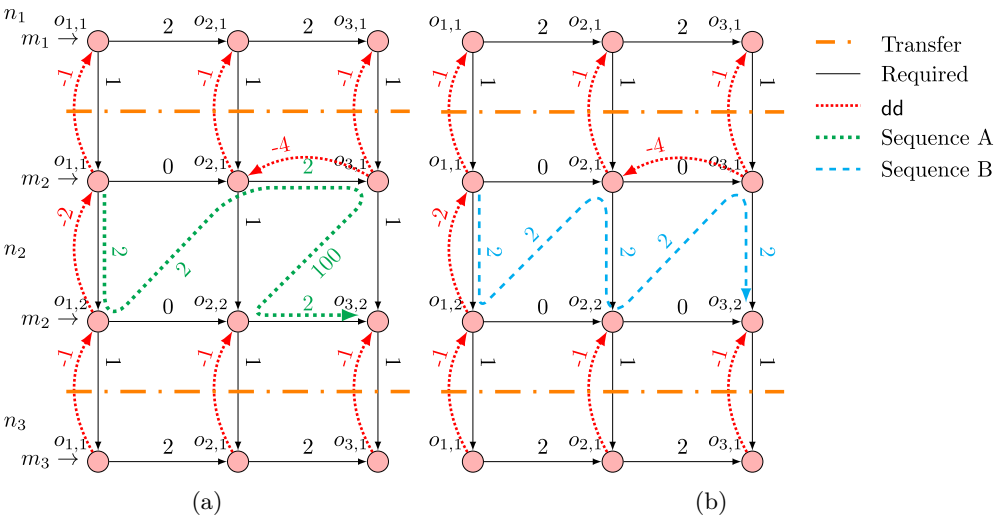

Figure 10: Constraint graph of Example 6.2 with two possible machine sequences. Sequence A is shown in (a) and sequence B in (b)

Iteration (k)		1	2	3	4	5	6	7
n_i	Sequence	A	B	A	B	A	B	A
n_1	$t^{\text{out,LB}}_{1,k,1,2}$	2	**4**	**104**	104	**206**	206	**308**
	$t^{\text{out,LB}}_{1,k,1,3}$	4	**6**	**108**	108	**210**	210	**312**
	$t^{\text{out,LB}}_{1,k,2,3}$	2	2	**4**	4	4	4	4
n_2	$t^{\text{in,LB}}_{2,k,1,2}$	4	**104**	104	**206**	206	**308**	308
	$t^{\text{in,LB}}_{2,k,1,3}$	6	**108**	108	**210**	210	**312**	312
	$t^{\text{in,LB}}_{2,k,2,3}$	2	**4**	4	4	4	4	4
	$t^{\text{out,LB}}_{2,k,1,2}$	104	104	**206**	206	**308**	308	**410**
	$t^{\text{out,LB}}_{2,k,1,3}$	106	**108**	**210**	210	**312**	312	**414**
	$t^{\text{out,LB}}_{2,k,2,3}$	2	**4**	4	4	4	4	4
n_3	$t^{\text{in,LB}}_{3,k,1,2}$	2	**104**	104	**206**	206	**308**	308
	$t^{\text{in,LB}}_{3,k,1,3}$	4	**106**	**108**	**210**	210	**312**	312
	$t^{\text{in,LB}}_{3,k,2,3}$	2	2	**4**	4	4	4	4

Table 4: Values of the bounds of each module, as difference between start times for input bounds and completion times for output bounds, at the end of every iteration of Algorithm 1. The table heading shows the iteration count and the sequence used.

The purpose of the due dates $\mathsf{dd}_2\big(o_{1,1}, o_{1,2}\big) = 1$ and $\mathsf{dd}_2\big(o_{2,1}, o_{3,1}\big) = 3$ is to create a mutual dependency between $t^{\text{in,LB}}_{2,k,1,2}$ and $t^{\text{out,LB}}_{2,k,1,2}$ when using sequences A and B. In iterations with sequence A, the bound $t^{\text{in,LB}}_{2,k,1,2}$ gets "amplified" by the setup time $\mathsf{si}_2\big(o_{3,1}, o_{1,2}\big) = 99$ and $t^{\text{out,LB}}_{2,k+1,1,2} \approx t^{\text{in,LB}}_{2,k,1,2} + 100$. After the first two iterations, the longest path from $o_{1,2}$ to $o_{2,2}$ with sequence A is the sequence of nodes $\langle o_{1,2}, o_{1,1}, o_{2,1}, o_{3,1}, o_{2,2}\rangle$ (because the weight w of the edge $(o_{1,1}, w, o_{2,1})$ increases as a consequence of adding the $t^{\text{in,LB}}_{2,k,1,2}$ bound). In iterations with sequence B, the value of the bound $t^{\text{out,LB}}_{2,k,1,2}$ is propagated back to $t^{\text{in,LB}}_{2,k+1,1,2}$ because the longest path from $o_{1,1}$ to $o_{2,1}$ with sequence B after the first two iterations involves the nodes $\langle o_{1,1}, o_{1,2}, o_{2,2}, o_{3,1}, o_{2,1}\rangle$ (because the weight w of the edge $(o_{1,2}, w, o_{2,2})$ increases as a consequence of adding $t^{\text{out,LB}}_{2,k,1,2}$). This process continues indefinitely.

Table 4 shows the bound values during the first seven iterations. *Iteration 1*: the scheduler of module n_2 selects sequence A. The bound $t^{\text{out,LB}}_{2,1,1,2}$ from $o_{1,2}$ to $o_{2,2}$ in n_2 is 104 (from completion time of $o_{1,2}$ to completion time of $o_{2,2}$) and it is added as the setup time $\mathsf{si}_{2,2}\big(o_{1,2}, o_{2,2}\big) = 103$ (subtracting the processing time $\mathsf{prc}_2\big(o_{1,1}\big) = 1$). *Iteration 2*: the scheduler of module n_2 chooses sequence B. With this sequence and the added setup time $\mathsf{si}_{2,2}\big(o_{1,2}, o_{2,2}\big) = 103$ the input bound $t^{\text{in,LB}}_{2,2,1,2}$ has the value 104 and gets added as the setup time $\mathsf{si}_{2,3}\big(o_{1,1}, o_{2,1}\big) = 103$. *Iteration 3*: the scheduler of module n_3 chooses sequence A. Now the output bound $t^{\text{out,LB}}_{2,3,1,2}$ has the value 206. At this point, the output bound has already been amplified once so the following iterations just repeat this process without termination.

7 Experimental evaluation

This section describes the experimental evaluation of our MAS approach. We integrate two state-of-the-art schedulers for re-entrant flow shops (BHCS (van Pinxten et al., 2017) and MNEH (Jeong & Kim, 2014)) into the MAS. The evaluation focuses on scalability and schedule quality, using the relative makespan, schedulability ratio, and scheduler runtime metrics defined in Section 5.6, as well as reusability of local schedulers. We use a monolithic, exact scheduler to quantify the quality of our modular solution. The monolithic scheduler has complete knowledge, mimicking an ideal scenario, but cannot be applied for modular setups. We implemented the exact monolithic scheduler (MON) using constraint programming (CP). Typically, CP provides the most competitive solutions for this type of scheduling problems (Laborie, Rogerie, Shaw, & Vilím, 2018). To implement the CP solution, we used IBM ILOG CP Optimizer 22.1.1.0 (IBM Corporation, 2023).

7.1 Experiment design

We designed synthetic experiments for both homogeneous and heterogeneous production lines, drawing inspiration from an industrial case. Each type of production line includes eight different scenarios. Homogeneous lines consist of multiple instances of the same module, while heterogeneous lines feature modules with varying timings. Across scenarios, jobs vary in processing times, setup times, and due dates. For each type and scenario, we generate flow-shop problems with 5, 10, 20, 30, 40, 60, 80, 100, 150, 200, 250, and 300 jobs, and 2 to 10 modules in a production line, giving a total of

8640 problem instances. The processing times, setup times, and due-date values are based on the experiments by van Pinxten et al. (2017).

We implemented both the broadcast strategy presented in Algorithm 1 and the cocktail-shaker propagation strategy discussed in Section 6.1.2 as a C++ program. In this program, we also implemented both the BHCS and MNEH local schedulers of van Pinxten et al. (2017) and Jeong and Kim (2014), respectively. Additionally, as an evaluation of a fallback mechanism, we also implemented a third local scheduler, referred to as the "simple" scheduler, that generates naive schedules that are always feasible in our specific problem case. The simple scheduler schedules all the re-entrant operations of a job before scheduling the re-entrant operations of the next job. Thus, it never interleaves operations of multiple jobs. This is always possible because we do not have due dates between jobs; we only have due dates within jobs. For the sake of comparison between variants of our MAS approach, we implemented the bound $k^{\max}$ in Algorithm 1 as a 10-minute CPU time limit.

As mentioned, our evaluation focuses on schedule quality and scalability. Scalability is investigated in terms of the number of modules in a production line and the number of jobs to be scheduled. As quality metric, we consider the schedule makespan (Definition 5). We use the monolithic scheduler MON, which has complete knowledge of the production-line setup and jobs to be scheduled, as a reference. Ideally, MON can provide optimal schedules, given enough time. We also evaluate the schedulability ratio (Definition 6) of the various setups. All problem instances are feasible by construction. So, the schedulability ratio provides insight in how often a backup scenario (i.e. falling back to the naive schedule generated with the simple local schedulers) is needed because the MAS cannot find a solution.

To obtain a broad overview of the behaviour of our MAS, we ran both our MAS and MON with an execution time limit of 10 minutes of CPU time per instance, i.e., a bound on the scheduler execution time as defined in Definition 7. Considering these results, we then selected 1080 instances from the heterogeneous production lines and we ran them with a time limit of 60 minutes of CPU time per instance. We used the increased time to find more cases of optimal solutions with the exact scheduler MON and to test whether an increase of $k^{\max}$ in our MAS approach would allow more instances to reach consensus with the MAS. To compare the effect of different local schedulers, we also evaluated these 1080 instances with BHCS, MNEH, and combinations of these two schedulers as local schedulers and a 60 minutes CPU time limit per instance.

7.2 Experiments results

This section first discusses the aggregated results of the 10-minute experiments to obtain a high-level overview of the behaviour of our MAS approach. Section 7.2.2 then investigates the scalability of our approach in terms of the number of modules and the number of jobs. We discuss the 60-minute experiments in Section 7.2.3 to understand the effect of having a larger time budget for scheduling challenging cases. Finally, we compare different local schedulers in Section 7.2.4.

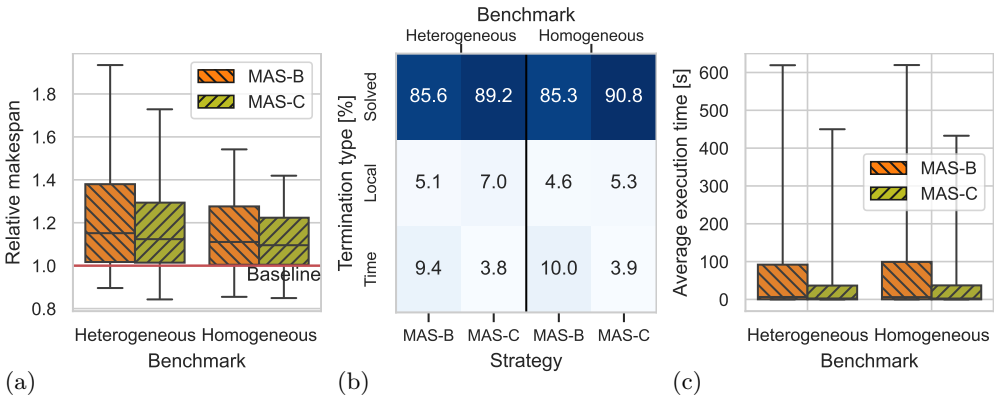

Figure 11: Evaluation results. (a) Relative makespan (lower is better) by benchmark type of our MAS with cocktail (MAS-C) and broadcast (MAS-B) propagation strategies, compared to MON. (b) Percentage of instances that were solved (Solved), ran into a local infeasibility (Local), or ran out of time (Time) by benchmark type and algorithm. (c) Average algorithm execution time by benchmark type. For both box plots, the whiskers represent the 5^{th} and 95^{th} percentile of all data.

7.2.1 Overview

Figure 11 presents the aggregated results of all the 10-minute experiments. MAS-B and MAS-C refer to our MAS with BHCS as the local scheduler using the broadcast and cocktail constraint propagation strategies, respectively. We used BHCS in this first set of experiments because of its good trade-off between efficiency and quality, enabling the testing of many scenarios.

First, we compare the relative makespan (see Definition 5) of MAS-B and MAS-C using MON as a baseline in Figure 11a. MON is expected to give near-optimal results. (Optimality is not guaranteed given the imposed run-time limit.) The relative makespan gives an indication of how good the solutions of the MAS are. The average relative makespan of MAS-C is 1.18 for heterogeneous production lines, 1.12 for homogeneous ones, and 1.15 overall. With MAS-B, it is 1.24, 1.15, and 1.20, respectively. These results are significantly better than the relative makespan using the simple local scheduler, which is 4.15 on average (irrespective of the constraint-propagation strategy). This confirms the importance of the local scheduler in the quality of the global schedules. The schedule quality is determined by the combination of the MAS constraint-propagation strategy and the used local scheduler.

As mentioned, it is also interesting to consider the schedulability ratio (Definition 6). We compare the two propagation strategies of our MAS with the BHCS local scheduler across the two benchmarks in Figure 11b. There is no significant difference between the benchmarks. Instances that could not be scheduled by MAS-C became locally infeasible (6.1%), or reached the time limit $k^{\max}$ (3.9%). Similarly, for MAS-B, 4.8% of the instances became locally infeasible and 9.7% reached the time limit. Figure 11b shows the breakdown over the two benchmark types. The differences in local infeasibility and time-outs between both propagation strategies are interesting.

MAS-C explicitly tries to propagate globally compatible constraints by going forward and backward through the modules in each iteration; MAS-B propagates constraints more aggressively without global alignment. The latter may limit scheduling freedom too much, in some cases leading to local infeasibility but often also leading to continued iterations until a time-out occurs. With the simple local backup scheduler, both MAS-C and MAS-B generated, as expected, a feasible schedule for all instances.

Finally, we evaluate the average execution time of MAS-B and MAS-C. Figure 11c shows a box plot of the execution times. In general, the execution time of MAS-C is lower than the execution time of MAS-B. This is due to the smarter propagation strategy, leading to local schedulers creating more compatible schedules in the first iterations. While the average and the median execution times are quite low, there are some instances that take a significant amount of time. As said previously, 3.9% of the instances for MAS-C and 9.7% for MAS-B reach the $k^{\max}$ limit. The size of the input problem has a strong impact on the execution time. This is further investigated in the next subsection.

7.2.2 Scalability

Figure 12 illustrates the behaviour of our MAS across different numbers of jobs and modules. In terms of relative makespan (Figure 12a), the difference of MAS-C (which outperforms MAS-B) against the near-optimal monolithic solution becomes less significant as the number of modules increases. Increasing the number of jobs also seems to lower the gap, but less significantly. This can be related to the fact that MON is unable to find near-optimal solutions in the allocated time. This is further investigated in Section 7.2.3 with the increased time budget of 60 minutes.

Regarding the number of iterations required by Algorithm 1 to converge, Figure 12b, shows the iterations required by MAS-C and MAS-B. The difference in number of iterations between the two propagation strategies shows the effectiveness of MAS-C by only scheduling a module after it has received the constraints from its neighbour. In contrast, MAS-B requires a lot of iterations before the constraints of each module have reached all the other ones. However, note that a MAS-C-iteration covers a full forward and backward traversal of all modules, whereas a MAS-B-iteration exchanges bounds between all neighbouring modules only once. Thus, one iteration of MAS-B requires local scheduling $|N|$ times while one iteration of MAS-C requires local scheduling $2|N|$ times. For instances with more than 150 jobs and more than 5 modules, the number of iterations required by MAS-B goes down. This is caused by MAS-B reaching the maximum execution time of 10 minutes (see Figure 12c). Thus, because the time per iteration is quite stable, only the instances that can be solved with a smaller number of iterations are solved within the execution time limit. Increasing the number of jobs or modules essentially has a linear effect on the execution time if solutions can be found within the time budget.

Figure 12d shows the schedulability ratio while varying the numbers of jobs and modules. In general, as the number of modules increases, the schedulability ratio goes down. This can be caused by incompatibilities arising from local scheduling decisions as more modules are involved in a production line. The same applies for an increased number of jobs. For MAS-B, part of the effect can be explained by reaching the

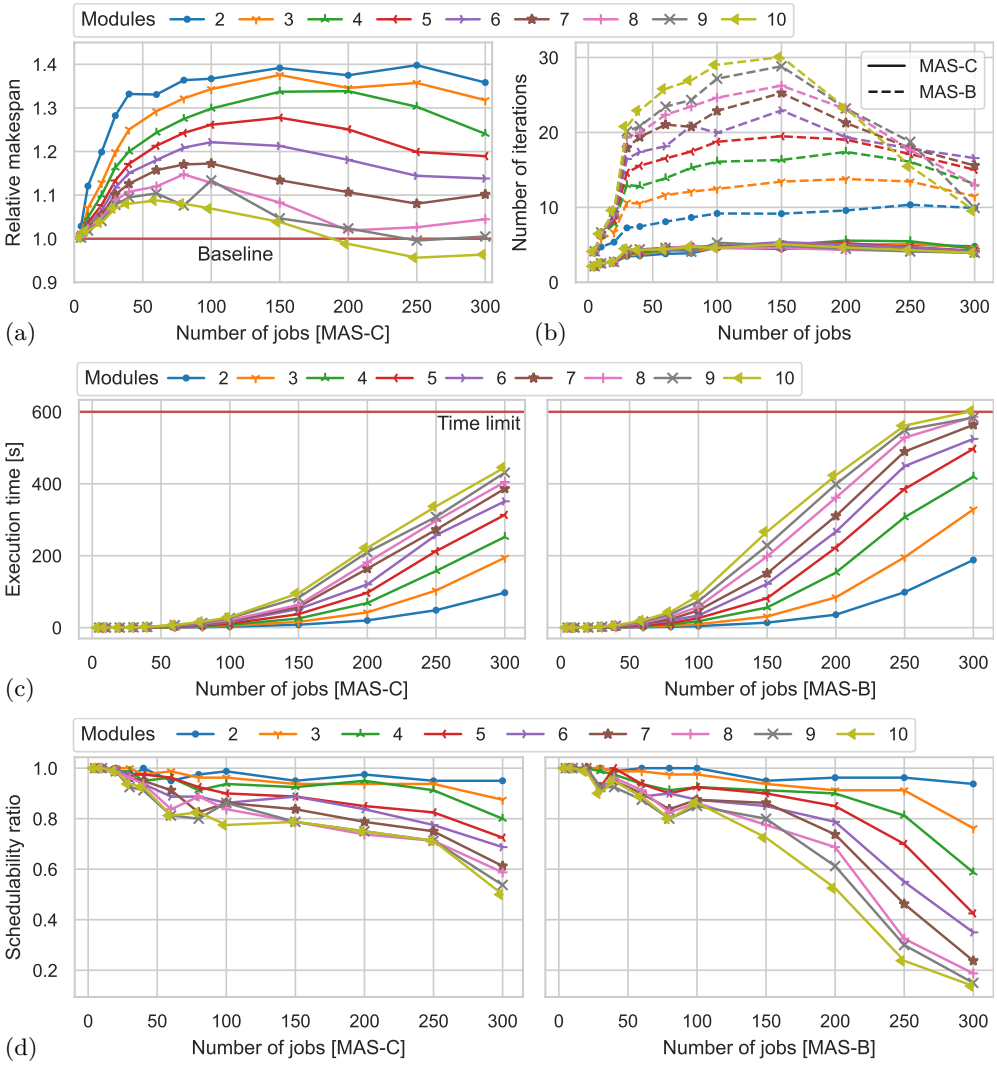

Figure 12: Scalability results while varying the numbers of jobs and modules. (a) Relative makespan (lower is better) of MAS-C. (b) Number of iterations comparing MAS-C and MAS-B. (c) Execution time in seconds of both MAS-C (left) and MAS-B (right). (d) Schedulability ratio of MAS-C (left) and MAS-B (right).

$k^{\max}$ limit for the larger instances. Overall, MAS-C outperforms MAS-B because its propagation strategy tries to ensure global compatibility of the exchanged bounds.

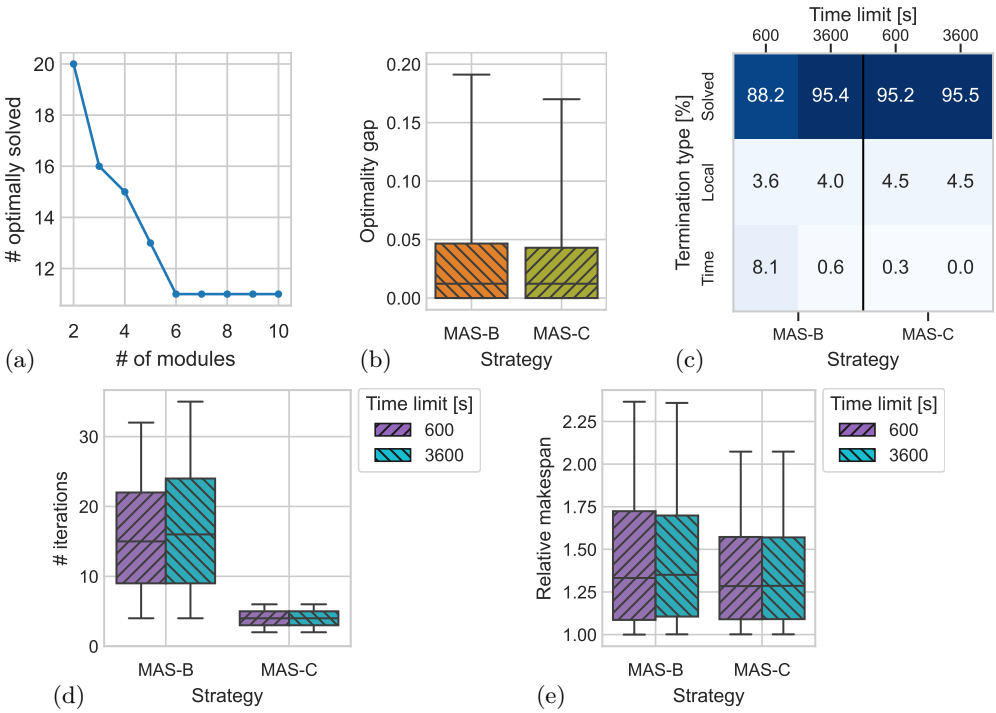

Figure 13: (a) Number of optimal solutions found with MON while varying the number of modules. (b) Box plot of the optimality gap (lower is better) between MAS-B, MAS-C, and the instance's optimal solution for solutions known to be optimal. (c) Percentage of instances that were solved (Solved), ran into local infeasibility (Local), or ran out of time (Time) by propagation strategy and time limit. (d) Number of iterations required to converge for both MAS-B and MAS-C by time limit. (e) Relative makespan of MAS-B and MAS-C by time limit when compared to the solutions of MON using a time limit of 60 minutes.

7.2.3 Quality analysis

Next, we evaluate the differences between known optimal solutions and the solutions provided by our MAS, as well as the effects of a larger $k^{\max}$ limit. We evaluated a subset of 1080 instances of the heterogeneous benchmark with our MAS and time limits of 10 minutes and 60 minutes, using optimal solutions found by MON with a time limit of 60 minutes as reference.

We were able to find 119 optimal instances with MON, all for 10 jobs or less. Their distribution over the number of modules is shown in Figure 13a. Both MAS-C and MAS-B were able to solve the same 42 out of the 119 instances (35%) to optimality. They solved the exact same instances due to the small problem size, since the differences between propagation strategies are not that visible with problems with 10 or less jobs. We also evaluate the optimality gap, which we define as $\frac{\text{makespan} - \text{optimal makespan}}{\text{makespan}}$,

in Figure 13b. There is a small difference between MAS-C and MAS-B but both are relatively close to the optimum (0 gap) in most cases.

We also evaluated the schedulability for the 1080 instances when increasing the $k^{\max}$ limit. Figure 13c shows the termination types of each strategy with different time limits (10 and 60 minutes). When increasing the time limit from 10 minutes to 60 minutes, the schedulability ratio for MAS-B increases from 88.2% to 95.4% with only 0.6% reaching the time limit. This increase is caused by MAS-B successfully converging in more instances. Similarly, for MAS-C, it increased slightly from 95.2% to 95.5% without any instances still reaching the time limit. The rest of the instances that could not be scheduled were because the problems became locally infeasible. This effect was expected from the earlier analysis of Figures 12b and 12c. It can now be observed in Figure 13d that the 95-th percentile of the number of iterations is higher with a 60-minutes time budget. A 10-minutes time limit may lead to unschedulability by a false positive of non-convergence.

Finally, we compare in Figure 13e the changes in relative makespan when increasing the iterations limit. For MAS-C, there is no significant difference as almost all of the instances were already solved within the time limit of 10 minutes. For MAS-B, there is a slight visible change caused by more instances being solved. Thus, we can conclude that the time limit of 10 minutes is mostly sufficient for MAS-C but too low for MAS-B.

7.2.4 Reusing local schedulers

Finally, as a way to test the applicability of our method to different local schedulers, we implemented the MNEH scheduling algorithm and used it in our MAS. We created two different scenarios. In one scenario, all the modules in the production line use MNEH as the local scheduler (MAS-MNEH). In the other scenario, we used both BHCS and MNEH in an interleaved manner (MAS-INTL). That is, if one module uses BHCS as the local scheduler, then the following one uses MNEH and vice versa. These are the most adverse conditions for our MAS for testing multiple local schedulers in a single production line. We then combined the results with the scenario from the previous section using only BHCS (MAS-BHCS). All setups use the cocktail constraint propagation, because the earlier experiments show that this strategy outperforms the broadcast strategy. We used the problem instances of Section 7.2.3 for these experiments.

Figure 14 shows the results of our experiments with these local schedulers. The top row shows results for instances up to and including 100 jobs and 10 modules; the bottom row shows results for instances up to and including 60 jobs and 6 modules. MNEH is a slower scheduler than BHCS. Since the goal is to illustrate the reuse of local schedulers and to compare the results for different local schedulers, we focus the discussion on problem instances that can be handled by both local schedulers. The 100 jobs/10 modules instances are the biggest instances that can be evaluated without any the three schedulers (MAS-BHCS,MAS-MNEH, and MAS-INTL) running out of time on the 60-minute time limit (only one MAS-MNEH instance ran out of time). Up to 60 jobs and 6 modules, almost all problem instances can be solved with both local schedulers (without invoking the simple backup scheduler). These cases illustrate the

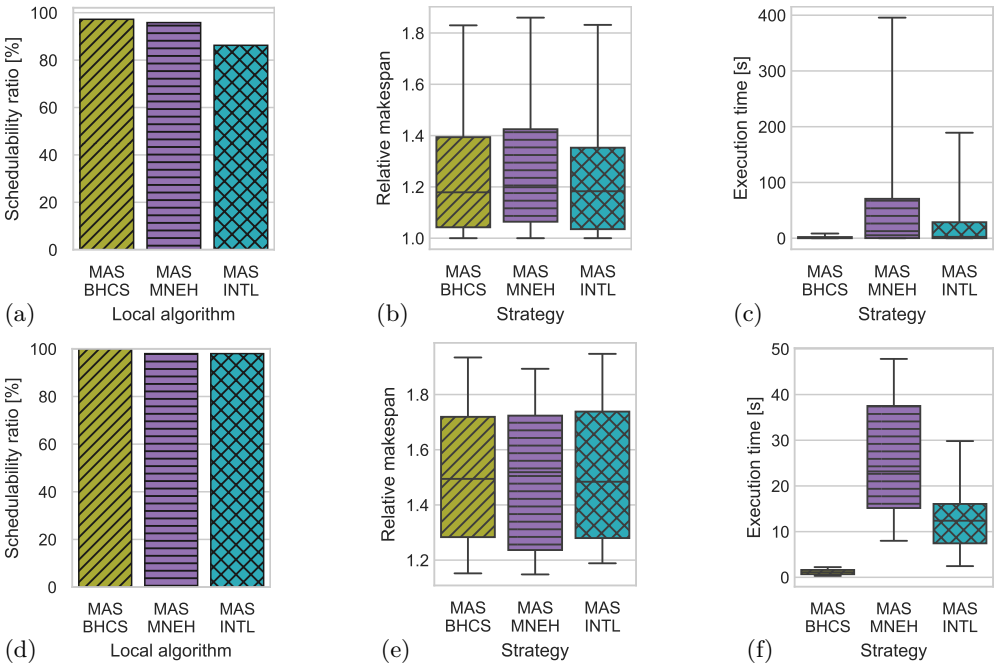

Figure 14: (a), (b), and (c): results with instances up to 100 jobs and 10 modules. (d), (e), and (f): results with instances up to 60 jobs and 6 modules. (a) and (d): schedulability ratio of each scheduler. (b) and (e): relative makespan of each scheduler when compared to MON. (c) and (f): execution time box plot of each scheduler. For the box plots, the whiskers represent the 5^{th} and 95^{th} percentile of all data.

best conditions for the local schedulers to work. In this way, we can make a proper comparison of these local schedulers in scenarios in which they perform the best.

In terms of schedulability, for up to 100 jobs (Figure 14a), MAS-BHCS and MAS-MNEH perform very similarly while the combination of both schedulers in MAS-INTL performs a bit worse. This is caused by the schedules between two neighbouring modules, derived by different local schedulers, not being completely compatible, making it more difficult for the MAS to find consensus. Interestingly, this does not happen for smaller instances (Figure 14d).

Regarding relative makespan when compared to the MON, all schedulers perform very similarly. Interestingly, MAS-MNEH seems to perform slightly worse than the other two schedulers for large instances (Figure 14b) but not for the smaller ones (Figure 14e). Also, the interleaving of MNEH and BHCS in MAS-INTL seems to provide solutions with quality closer to BHCS than to MNEH.

For execution time, MAS-BHCS requires much less execution time while MAS-MNEH requires significant execution time. This is to be expected since BHCS is an online scheduler while MNEH is not. Nevertheless, 98.7% of the instances of MAS-MNEH finished in less than 1 minute for the up to 60 jobs and 6 modules benchmark.

MAS-INTL is in between the other two since half of the modules use BHCS and the other half MNEH.

7.3 Discussion

In the previous three subsections, we evaluated our MAS approach with various propagation strategies and various local schedulers for scheduling modular production lines, using results from a monolithic CP solution as reference. It is important to observe that the latter is not a modular solution and requires complete knowledge of the production line. This is often not feasible when production lines are built from systems of different vendors, as in chip manufacturing and production printing (Traganos et al., 2020; Yang, Chang, & Chou, 2008). In today's practice, such production lines are typically loosely coupled, with buffers between systems of different vendors and no schedule alignment among these systems. In such setups, our modular MAS approach, that allows the integration of different modules and schedulers, may provide an attractive solution. The results show that the MAS provides for heterogeneous setups schedules with on average 15% larger makespan than the near-optimal monolithic scheduler, which is likely to be an improvement, productivity-wise, compared to a loosely coupled production line with buffering.

Considering the runtime, the MAS approach (with the BHCS local scheduler) is competitive as it can schedule large instances of 10 modules and 300 jobs in less than 8 minutes on average (about 1.5 seconds per job). In a modularised production line, the search space of each module is smaller compared to the global search space of a monolithic setup. For schedulers that use Bellman-Ford, such as BHCS, a reduction in the size of the constraint-graph model significantly benefits speed. Furthermore, the MAS is able to leverage existing heuristics for local modules. The scalability of our solution is evident from Figure 12.

When comparing the different constraint propagation strategies, the cocktail propagation strategy is faster, it has a higher schedulability ratio, and it produces schedules with smaller makespans than the broadcast strategy. However, it is important to note that the reported execution times refer to a single-core implementation. In practice, since modules typically have their own compute resources, the broadcast strategy is naturally parallelised, while cocktail cannot be parallelised, because each module evaluates its bounds (`LAObtainBounds`) separately in order. This would reduce the practical execution time of the broadcast strategy by approximately a factor equal to the number of modules, with a constant offset for the SA computations and for communication between the SA and the modules. This speed-up may be interesting for setups where the execution time of the scheduler is critical and it is a good compromise to have faster results at the expense of slightly worse makespan. Propagation strategies may be further customised to meet the needs of a particular system setup.

8 Conclusions

The distributed multi-agent flow-shop scheduling framework presented in this paper enables the coordination of multiple production modules within a tightly coupled production line in which jobs are processed in a fixed, sequential order. It abstracts

the behaviour of a module into bounds on start and completion times of jobs that are shared among the modules. We have proven that the approach is sound, for non-cyclic module patterns with fixed output order, in the sense that the scheduled handovers of work in progress between modules are compatible with the local module schedules. The strength of our MAS lies in the interaction between the SA and LAs. The LAs (Algorithm 2) provide constraints abstracting the space of solutions of a module and the SA (Algorithm 1) translates and propagates the constraints to the appropriate LAs. The design allows integration of local schedulers without modification, making only minimal assumptions on their behaviour. The approach provides near-optimal results when compared to a monolithic exact scheduler (that is not applicable to modular setups). This is the best achievable result given that the modular approach lacks a comprehensive view of the scheduling process. It offers a good trade-off between scheduling speed and schedule quality and it scales to large production lines. The framework has the potential to integrate different schedulers, thereby combining their strengths and harnessing the benefits of modularity. We have shown the successful integration of two state-of-the-art schedulers (BHCS (van Pinxten et al., 2017) and MNEH (Jeong & Kim, 2014)) into our MAS environment. It provides a solution when monolithic scheduling is not feasible (e.g., in production lines integrating equipment from different vendors). Our MAS framework enables factory designers to assemble customized production lines using available modules and their schedulers, provided that each module implements a compatible LA. Once LAs has been implemented, it is straightforward to reconfigure a production line scheduler. Only the configuration and handover times need to be provided to the SA.

Future research should aim to extend the framework to accommodate more types of production lines. Being able to handle assembly production lines would give a lot more freedom on the type of topologies that are possible. This would allow the merging or splitting of products as well as handling parallel modules. Realizing such an approach requires significant changes in the SA to track the flow of jobs in the production facility, as well as modifications to the propagation strategy to make sure that the algorithm terminates and produces a feasible overall schedule. Another interesting extension would be towards covering production lines that do not require a fixed output order of products. Also such an extension would allow many more scenarios to be covered, since there are many production scenarios where the output order is not significant. But also this extension needs adaptation of the system agent. The SA then needs to achieve consensus on both the processing order of jobs on the various modules and the handover times between modules. Again, termination of the approach and soundness of the obtained global schedules will be the main challenges, besides achieving competitive makespans. It would also be interesting to investigate different, richer abstractions for module behaviour, e.g., abstractions that capture all timing constraints between inputs and outputs of a module, instead of only bounds. This would potentially improve schedule quality at the cost of scheduling time. Such extensions would enhance the versatility and applicability of the framework.

Statements and Declarations. The authors have no competing interests to declare that are relevant to the content of this article. This publication is a result of

the project SAM-FMS, Scheduling Adaptive Modular Flexible Manufacturing Systems (with project number 17931), of the research programme Mastering Complexity (MasCot), which is partly financed by the Dutch Research Council (NWO).

Data availability. We provide the source code to generate the experimental data, to run our MAS, and to obtain the experimental results in GitHub (https://github.com/TUE-EE-ES/modular-flow-shop-scheduler) and Zenodo (https://zenodo.org/records/15306150) with DOI https://doi.org/10.5281/zenodo.15306150.

Acronyms

ASAP as soon as possible. 19, 27–30, 32

BHCS bounded heuristic constraint-based scheduler. 5, 6, 31, 33, 37–39, 43–47

CP constraint programming. 37, 45

DFSP distributed flow-shop scheduling problem. 4, 5
DJSP distributed job-shop scheduling problem. 4, 5

FSSP flow-shop scheduling problem. 4, 5

HCS heuristic constraint-based scheduler. 5

JSPT job-shop scheduling problem with transportation resources. 4, 5
JSSP job-shop scheduling problem. 4, 7

LA local agent. 3, 6, 7, 14–21, 23–27, 31–33, 35, 46

MAS multi-agent system. 3, 4, 6, 7, 11, 13, 14, 31, 37–40, 42–47
MAS-B our multi-agent system with the broadcast propagation strategy. 39–43
MAS-BHCS our multi-agent system with the BHCS scheduler. 43, 44
MAS-C our multi-agent system with the cocktail propagation strategy. 39–43
MAS-INTL our multi-agent system interleaving both the BHCS and MNEH schedulers. 43–45
MAS-MNEH our multi-agent system with the MNEH scheduler. 43, 44
MD-BHCS multi-dimensional bounded heuristic constraint-based scheduler. 5, 6
MFL modified FL. 5
MNEH modified NEH. 5, 31, 37, 38, 43–47
MON monolithic scheduler. 37–40, 42, 44

OEM original equipment manufacturer. 7

PLP production-line problem. 11

SA system agent. 3, 6, 14, 15, 17–21, 23–25, 27, 31, 32, 34, 45, 46

WIP work in progress. 4, 6

References

Bagheri Rad, N., & Behnamian, J. (2022, April). Recent trends in distributed production network scheduling problem. *Artificial Intelligence Review*, *55*(4), 2945–2995, https://doi.org/10.1007/s10462-021-10081-5

Balas, E., Simonetti, N., Vazacopoulos, A. (2008, August). Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, *11*(4), 253–262, https://doi.org/10.1007/s10951-008-0067-7

Chan, F.T., Prakash, A., Ma, H., Wong, C. (2012, November). A hybrid Tabu sample-sort simulated annealing approach for solving distributed scheduling problem. *International Journal of Production Research*, *51*(9), 2602–2619, https://doi.org/10.1080/00207543.2012.737948

Emmons, H., & Vairaktarakis, G. (2013). *Flow shop scheduling* (1st ed., Vol. 182). Springer US.

Fazlirad, A., & Brennan, R.W. (2018, August). Multiagent Manufacturing Scheduling: An Updated State of the Art Review. *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)* (pp. 722–729).

Fioretto, F., Pontelli, E., Yeoh, W. (2018, March). Distributed Constraint Optimization Problems and Applications: A Survey. *Journal of Artificial Intelligence Research*, *61*, 623–698, https://doi.org/10.1613/jair.5565

Glavic, M. (2006, May). *Agents and Multi-Agent Systems: A Short Introduction for Power Engineers* (Tech. Rep.). University of Liege, Electrical Engineering and Computer Science Department. Retrieved from https://hdl.handle.net/2268/209564

Gupta, J.N.D., & Stafford, E.F. (2006, March). Flowshop scheduling research after five decades. *European Journal of Operational Research*, *169*(3), 699–711, https://doi.org/10.1016/j.ejor.2005.02.001

IBM Corporation (2023). *IBM® ILOG® CP Optimizer*. Retrieved from https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-cp-optimizer

Jeong, B., & Kim, Y.-D. (2014, July). Minimizing total tardiness in a two-machine re-entrant flowshop with sequence-dependent setup times. *Computers & Operations Research*, *47*, 72–80, https://doi.org/10.1016/j.cor.2014.02.002

Jia, H., Fuh, J., Nee, A., Zhang, Y. (2002, March). Web-based Multi-functional Scheduling System for a Distributed Manufacturing Environment. *Concurrent Engineering*, *10*(1), 27–39, https://doi.org/10.1177/1063293X02010001054

Kouider, A., & Bouzouia, B. (2011, June). Multi-agent job shop scheduling system based on co-operative approach of idle time minimisation. *International Journal of Production Research*, *50*(2), 409–424, https://doi.org/10.1080/00207543.2010.539276

Laborie, P., Rogerie, J., Shaw, P., Vilím, P. (2018, April). IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, *23*, 210–250, https://doi.org/10.1007/s10601-018-9281-x

Lin, B.M.T., & Hwang, F.J. (2011). Total completion time minimization in a 2-stage differentiation flowshop with fixed sequences per job type. *Information Processing Letters*, *111*(5), 208–212, https://doi.org/10.1016/j.ipl.2010.11.021

Liu, J., & Sycara, K.P. (1993, May). Distributed Constraint Satisfaction through Constraint Partition and Coordinated Reaction. *Proceedings of the 12th International Workshop on Distributed AI.* Hidden Valley, PA.

Liu, S.Q., Kozan, E., Masoud, M., Zhang, Y., Chan, F.T. (2018, May). Job shop scheduling with a combination of four buffering constraints. *International Journal of Production Research*, *56*(9), 3274–3293, https://doi.org/10.1080/00207543.2017.1401240

Lohmer, J., & Lasch, R. (2021, April). Production planning and scheduling in multi-factory production networks: a systematic literature review. *International Journal of Production Research*, *59*(7), 2028–2054, https://doi.org/10.1080/00207543.2020.1797207

Maturana, F.P., & Norrie, D.H. (1996, August). Multi-agent Mediator architecture for distributed manufacturing. *Journal of Intelligent Manufacturing*, *7*(4), 257–270, https://doi.org/10.1007/BF00124828

Miyamoto, T., Umeda, T., Takai, S. (2020). Distributed Job Shop Scheduling using Consensus Alternating Direction Method of Multipliers. *IFAC-PapersOnLine*, *53*(2), 10785–10790, https://doi.org/10.1016/j.ifacol.2020.12.2862

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, *37*(4), 754–768, https://doi.org/10.1016/j.cor.2009.06.019

Nouri, H.E., Driss, O.B., Ghédira, K. (2016). A Classification Schema for the Job Shop Scheduling Problem with Transportation Resources: State-of-the-Art Review. R. Silhavy, R. Senkerik, Z.K. Oplatkova, P. Silhavy, & Z. Prokopova (Eds.), *Proceedings of Artificial Intelligence Perspectives in Intelligent Systems* (pp. 1–11). Cham: Springer International Publishing.

Pan, C., Zhou, M., Qiao, Y., Wu, N. (2018, April). Scheduling Cluster Tools in Semiconductor Manufacturing: Recent Advances and Challenges. *IEEE Transactions on Automation Science and Engineering*, *15*(2), 586–601, https://doi.org/10.1109/TASE.2016.2642997

Qin, M., Wang, R., Shi, Z., Liu, L., Shi, L. (2021, January). A Genetic Programming-Based Scheduling Approach for Hybrid Flow Shop With a Batch Processor and Waiting Time Constraint. *IEEE Transactions on Automation Science and Engineering*, *18*(1), 94–105, https://doi.org/10.1109/TASE.2019.2947398

Riezebos, J., Gaalman, G.J.C., Gupta, J.N.D. (1995). Flow shop scheduling with multiple operations and time lags. *Journal of Intelligent Manufacturing*, *6*(2), 105–115, https://doi.org/10.1007/BF00123682

Rifai, A.P., Nguyen, H.-T., Dawal, S.Z.M. (2015). Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. *Applied Soft Computing*, *40*, 42–57, https://doi.org/10.1016/j.asoc.2015.11.034

Sadiku, M.N.O., Ajayi-Majebi, A.J., Adebo, P.O. (2023). Distributed Manufacturing. M.N.O. Sadiku, A.J. Ajayi-Majebi, & P.O. Adebo (Eds.), *Emerging Technologies in Manufacturing* (pp. 175–184). Cham: Springer International Publishing.

Seitz, M., Gehlhoff, F., Cruz Salazar, L.A., Fay, A., Vogel-Heuser, B. (2021, October). Automation platform independent multi-agent system for robust networks of production resources in industry 4.0. *Journal of Intelligent Manufacturing*, *32*(7), 2023–2041, https://doi.org/10.1007/s10845-021-01759-2

Toptal, A., & Sabuncuoglu, I. (2010, September). Distributed scheduling: a review of concepts and applications. *International Journal of Production Research*, *48*(18), 5235–5262, https://doi.org/10.1080/00207540903121065

Traganos, K., Vanderfeesten, I., Grefen, P., Erasmus, J., Gerrits, T., Verhofstad, W. (2020, October). End-to-End Production Process Orchestration for Smart Printing Factories: An Application in Industry. *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)* (pp. 155–164). Eindhoven, Netherlands: IEEE.

van der Tempel, R., van Pinxten, J., Geilen, M., Waqas, U. (2018). A heuristic for variable re-entrant scheduling problems. *Euromicro conference on digital system design (DSD)* (Vol. 2018, pp. 336–341).

van Pinxten, J., Waqas, U., Geilen, M., Basten, T., Somers, L. (2017). Online scheduling of 2-re-entrant flexible manufacturing systems. *ACM Transactions on Embedded Computing Systems*, *16*(5), , https://doi.org/10.1145/3126551

Waqas, U., Geilen, M., Kandelaars, J., Somers, L., Basten, T., Stuijk, S., ... Corporaal, H. (2015). A re-entrant flowshop heuristic for online scheduling of the paper path in a large scale printer. *Design, automation & test in europe conference & exhibition (DATE)* (pp. 573–578). IEEE Conference Publications.

Yang, C., Chang, C.-J., Chou, Y.-C. (2008, January). Selection criteria for equipment vendors by wafer foundry in Taiwan. *Journal of Information and Optimization Sciences*, *29*(1), 115–128, https://doi.org/10.1080/02522667.2008.10699794

Zhang, C., Shi, Z., Huang, Z., Wu, Y., Shi, L. (2017, June). Flow shop scheduling with a batch processor and limited buffer. *International Journal of Production Research*, *55*(11), 3217–3233, https://doi.org/10.1080/00207543.2016.1268730

Zhang, J., & Wang, X. (2016). Multi-agent-based hierarchical collaborative scheduling in re-entrant manufacturing systems. *International Journal of Production Research*, *54*(23), 7043–7059, https://doi.org/10.1080/00207543.2016.1194535

Ziaee, M. (2017, March). Modeling and solving the distributed and flexible job shop scheduling problem with WIPs supply planning and bounded processing times. *International Journal of Supply and Operations Management*, *4*(1), 78–89, https://doi.org/10.22034/2017.1.06