Summary

Industry 4.0 [13] describes a new era within manufacturing. With this new era, machines, factories, and supply-lines are envisioned as one connected whole, which will allow for a data-driven approach to decision-making. More than ever before, industry needs to be able to change not only what is produced, but also how things are produced. This calls for a more flexible approach to manufacturing - an approach which must enable factories to utilise their existing machinery and change production procedures at a whim. Central to this flexibility of these cyber-physical systems is the effective planning and scheduling of operations, which forms the backbone of manufacturing systems. In this context, model-checking, simulation, and formal modelling emerges as a promising approach to ensure the reliability, optimality, and correctness of planning and scheduling of products for Flexible Manufacturing Systems (FMSs)[3]. By leveraging formal verification techniques, model-checking enables rigorous analysis and validation of system models against specified properties, mitigating risks and enhancing the robustness of the decision-making process during system operation. Creating schedules for production across multiple machines, known as the job-shop problem, has been examined in model-checking to establish when each product should begin production in the next machine. To this aim, many heuristics have been created to speed up the creation of schedules. However, the creation of these schedules does not always capture all flexible elements of the systems. The movement of products between machines, capacities, and requirements for machines are often ignored. While many models of these systems have been explored extensively, the degree of freedom of the models have often been severely limited due to the state-space explosion problem. This in turn makes the modelling of the system hard. Indeed, the lack of expressiveness can lead to an incorrect model, which in turn lead to wrong assertions about the system and its operations. In turn, the schedules created using the models will not be useful.

In this paper, we first present some of the current related work within modelling and evaluation of FMSs. We then present the syntax and semantics of a formal model of our own device, which we call Context-Aware Timed Flexible Manufacturing System (CAT-FMS). Our model, rather than relying on a formal or graphical modelling language, is described using the specification and detailing of:

1. A number of machines that can be turned on, turned off, or remain active throughout the manufacturing process,
2. a number of arms that moves products around the system,
3. protocols describing which machines each product must visit, in which order, and for how long, and
4. a critical section of the protocol which must be traversed within a time limit.

For this model, we define the Input Arrival Problem (IAP) for which a solution is a schedule that follows the semantics of the model.

We use our model to verify schedules created by an existing tool, and show that the schedules can be incorrect according to the semantics. Furthermore, we present two algorithms which, given a system model and an IAP, synthesise a correct schedule for each of the products. For these algorithms, a verifier has been created to ensure the produced schedules are correct. Both the verifier and the synthesis algorithms have been implemented using `C#`. The algorithms rely on search heuristics. Four of these heuristics have been explored, two of which rely on the topology of the provided system, and gives estimates on the remaining make-span of products in the system. The heuristics are evaluated in terms of the number of configurations explored, the quality of the solution, and time used to synthesise a schedule. Using the most promising of these heuristics, a comparison of a depth-first search of the problem has been compared to a guided-depth first approach.

Scheduling for Timed Context-Aware Flexible Manufacturing Systems*

Rasmus Høyer Hansen and Daniel Overvad Nykjær

Aalborg University, Denmark {rhha19,dnykja18}@student.aau.dk
https://vbn.aau.dk/en/organisations/institut-for-datalogi

Abstract. With the rise of industry 4.0, flexible manufacturing systems are expected to become the dominant approach to product manufacturing. The intelligent and correct operations of these systems are crucial to ensure that production occurs in a safe and time-wise feasible manner. Model-based approaches to intelligent manufacturing promises to ensure correctness of the manufacturing of products. In this paper, we demonstrate how one can model Context-Aware Timed Flexible Manufacturing Systems. These systems define protocols for all products that can be manufactured, a deadline for the traversal of a subset of this protocol, machines that are active, or state-full, arms which can reach subsets of the machines. We use this model to define the synthesis of schedules for products for the system. We define semantics for Context-Aware Timed Flexible Manufacturing Systems, and use it to implement a verifier and schedule synthesiser. We use the semantics and the implemented verifier to show that an existing schedule synthesiser does not always create correct schedules. We then present a schedule synthesis algorithm tailored to Context-Aware Timed Flexible Manufacturing Systems and show four heuristic functions for the algorithm. The synthesiser, verifier and the different heuristic functions are implemented. The heuristic functions are then compared to each other and an unguided depth-first search, and one of the heuristics is selected for further examination. We then compare a guided depth-first algorithm using this heuristic to the unguided depth-first algorithm. We find that substituting schedule optimality for synthesis speed using the guided depth-first can produce a schedule that is more optimal than those found by an unguided depth-first search. The results are discussed and evaluated based on the quality of the synthesised schedule and the time to synthesise a solution.

1 Problem Analysis

Flexible Manufacturing Systems (FMSs) [12] is a category of production systems characterised by their capabilities of producing multiple products utilising different machines, each being able to perform one or more operations. To produce a product, it is moved around in the manufacturing system, reaching one or more machines in a specific order, where an operation is performed. The operation will

* Supported by SigmaNet ApS who gave insight into industry systems and practices

take some amount of time and blocks other products from entering the machine. Multiple products can be inside the FMS at the same time and might require processing in the same machines multiple times.

Therefore, one product might idle in a machine waiting for another machine to become available. The waiting product can block other products, in turn leading to a deadlock of the entire FMS. As the operations on the products can take a varying amount of time to perform, the order in which the products are manufactured can affect the time it takes to finish the manufacturing of all of the products.

The creation of schedules for FMSs is related to variants of the Job Shop Problem (JSP)[20]. The problem consists of a finite number of jobs $J_1, \ldots, J_n$, each with a set of operations $O_1, \ldots, O_n$ that needs to be performed in order, and a number of machines $M_1, \ldots M_i$ with varying manufacturing speeds. The goal is to create a schedule s.t the total time to perform all jobs is minimised.

The choice of modelling formalism used to represent the constraints and behaviour of the FMS can impact the time to find a schedule and the optimality of the found schedule. Furthermore, FMSs come in a wide variety in terms of characteristics. Some modelling formalisms are therefore more suited for representing FMSs than others. The model must represent the system in a manner where it is easy to verify whether it correctly represents the system and its intended behaviour, as schedules created using an incorrect model can obstruct manufacturing.

SigmaNet ApS [1] create controllers for Flexible Manufacturing Systems. One of these systems is characterised by

- machines having varying capacities,
- machines can be used as buffers when they are not turned on,
- robotic arms and cranes move products between machines,
- products can be processed multiple times in same machines, but for various duration of time, and
- products can become unstable at some point during processing and become stable again after some machine has processed the product.

SigmaNet ApS is looking for ways to create scheduling tools to produce schedules that correctly captures all system behaviours. Ideally, the schedules should be created in a timely manner, be safe, and obtain schedules of good quality.

They have created their own experimental scheduling tool called ST. While the tool can produce schedules fast, the engineers at SigmaNet ApS has found that the modelling language used in their tool is not well-suited for expressing the characteristics of the systems. The language is constricted due to their implementation relying on Google's OR-Solver[8] and the accompanying constraint solving language. They suspect that this lack of expressiveness will lead to incorrect system models, which in turn will produce incorrect schedules. Furthermore, the tool does not guarantee optimal schedules for the systems.

In this report, we define a formalism for describing FMSs that capture the characteristics described by SigmaNet ApS and use it to create a formal model. The formal model is used to verify schedules created by ST and a developed

synthesiser used to create new schedules for a model of the system. Heuristics are devised for the synthesiser to speed up the synthesis of schedules.

1.1 Problem definition

The problem is three-fold:

1. How can we formally express the characteristics of the systems SigmaNet ApS works with?
2. How can this formalism be used to verify schedules produced by SigmaNet ApS's current, experimental tool?
3. How can the formalism be used to synthesise provably correct schedules?

The report is structured as follows. First, we present work related to modelling FMSs. Work examining how different search strategies can impact the speed of schedule synthesis and the quality of the schedule is then presented. Then we provide syntax and semantics for Context-Aware Timed Flexible Manufacturing System (CAT-FMS), a model of FMSs with the characteristics presented by SigmaNet ApS. We then examine schedules created by SigmaNet ApS's scheduler-tool and provide examples of errors in their schedules. We then present a developed synthesis algorithm CAT^* and provide heuristics for exploring the state-space of CAT-FMSs. Experiments are conducted using a developed synthesiser and the developed heuristics to establish the performance of the heuristics in terms of runtime, number of explored configurations, and quality of the found solution.

1.2 Related work

We now present work related to the modelling of the job-shop problem and the synthesis of schedules that satisfy the problem.

Previously, we have conducted work were we examined a problem and model similar to what is described in this paper [10]. We examined the use of Timed-Arc Coloured Petri Nets and used this formalism to model Flexible Manufacturing Systems. We found that the model can express many of the flexible aspects ofFlexible Manufacturing Systems. However, the time it takes to synthesise a schedule leaves much to be desired. Therefore, additional research is needed to establish how to solve the problem in a time-wise efficient manner.

The examination of Petri Nets [23] and variants of Petri Nets combined with an A^* [11] search algorithm combined with search heuristics have been extensively researched and promising results have been established [2, 16, 4, 17, 15, 18, 22]. B. Huang and Sun [2] show how systems relying on shared resources, but with products traversing the system in a linear fashion, can be modelled using Timed Petri Nets[6]. The modelled systems are simulated and their state-space explored using an A^* search algorithm with different search heuristics. While they do show that A^* can be an effective way to solve the job-shop problem for linear systems, they do not take into account the flexible aspects some systems

might require. Our model takes into account, that a product can be moved into any machine without starting the machine.
Xiong and Zhou [25] present a heuristic function that is based on the estimated time of the current marking and the minimum expected manufacturing time remaining in the system, by adding together all the jobs each machine still have left to do. Their approach solely takes into consideration the processing time of the machines while it disregards the movement of parts.
Luo et al. [17] present a slightly altered version of Xiong and Zhou heuristics function, where the cost of the current marking is the firing time of the last transition, while the expected remaining cost of reaching a goal state is the sum of the remaining processing time for each product in the system. This processing time is based on a minimum processing time matrix. Both Luo et al. [17] and Xiong and Zhou [25] heuristic functions are based on the assumptions that all resources are available when needed and that the product always follow the path providing the lowest processing time. This in turn means they might prioritise transitions that are not optimal due to them hindering other products' movement through the system journeys. Our model does not assume resource availability, but instead focus on the behaviour of the FMS - we model machine availability by considering both when a machine can be turned on and if the correct amount and types of products are present in the machine.
Lee and DiCesare [14] present two heuristic functions. The first one changes the heuristic function to a depth component subtracting the depth of a marking from its current cost. This means that even if a marking has a large cost, it can be prioritised as it is assumed deeper markings are closer to finding a schedule. The second presented heuristic allows the search algorithm to prioritise markings where operations are closer to ending. This allows for the prioritisation of moving products forward in their journeys as well as freeing up resources.
Sun, Cheng, and Fu [24] present an alternative version of the depth prioritising heuristic function by adding a weight component. This change allows the user to control the importance of the depth information, with a higher value resulting in the prioritisation of deeper markings. However, due to the state space explosion that occurs when trying to find a suitable schedule for larger systems [2, 18, 22, 19, 17] steps have been taken towards approaches which trades optimality for computational improvements.
B. Huang and Sun [2] show that by implementing a cost factor and Depth-First Search component, the search effort can be reduced by up to 70% by giving up 15% optimality.
Moro, Yu, and Kelleher [18] demonstrate an approach where a dynamic window is used to limit the scope of the search by limiting the backtracking capabilities. They show that a search using the dynamic window can reduce the search effort by up to 33%. Additionally Peng et al. [22] show that improvements with a lower optimality cost can be obtained by improving the backtracking rules. They aim to limit the amount of good solutions the dynamic window disregards due to overly aggressive pruning of the search space. Like [2] and [22] we also examine the

trade-off between schedule optimality and computational time, and also examine a dept-first component.

2 Notation

Throughout this report, we let the set of natural number $\mathbb{N}$ include all non-negative integers. For cases where ∞ is included we use $\mathbb{N}^\infty$.

A *multi-set* M over a set A is a set containing elements of A but where duplicates are allowed. $\mathcal{M}(A)$ denotes all possible multi-sets over A. The size of a multi-set M is given by $|M|$. For any set or multi-set S we denote that a condition ϕ must hold over all elements of S by $\forall_{s \in S}\, \phi$.

A *sequence* of elements $\pi = s_1 \cdot s_2 \cdot \ldots \cdot s_n$ is an ordered, enumerated collection of elements. The *length* of π is given by $|\pi| = n$. If π has length 0, we say that it is *empty* and denote it by ε. For sequences, we define the function *head* to give the first element of a sequence if it has one or more elements and ε otherwise. The function *tail* gives all elements of the sequence, except the head. For $\pi = s_1 \cdots s_n$, $\mathit{head}(\pi) = s_1$ and $\mathit{tail}(\pi) = s_2 \cdot \ldots \cdot s_n$. Finally, the function *suffix* is defined s.t $\mathit{suffix}(\pi, i) = s_i \cdots s_n$ if $i \leq n$ and ε otherwise.

3 CAT-FMS

In this section, we first introduce the notion of a Context-Aware Timed Flexible Manufacturing Systems (CAT-FMSs) used to represent FMSs. Following this, we provide definitions to establish the semantics of CAT-FMSs and then we establish the Input Arrival Problem (IAP) and its solution.

The following definition is derived from information gained from a meeting with the CEO of SigmaNet ApS and from system documentation and presentation materials provided by SigmaNet ApS. These materials have been excluded, as they contain customer sensitive information.

Definition 1 (Context-Aware Timed Flexible Manufacturing System). *A Context-Aware Timed Flexible Manufacturing System (CAT-FMS) is a tuple* $S = (Id, E, M, A, A_T, reach, cap, req, prot, crit)$ *where*

- Id *is a finite set of product ids,*
- E *is a finite set of exit location,*
- $M = M_a \uplus M_s$ *is a finite set of machines where* M_a *are machines that are always active and* M_s *are machines that can be turned on or off,*
- A *is a finite set of arms,*
- A_T: $A \to \mathbb{N}$ *is a function mapping arms to the duration it takes for them to move products,*
- $reach$: $A \to \mathcal{P}(M \cup E)$ *is a function dictating which machines and exits each arm can reach,*
- cap: $M \cup A \to \mathbb{N}$ *assigns each arm and machine their capacity,*
- req: $M_s \to \mathcal{P}(\mathbb{N})$ *gives possible amounts of products needed in each machine before it can be started,*

- *prot*: $Id \to (M \times \mathbb{N} \times \mathbb{N})^*$ *is a function mapping products to a sequence of machines and processing intervals s.t if a product with* $id \in Id$ *then* $(m, s, e) \in prot(id)$ *defines that* id *must be in machine* m *that is turned on or* $m \in M_a$ *for a duration of* $d \in [s, e]$*, and*
- *crit*: $Id \to \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ *is a function defined over all* $id \in Id$ *s.t* $(s, e, t) = crit(id)$ *dictates a subsection of* $prot(id)$ *starting at index* s *and ending at index* e *that must be traversed within* t *time units.*

The objective of a CAT-FMS is to create a schedule defining when and which action should be executed s.t production is facilitated in an legal manner. In the following, we provide definitions and the meaning behind these schedules and actions.

Definition 2 (Schedule). *A schedule is a sequence of pairs* $(t_1, \alpha_1) \cdots (t_n, \alpha_n)$ *where* $t_1 \leq \ldots \leq t_n$ *which defines that action* α_i *occurs at time* $t_i \in \mathbb{N}$*, where each* α_i *can be either*

- $\mathbf{arrive}(pt, m)$*, meaning that products* $pt \subseteq Id, pt \neq \varnothing$ *arrive in machine* $m \in M_a$*,*
- $\mathbf{take}(a, pt, m)$*, meaning that arm* $a \in A$ *picks up products with ids* $pt \subseteq Id, pt \neq \varnothing$ *from machine* $m \in reach(a), m \in M$*,*
- $\mathbf{place}(a, pt, m)$*, meaning that arm* $a \in A$ *places products with ids* $pt \subseteq Id, pt \neq \varnothing$ *in machine or exit* $m \in reach(a)$*,*
- $\mathbf{start}(m)$*, meaning that machine* $m \in M_s$ *starts, or*
- $\mathbf{stop}(m)$*, meaning that machine* $m \in M_s$ *stops*

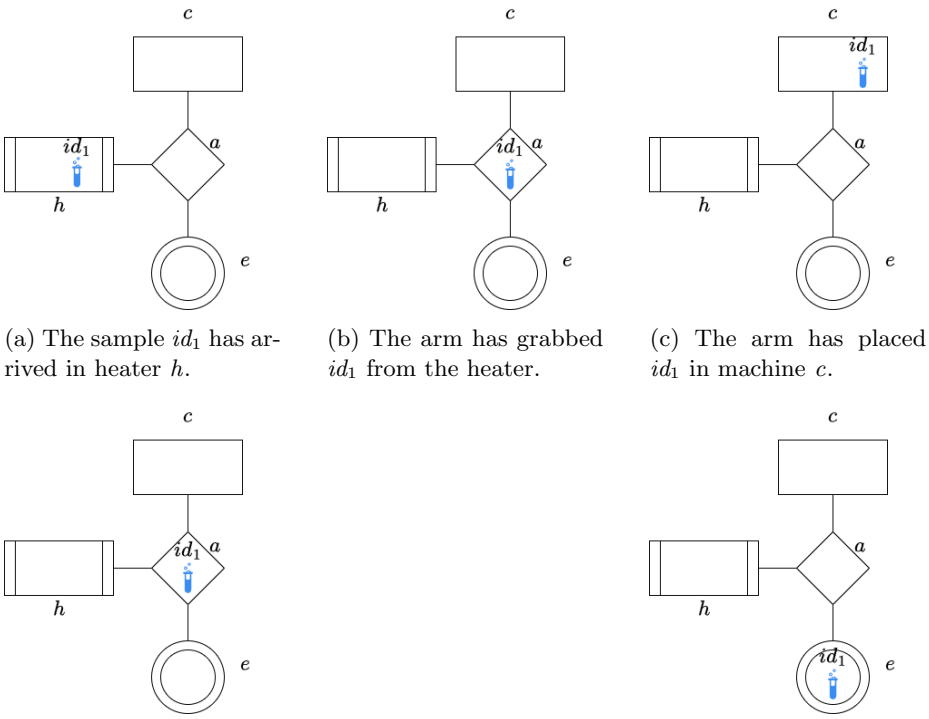
for each $1 \leq i \leq n$*.*

Let $\pi = (t_1, \alpha_1) \cdots (t_n, \alpha_n)$ be a schedule. We say that each (t, α) from π is a *schedule step*. The function $arrivals(\pi) = (t_i, \alpha_i) \cdots (t_j, \alpha_j)$ projects π to a sub-sequence where each schedule-step is an $(t, \mathbf{arrive}(pt, m))$.

The *perspective* of a machine $m \in M$ under schedule π is the sub-sequence of π where each element is of type $\mathbf{arrive}(pt, m), \mathbf{start}(m)$, or $\mathbf{stop}(m)$. The function $perspective(m, \pi)$ projects π to this sub-sequence. We say that $perspective(m, \pi)$ is the schedule π from *the perspective of* m.

For a schedule $\pi' = (t_1, \alpha_1) \cdots (t_n, \alpha_n)$ we say that π' is *operation oblivious* if it contains only actions of type $\mathbf{arrive}(pt, m), \mathbf{place}(a, pt, m)$, or $\mathbf{take}(a, pt, m)$. We say that π' is an Operation Oblivious Schedule (OOS).

3.1 Schedule example



(a) The sample id_1 has arrived in heater h.

(b) The arm has grabbed id_1 from the heater.

(c) The arm has placed id_1 in machine c.

(d) The arm has grabbed id_1 from machine c.

(e) The arm has placed id_1 in the exit location e.

Fig. 1: The different placement of the sample id_1. The active machine $h \in M_a$ is located to the west, the arm a is placed in the center, the centrifuge c to the north, and the exit e to the south.

We now present a small example to illustrate the behaviour and meaning of the actions from Definition 2. We define a small system based on the two first machines of one of SigmaNet ApS's systems. The example system contains

- a centrifuge $c \in M_s$ with $cap(c) = 2$ which can be started with 1, 2 or 4 products $req(c) = \{1, 2, 4\}$,
- a heater $h \in M_a$ with $cap(h) = 1$,
- an exit $e \in E$, and
- a robotic arm $a \in A$ with a travel time $A_T(a) = 6$ and capacity $cap(a) = 1$ which can reach all the different components $reach(a) = \{c, h, e\}$.

The system processes chemical samples - a sample id_1 have to first be heated for 50-60 time units, then be centrifuged for 30 time units. This is defined by

the protocol of the product $prot(id_1) = (h, 50, 60) \cdot (c, 30, 30)$. The sample must not get cold before centrifuged, and so must reach and be done processing in the centrifuge before 120 time units after leaving the heater. This is defined by the critical section of the product $crit(id_1) = (1, 2, 120)$.

A schedule for this system will first input id_1 to the heater, then move id_1 to the centrifuge which will then be started. Once the sample has been in the centrifuge for an appropriate amount of time, the centrifuge must be stopped and the sample moved to the exit. Figure 1 shows the different locations the sample can be placed in. When it arrives it is placed in h (Figure 1a), then it is picked up by a (Figure 1b), and then placed in centrifuge c (Figure 1c). The centrifuge is then started - once done the product is grabbed by a (Figure 1d), and finally moved to the system exit (Figure 1e).

For this system, a valid schedule could be
$\pi = (0, \mathbf{arrive}(\{id_1\}, h)) \cdot (50, \mathbf{take}(\{id_1\}, a, h)) \cdot (56, \mathbf{place}(\{id_1\}, a, c)) \cdot$
$(56, \mathbf{start}(c)) \cdot (86, \mathbf{stop}(c)) \cdot (86, \mathbf{take}(\{id_1\}, a, c)) \cdot (92, \mathbf{place}(\{id_1\}, a, e))$.

The schedule shows a scenario where both a **place** and **start** action interact with the same machine at time 56. This is still safe due to a list of operating guarantees that holds for each of the actions. Invoking **arrive**, **place**, or **stop** at time t_1 guarantees that the action is finished at time t_1. Meanwhile, invoking **take** or **start** guarantees that the action is executing at time $t_1 \leq t' \leq t_1 + 1$. For this scenario, it means that arm a has finished placing id_1 in c before we reach time 56. This allows c to begin processing id_1 at time 56 as it is not blocked by a.

The perspective of machine c is given by $perspective(c, \pi) = (56, \mathbf{place}(\{id_1\}, a, c)) \cdot$ $(56, \mathbf{start}(c)) \cdot (86, \mathbf{stop}(c)) \cdot (86, \mathbf{take}(\{id_1\}, a, c))$. From π the Operation Oblivious Schedule $\pi' = (0, \mathbf{arrive}(\{id_1\}, h)) \cdot (50, \mathbf{take}(\{id_1\}, a, h)) \cdot (56, \mathbf{place}(\{id_1\}, a, c)) \cdot$ $(86, \mathbf{take}(\{id_1\}, a, c)) \cdot (92, \mathbf{place}(\{id_1\}, a, e))$ can be achieved by excluding all schedule steps containing a $\mathbf{start}(m)$ or $\mathbf{stop}(m)$ action.

3.2 Preliminary definitions

In this section, we provide definitions used to establish when a schedule is *feasible*. In Definitions 3 and 4 we provide functions used to establish when an action interacts with an arm or machine.

Definition 3 (Arm interaction). *The set of all possible arm operations for an arm $a \in A$ is defined by $operation(a) = \{\mathbf{place}(a, pt, x) | pt \subseteq Id, x \in M \cup E, |pt| \leq cap(a)\} \cup \{\mathbf{take}(a, pt, m) | pt \subseteq Id, |pt| \leq cap(a), m \in M\}$.*

Definition 4 (Machine interaction). *The set of all possible actions involving a machine $m \in M$ is given by $interact(m) = \{\alpha | \alpha = \mathbf{place}(a, pt, m) \vee \alpha = \mathbf{take}(a, pt, m) \vee \alpha = \mathbf{stop}(m) \vee \alpha = \mathbf{start}(m), |pt| \leq cap(m), pt \subseteq Id, a \in A\}$.*

Definitions 5 and 6 define the state of the products of the CAT-FMS and a mapping from system components and the products contained within them, respectfully.

Definition 5 (Configuration). *A* configuration $\mathcal{C}$: $Id \to ((M \cup A) \times \mathbb{N}) \cup \{\bot, \top\}$ *maps all products to a location and the number of completed steps of the product's protocol. If the product is not in the system, the function gives* $\bot$*. If the product is placed in an exit, the function gives* $\top$*. The set of all configurations is* $\mathbb{C}$*.*

Definition 6 (Content). *The function content*: $(M \cup A) \times \mathbb{C} \to \mathcal{P}(Id)$ *is defined by* $content(x, \mathcal{C}) = \{id \in Id | \mathcal{C}(id) = (x, s)\}$*. As a shorthand we use* $content_{\mathcal{C}}(x)$ *to denote* $content(x, \mathcal{C})$ *for* $x \in M \cup A$ *and* $\mathcal{C} \in \mathbb{C}$*.*

Definition 7 defines the next machine each product must be processed in and Definition 8 the minimum and maximum time a set of products can be processed.

Definition 7 (Next protocol step). *The next step of a product's protocol under a configuration* $\mathcal{C}$ *is given by a function next*: $Id \times \mathbb{C} \to M \times \mathbb{N} \times \mathbb{N}$ *and is defined as* $next(id, \mathcal{C}) = head(suffix(prot(id), s))$ *where* $(m, s) = \mathcal{C}(id)$*.*

Definition 8 (Legal processing interval). *The function respect maps a set of products to the highest of minimal processing times and the lowest maximum processing times given by the next protocol steps of the products. It is defined as* $respect(pt, \mathcal{C}) = (l_b, u_b)$ *with* $l_b = max(\{l | (m, l, u) = next(id, \mathcal{C}), id \in pt\})$ *and* $u_b = min(\{u | (m, l, u) = next(id, \mathcal{C}), id \in pt\})$ *where* $\mathcal{C} \in \mathbb{C}$ *and* $pt \subseteq Id$*.*

We now present *schedule executions* and rules for how these executions behave.

3.3 Schedule execution

Let $s = (t_1, \alpha_1) \cdots (t_n, \alpha_n)$ be a schedule. An *execution* of s is a sequence of configurations separated by elements of s written as $\mathcal{C}_0 \xrightarrow{(t_1,\alpha_1)} \mathcal{C}_1 \xrightarrow{(t_2,\alpha_2)} \cdots \xrightarrow{(t_n,\alpha_n)} \mathcal{C}_n$ where the following conditions and rules are respected. If the conditions and rules are satisfied, we say that s and the execution of s is *feasible*.

Rule 1 every product can arrive at most once;

$$\forall_i \text{ if } \alpha_i = \mathbf{arrive}(pt, m) \text{ then } \nexists_{j>i} \alpha_j = \mathbf{arrive}(pt', m') \wedge pt' \cap pt \neq \varnothing$$

Rule 2 a machine $m \in M_s$ must be stopped after it is turned on, and it cannot be interacted with while being turned on;

$$\forall_i \text{ if } \alpha_i = \mathbf{start}(m) \text{ then } \exists_{j>i}\, t_j > t_i \wedge \alpha_j = \mathbf{stop}(m) \text{ and}$$
$$\nexists_{i<k<j} \alpha_k \in interact(m)$$

Rule 3 when an arm $a \in A$ picks up products $pt \subseteq Id$, it holds them for exactly $A_T(a)$ time-units before placing them in $x \in M \cup E$, and no other operation can be performed while using the arm;

$$\forall_i \text{ if } \alpha_i = \mathbf{take}(a, pt, m) \text{ then } \exists_{j>i}\, \alpha_j = \mathbf{place}(a, pt, x)$$
$$\wedge\, t_i + A_T(a) = t_j \text{ and } \nexists_{i<k<j} \alpha_k \in operation(a)$$

Rule 4 no product exceeds its deadline before leaving its protocol's critical section;

$$\forall_i \,.\, \forall_{id \in Id} \text{ where } crit(id) = (c_s, c_e, d) \text{ if } \mathcal{C}_i(id) = (x, c_s) \\ \text{then } \exists_{j>i}\, \mathcal{C}_j(id) = (x, c_e + 1) \wedge t_j - t_i \leq d$$

Rule 5 once a machine is started, it is stopped within the interval defined by all product's minimum and maximum processing times;

$$\forall_i \text{ if } (\alpha_i = \mathbf{start}(m) \text{ then } \exists_{j>i}\, \alpha_j = \mathbf{stop}(m)) \wedge t_j - t_i \in [l_b, u_b] \wedge \\ l_b \leq u_b \text{ where } (l_b, u_b) = respect(content_{\mathcal{C}_i}(m), \mathcal{C}_i)$$

Rule 6 once a product enters an active machine the product is taken out of the machine before reaching maximum processing time;

$$\forall_i \,.\, \forall_{id \in Id} \text{ where } (m, min, max) = next(id, \mathcal{C}_i) \text{ if } (\alpha_i = \mathbf{place}(a, pt, m) \wedge \\ m \in M_a \wedge id \in pt) \vee (\alpha_i = \mathbf{arrive}(pt, m) \wedge m \in M_a \wedge id \in pt) \text{ then} \\ \exists_{j>i}\, \alpha_j = \mathbf{take}(a', pt', m) \wedge id \in pt' \wedge t_j - t_i \in [min, max].$$

Furthermore, the actions in the schedule must be enabled and the execution of the actions must yield the correct configurations. For the following rules dictating enabledness and execution of an (t_i, α_i), we assume $\mathcal{C}_i(id) = \mathcal{C}_{i-1}(id)$ for any $id \in Id \setminus pt$ where $pt \subseteq Id$ are products involved in the action.

Rule 7 if $\alpha_i = \mathbf{arrive}(pt, m))$ then

a) the arrival of the products does not exceed the capacity of the machine; $cap(m) \geq |content_{\mathcal{C}_{i-1}}(m)| + |pt|$,

b) the arrived product has not completed any steps of their protocol; $\mathcal{C}_i(id) = (m, 0)$ for every product $id \in pt$, and

c) the products are not in the system yet; $\mathcal{C}_{i-1}(id) = \bot$ for all $id \in pt$.

Rule 8 if $\alpha_i = \mathbf{take}(a, pt, m)$ then

a) the products are in the machine; $content_{\mathcal{C}_{i-1}}(m) \supseteq pt$,

b) the arm is empty; $content_{\mathcal{C}_{i-1}}(a) = \varnothing$,

c) the arm has sufficient capacity; $cap(a) \geq |pt|$,

d) if $m \in M_a$ then the number of completed steps of the moved products are incremented; $\mathcal{C}_i(id) = (a, s + 1)$ where $(m, s) = \mathcal{C}_{i-1}(id)$ for every product $id \in pt$, and

e) if $m \in M_s$ then the number of completed steps remain the same; $\mathcal{C}_i(id) = (a, s)$ where $(m, s) = \mathcal{C}_{i-1}(id)$ for every product $id \in pt$.

Rule 9 if $\alpha_i = \mathbf{place}(a, pt, x)$ then

a) the product must be held by the arm; $(a, s) = \mathcal{C}_{i-1}(id)$ for every product $id \in pt$,

b) if $x \in M$ then the capacity of the machine is not violated; $|content_{\mathcal{C}_{i-1}}(x)| + |pt| \leq cap(x)$,

c) the next step of the product's protocol is x if $x \in M_a$; $next(id, \mathcal{C}_{i-1}) = (x, min, max)$, for every $id \in pt$

d) the remaining protocol of the product is empty if $x \in E$; $suffix(prot(id), s) = \varepsilon$ where $(x, s) = \mathcal{C}_{i-1}(id)$, for every id $\in pt$

e) the product leaves the system if $x \in E$; $\mathcal{C}_i(id) = \top$ for every $id \in pt$,

f) the product is placed in x if $x \in M$; $\mathcal{C}_i(id) = (x, s)$ where $(a, s) = \mathcal{C}_{i-1}(id)$ for every $id \in pt$, and

g) all products disappear from the arm; $content_{\mathcal{C}_i}(a) = \varnothing$.

Rule 10 if $\alpha_i = \mathbf{start}(m)$ then

a) the number of products in m allows the machine to start; $|content_{\mathcal{C}_i - 1}(m)| \in req(m)$, and

b) the next protocol step of all products must be m; $next(id, \mathcal{C}_{i-1}) = (m, min, max)$ for all $id \in content_{\mathcal{C}_{i-1}}(m)$.

Rule 11 if $\alpha_i = \mathbf{stop}(m)$ then

a) the number of completed steps of all products are incremented; $\mathcal{C}_i(id) = (m, s + 1)$ for all $id \in content_{\mathcal{C}_{i-1}}(m)$.

Having established the legal behaviour of CAT-FMSs and how executing the different actions result in new configurations, the task at hand is to create a schedule for a CAT-FMS s.t manufacturing using the system can occur. We therefore define the Input Arrival Problem and its solution, and following this we describe how the rules can be used to take a Operation Oblivious Schedule and convert it into a schedule which can be verified.

Definition 9 (Input Arrival Problem (IAP)). *Given an input schedule* $in = (t_1, \mathbf{arrive}(pt_1, m_1)) \cdots (t_n, \mathbf{arrive}(pt_n, m_n))$ *synthesise a schedule* $\pi = (t_1, \alpha_1) \cdots (t_j, \alpha_j)$ *with feasible execution* $\mathcal{C}_0 \xrightarrow{(t_1, \alpha_1)} \cdots \xrightarrow{(t_j, \alpha_j)} \mathcal{C}_j$ *s.t* $arrivals(\pi) = in$, $\mathcal{C}_j(id) = \top$, *and* $\mathcal{C}_0(id) = \bot$ *for all* $id \in Id$. *We refer to* $\mathcal{C}_j$ *as being a* goal configuration. *The time* t_j *is referred to as the* makespan *of the solution.*

Based on the semantics of CAT-FMSs, we develop a verifier, which can be used to establish whether a provided schedule is feasible. It will be used to establish whether a number of OOSs created using a tool developed by SigmaNet ApS are feasible.

4 Verification of SigmaNet ApS's schedules

SigmaNet ApS have developed their own scheduler using Google's tool OR-solver [8]. However, SigmaNet ApS's experimental tool, called ST, creates Operation Oblivious Schedules (OOSs). By using ST we create three OOSs by providing three different Input Arrival Problems (IAPs) and the example system from Section 3.1. The tool does not respond in a deterministic manner, and providing it with an IAP multiple times will not necessarily produce the same OOSs.

For an OOS $\pi' = (t_1, \alpha_1) \cdots (t_n, \alpha_n)$, we say that π' is *time-eligible* if it is possible to interleave π' with tuples $(t, \mathbf{start}(m))$ and $(t', \mathbf{stop}(m))$ s.t Rules 2 and 5 are both satisfied.

To examine whether ST creates OOSs which could be turned into a feasible schedule, we first establish whether the produced OOS is time-eligible. We find the times where $\mathbf{start}(m)$ and $\mathbf{stop}(m)$ actions can occur by observing when products enters each machine. If a product enters a machine at time t_i and another product at time t_j, then the $\mathbf{start}(m)$ action can occur at any time between t_i and t_j, and $\mathbf{stop}(m)$ must occur at a time less than or equal to t_j but after t_i. If the OOS is time-eligible, schedules are created by interleaving legal $\mathbf{start}(m)$ and $\mathbf{stop}(m)$ actions, and the developed verifier is used to establish whether the schedule is feasible.

4.1 First Operation Oblivious Schedule

The first schedule is created by providing ST with the IAP $in = (0, \mathbf{arrive}(\{id_1, id_2\}, h))$. ST responds with the OOS shown in Equation (1).

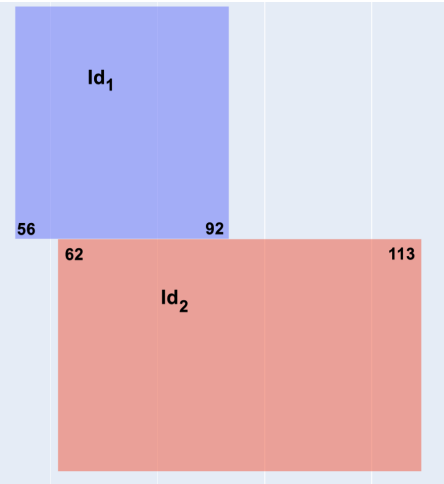
$$\begin{aligned}\pi'_1 =&(0, \mathbf{arrive}(\{id_1, id_2\}, h)) \cdot (50, \mathbf{take}(a, \{id_1\}, h)) \cdot \\ &(56, \mathbf{place}(a, \{id_1\}, c)) \cdot (56, \mathbf{take}(a, \{id_2\}, h)) \cdot \\ &(62, \mathbf{place}(a, \{id_2\}, c)) \cdot (92, \mathbf{take}(a, \{id_1\}, c)) \cdot \\ &(98, \mathbf{place}(a, \{id_1\}, e)) \cdot (113, \mathbf{take}(a, \{id_2\}, c)) \cdot \\ &(119, \mathbf{place}(a, \{id_2\}, e))\end{aligned} \tag{1}$$

We now establish whether π'_1 satisfies Rules 2 and 5 and is time-eligible. This is done by examining the perspective of each of the machines from M_s. For the case of this system only $c \in M_s$.

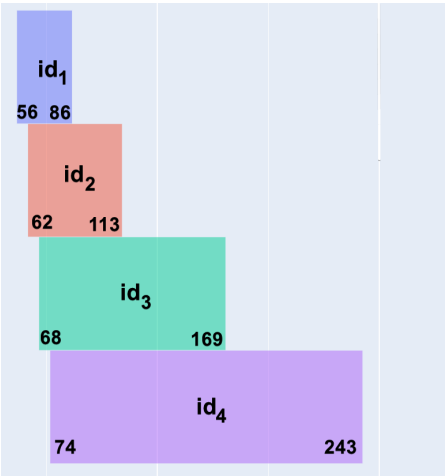
We construct the *perspective of* c which is shown in Equation (2) and is visualised on Figure 2a.

$$\begin{aligned}\mathit{perspective}(c, \pi'_1) =&(56, \mathbf{place}(a, \{id_1\}, c)) \cdot (62, \mathbf{place}(a, \{id_2\}, c)) \cdot \\ &(92, \mathbf{take}(a, \{id_1\}, c)) \cdot (113, \mathbf{take}(a, \{id_2\}, c))\end{aligned} \tag{2}$$

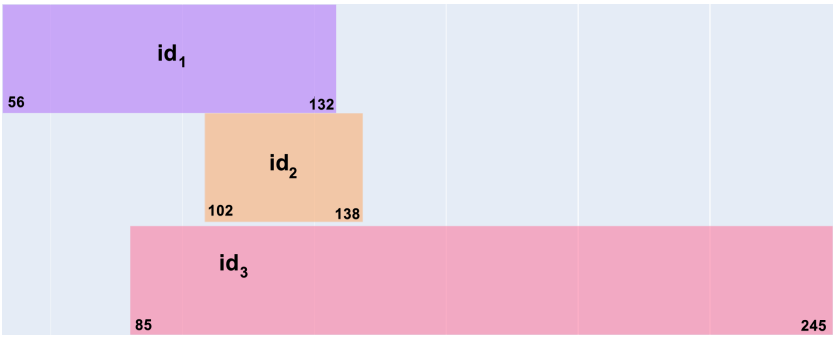
By using the described approach, we examine Equation (2) and see that it is a time-eligible OOS. A schedule which can be created by interleaving $(62, \mathbf{start}(c))$ and $(92, \mathbf{stop}(c))$ with Equation (2) satisfy both Rules 2 and 5. We create this



(a) The perspective of c given by $perspective(c, \pi_1')$ is *time-eligible* and *feasible* as the overlap of the products are more than 30 time units.



(b) The perspective of c given by $perspective(c, \pi_2')$ is *not time-eligible* due to the time-slots not overlapping for at least 30 time units.



(c) The perspective of c given by $perspective(c, \pi_3')$ is *time-eligible* but is *not feasible* because by Rule 10a machine c can only be started with $1, 2$ or 4 products.

Fig. 2: Three different OOS π_1', π_2', and π_3' from the perspective of machine c from Section 3.1. The OOSs are created using ST.

schedule, shown in Equation (3), and verify it using the developed verifier. The

verifier correctly states that π_1 is feasible.

$$\begin{aligned}\pi_1 =&(0, \mathbf{arrive}(\{id_1, id_2\}, h)) \cdot (50, \mathbf{take}(a, \{id_1\}, h)) \cdot \\ &(56, \mathbf{place}(a, \{id_1\}, c)) \cdot (56, \mathbf{take}(a, \{id_2\}, h)) \cdot \\ &(62, \mathbf{place}(a, \{id_2\}, c)) \cdot (62, \mathbf{start}(c)) \cdot \\ &(92, \mathbf{stop}(c)) \cdot (92, \mathbf{take}(a, \{id_1\}, c)) \cdot \\ &(98, \mathbf{place}(a, \{id_1\}, e)) \cdot (113, \mathbf{take}(a, \{id_2\}, c)) \cdot \\ &(119, \mathbf{place}(a, \{id_2\}, e))\end{aligned} \tag{3}$$

4.2 Second Operation Oblivious Schedule

We create the second OOS π_2' in the same manner as the creation of π_1'. We supply ST with IAP $in = (0, \mathbf{arrive}(\{id_1, id_2, id_3, id_4\}, h))$. We get the OOS in Equation (4).

$$\begin{aligned}\pi_2' =&(0, \mathbf{arrive}(\{id_1, id_2, id_3, id_4\}, h)) \cdot (50, \mathbf{take}(a, \{id_1\}, h)) \cdot \\ &(56, \mathbf{place}(a, \{id_1\}, c)) \cdot (56, \mathbf{take}(a, \{id_2\}, h)) \cdot \\ &(62, \mathbf{place}(a, \{id_2\}, c)) \cdot (62, \mathbf{take}(a, \{id_3\}, h)) \cdot \\ &(68, \mathbf{place}(a, \{id_3\}, c)) \cdot (68, \mathbf{take}(a, \{id_4\}, h)) \cdot \\ &(74, \mathbf{place}(a, \{id_4\}, c)) \cdot (86, \mathbf{take}(a, \{id_1\}, c)) \cdot \\ &(92, \mathbf{place}(a, \{id_1\}, e)) \cdot (113, \mathbf{take}(a, \{id_2\}, c)) \cdot \\ &(119, \mathbf{place}(a, \{id_2\}, e)) \cdot (169, \mathbf{take}(a, \{id_3\}, c)) \cdot \\ &(175, \mathbf{place}(a, \{id_3\}, e)) \cdot (243, \mathbf{take}(a, \{id_4\}, c)) \cdot \\ &(249, \mathbf{place}(a, \{id_4\}, e))\end{aligned} \tag{4}$$

The perspective of machine c is then given by Equation (5) and is visualised on Figure 2b.

$$\begin{aligned}\mathit{perspective}(c, \pi_2') =&(56, \mathbf{place}(a, \{id_1\}, c)) \cdot (62, \mathbf{place}(a, \{id_2\}, c)) \cdot \\ &(68, \mathbf{place}(a, \{id_3\}, c)) \cdot (74, \mathbf{place}(a, \{id_4\}, c)) \cdot \\ &(86, \mathbf{take}(a, \{id_1\}, c)) \cdot (113, \mathbf{take}(a, \{id_2\}, c)) \cdot \\ &(169, \mathbf{take}(a, \{id_3\}, c)) \cdot (243, \mathbf{take}(a, \{id_4\}, c))\end{aligned} \tag{5}$$

From Equation (5) we see that no schedule with an execution satisfying Rules 2 and 5 can be created by interleaving $\mathbf{start}(c)$ and $\mathbf{stop}(c)$ actions. This can be seen, as there are no uninterrupted sections of 30 time units, which is required by the protocol of the products. This means that no legal $\mathbf{start}(c)$ and $\mathbf{stop}(c)$ can satisfy the duration [30, 30] given by the product's protocol. No feasible schedule based on Equation (4) can therefore be created, and we need not verify any schedules.

4.3 Third Operation Oblivious Schedule

Lastly, we provide ST with IAP $in = (0, \mathbf{arrive}(\{id_1, id_2, id_3\}, h))$ and get the OOS in Equation (6).

$$\begin{aligned}\pi_3' =&(0, \mathbf{arrive}(\{id_1, id_2, id_3\}, h))\cdot\\ &(50, \mathbf{take}(a, \{id_1\}, h)) \cdot (56, \mathbf{place}(a, \{id_1\}, c))\cdot\\ &(79, \mathbf{take}(a, \{id_3\}, h)) \cdot (85, \mathbf{place}(a, \{id_3\}, c))\cdot\\ &(96, \mathbf{take}(a, \{id_2\}, h)) \cdot (102, \mathbf{place}(a, \{id_2\}, c))\cdot\\ &(132, \mathbf{take}(a, \{id_1\}, c)) \cdot (138, \mathbf{place}(a, \{id_1\}, e))\cdot\\ &(138, \mathbf{take}(a, \{id_2\}, c)) \cdot (144, \mathbf{place}(a, \{id_2\}, e))\cdot\\ &(245, \mathbf{take}(a, \{id_3\}, c)) \cdot (251, \mathbf{place}(a, \{id_3\}, e))\end{aligned} \tag{6}$$

Again, we construct the perspective of c, which is given by Equation (7) and is visualised on Figure 2c.

$$\begin{aligned}\mathit{perspective}(c, \pi_3') =&(56, \mathbf{place}(a, \{id_1\}, c)) \cdot (85, \mathbf{place}(a, \{id_3\}, c))\cdot\\ &(102, \mathbf{place}(a, \{id_2\}, c)) \cdot (132, \mathbf{take}(a, \{id_1\}, c))\cdot\\ &(138, \mathbf{take}(a, \{id_2\}, c)) \cdot (245, \mathbf{take}(a, \{id_3\}, c))\end{aligned} \tag{7}$$

We see that we cannot interleave $(56, \mathbf{start}(c))$ and $(84, \mathbf{stop}(c))$ because this would violate Rules 2 and 5, as the protocol of id_1 would be violated. This applies for any $\mathbf{stop}(c)$ occurring before time 85. We cannot interleave $(85, \mathbf{start}(c))$ and $(102, \mathbf{stop}(c))$ as this would violate the protocol of p_3. But we can interleave $(102, \mathbf{start}(c))$ and $(132, \mathbf{stop}(c))$ to get an execution which satisfies Rules 2 and 5. This schedule is shown in Equation (8) We then try to verify π_3 using the verifier, which responds with an error; executing $\mathbf{start}(c)$ at any time where three products are placed in c will violate Rule 10a, and therefore a feasible schedule cannot be created from π_3'.

$$\begin{aligned}\pi_3 =&(56, \mathbf{place}(a, \{id_1\}, c)) \cdot (85, \mathbf{place}(a, \{id_3\}, c))\cdot\\ &(102, \mathbf{place}(a, \{id_2\}, c)) \cdot (102, \mathbf{start}(c))\cdot\\ &(132, \mathbf{stop}(c)) \cdot (132, \mathbf{take}(a, \{id_1\}, c))\cdot\\ &(138, \mathbf{take}(a, \{id_2\}, c)) \cdot (245, \mathbf{take}(a, \{id_3\}, c))\end{aligned} \tag{8}$$

We have now found that two of the three OOSs could not be used to create a feasible schedule. Thus, ST produces schedules which, if executed in any manner, may not always guarantee correct operation of the manufacturing system, as defined by the semantics in Section 3.3. We therefore develop a *synthesiser* called CAT-Synth which integrates with the verifier. The synthesiser can be used to create schedules which are verifiably feasible. In the following section, we present a synthesis algorithm based on the exploration of CAT-FMSs as well as different heuristics which the algorithm can use.

5 CAT-FMS Synthesis

We have now established that SigmaNet ApS's tool ST does not always produce feasible schedules. The task at hand is now to synthesise feasible schedules which are verifiably correct and as fast as possible.

Given an IAP p and CAT-FMS s, we need to synthesise a feasible schedule by executing actions defined in Definition 2 while adhering to Rules 1 to 11.

We present a search algorithm called CAT^*. The algorithm is a variation of the A^* algorithm [11, 21] adapted to CAT-FMSs. CAT^* searches for a goal configuration by maintaining a priority queue of configurations, where the priority is given by the function $f(\mathcal{C}) = g(\mathcal{C}) + h(\mathcal{C})$ where $\mathcal{C} \in \mathbb{C}$. The heuristic function $h(\mathcal{C})$ estimates the time-wise distance from the given configuration to the goal configuration, while $g(\mathcal{C})$ is the time used to reach the given configuration. This is visualised on Figure 3. The algorithm also maintains pointers from

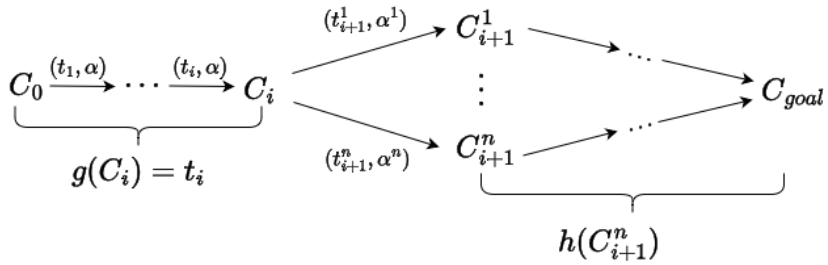


Fig. 3: Illustration showing how CAT^* explores new configurations and evaluates them based on the heuristic function. For $\mathcal{C}_{i+1}^n$ the priority is given $f(\mathcal{C}_{i+1}^n) = t_i + h(\mathcal{C}_{i+1}^n)$

.

a configuration back to its predecessor. The pointers from a configuration $\mathcal{C}_i$ can be traversed to create an execution of the form $\mathcal{C}_0 \xrightarrow{(t_1,\alpha_1)} \cdots \xrightarrow{(t_{i+1},\alpha_{i+1})} \mathcal{C}_{i+1}$, which is also shown on Figure 3.

The algorithm tracks which configurations have been reached before, and may re-explore a configuration if it is reached in a cheaper manner than previously. If this occurs, the predecessor pointers are adjusted accordingly.

To progress the search for a goal configuration, the algorithm must construct all possible executions by extending the current execution with $\mathcal{C}_i \xrightarrow{(t_{i+1},\alpha_{i+1})} \mathcal{C}_{i+1}$, and then select which configuration to explore next based on the configurations' f value. This is repeated until all possible configurations are explored or a goal configuration is reached. As shown on Figure 3, an infinite number of new configurations may be reached by executing an action α under $\mathcal{C}_i$ in a manner dictated by Rules 7 to 11.

To bind the number of possible schedule steps to a finite amount, an upper bound for t_{i+1} is computed by considering the following limiting factors:

Limiting Factor 1 If any arm a has taken any product, the upper bound is limited by the remaining time until a placement must occur (Rule 3).

Limiting Factor 2 If a product is within the critical section of its protocol, then the time to reach it's deadline sets an upper limit (Rule 4).

Limiting Factor 3 If any machine is started or a product is placed in an active machine, the upper bound is limited by the protocol of the product (Rules 5 and 6).

Limiting Factor 4 Any $(t_a, \mathbf{arrive}(pt, m))$ from the given Input Arrival Problem sets an upper bound of t_a if $t_i < t_a$.

The upper bound of t_{i+1} is then the minimum of Limiting Factors 1 to 4. In case non of these factors sets an upper limit, the action must occur at $t_{i+1} = t_i$.

Theorem 1. *For any feasible schedule there exist another feasible schedule that satisfies Limiting Factors 1 to 4 and with an equal or better makespan.*

Proof

Let $\pi = (t_1, \alpha_1) \cdot (t_2, \alpha_2) \cdots (t_n, \alpha_n)$ be a feasible schedule that solves IAP p. We show how to construct a restricted solution satisfying Limiting Factors 1 to 4. The execution of π satisfies

- Limiting Factor 1 because if this is not the case and there exists a reached configuration s.t the time difference between a $\mathbf{place}(a, pt, m)$ and a $\mathbf{take}(a, pt, m)$ is greater than $A_T(a)$, then this contradict the feasibility of π by breaking Rule 3.
- Limiting Factor 2 because if this is not the case and there exists a reached configuration s.t the time since a part has entered its critical state is greater than it's deadline, then this breaks Rule 4 and π would not be feasible.
- Limiting Factor 3 because if this is not the case and there exists a reached configuration s.t the time difference between a $\mathbf{place}(a, pt, m)$ and $\mathbf{take}(a, pt, m)$ where $m \in M_A$ or an $\mathbf{start}(m)$ and $\mathbf{stop}(m)$ is greater than what is dictated by any relevant product's protocol, then this contradicts the feasibility of π by breaking Rule 5 or Rule 6.
- Limiting Factor 4 because if this is not the case then π does not solve p, since p defines when **arrive** actions must occur and any schedule delaying past any of these would not solve p.

We show that when non of Limiting Factors 1 to 4 applies, a solution of equal or better quality can be created.
Let $\pi_{sub} = \mathcal{C}_i \xrightarrow{(t_{i+1}, \alpha_{i+1})} \mathcal{C}_{i+1} \cdots \xrightarrow{(t_n, \alpha_n)} \mathcal{C}_n$ be an execution of a sub-sequence of π where $t_{i+1} \neq t_i$ and let $\mathcal{C}_{i+1}$ be a configuration where Limiting Factors 1 to 4 does not apply.
Shifting all times in π_{sub} to the execution $\mathcal{C}_i \xrightarrow{(t_{i+1}-d, \alpha_{i+1})} \mathcal{C}_{i+1} \xrightarrow{(t_2-d, \alpha_2)} \cdots \xrightarrow{(t_n-d, \alpha_n)} \mathcal{C}_n$ will not break any rules because:

- Rule 1: If any $\mathbf{arrive}(pt, m)$ must occur at time $t_j \geq t_i$ then Limiting Factor 4 applies, and an upper bound exists.
- Rule 2: If a $\mathbf{start}(m)$ has occured without a subsequent $\mathbf{stop}(m)$ then Limiting Factor 3 applies, and an upper bound exists.
- Rule 3: Shifting all times by d means that any pair of $\mathbf{take}(a, pt, m)$ and $\mathbf{place}(a, pt, m')$ will be shifted an equal amount, and Rule 3 is not violated.
- Rule 4: By Limiting Factor 2, if a product is within the critical section of its protocol, then an upper bound exists.
- Rule 5: Shifting all times by d means that any pair of $\mathbf{start}(m)$ and $\mathbf{stop}(m)$ will be shifted an equal amount, and Rule 5 is not violated.
- Rule 6: If Rule 6 is relevant, then Limiting Factor 3 applies and an upper bound exist.
- Rules 7 to 11 are all vacuous as they do not include time constraints.

Having established how CAT^* restricts the generation of new configurations to a finite amount we now describe a number of heuristic functions which can be used by CAT^*.

5.1 Heuristics

Heuristic functions are used to guide the search of CAT^*. The more precise a heuristic function can estimate the cost to reach a goal configuration, the faster a solution can be found. However, an optimal solution is only guaranteed if the heuristic is admissible.

Definition 10 (Admissible heuristic function). *A heuristic function* h: $\mathbb{C} \rightarrow \mathbb{R}$ *is admissible iff* $\forall_{\mathcal{C} \in \mathbb{C}}\ h(\mathcal{C}) \leq h^*(\mathcal{C})$ *where* $h^*(\mathcal{C})$ *is the exact cost to reach the goal configuration. [11]*

Theorem 2 ([11]). *If an admissible heuristic* h *is used, then A* finds an optimal solution.*

Theorem 3 ([5]). *If* $h(\mathcal{C}) = 0$ *for all* $\mathcal{C} \in \mathbb{C}$ *the search turns into an unguided search until an optimal solution is found.*

Many of the our developed heuristics examine the minimum time it takes to reach each machine and arm, and compute the minimum remaining processing times to establish the cost of a configuration. We now define a number of functions used in the heuristics. The multi-set of minimum remaining processing times of a configuration is given by

$$min_{times}(\mathcal{C}) = \{t_{min} | (x, s) = \mathcal{C}(id), (m, t_{min}, t_{max}) \in \mathit{suffix}(\mathit{prot}(id), s) \text{ for all } id \in Id\}$$

The multi-set of minimum remaining processing times of a configuration with the addition of travel times for each product uses a reachability graph. This directed graph is defined as

$$G_{travel} = \{(x, a, 0) | x \in \mathit{reach}(a), a \in A\} \cup \{(a, x, \frac{A_T(a)}{\mathit{cap}(a)}) | x \in \mathit{reach}(a), a \in A\}$$

Each $(x, x', n) \in G_{travel}$ describes that an edge of weight n exists between x and x'. The reachability graph assumes the most direct path to each element can be used, and that no blocking occurs along this path. It also assumes that the arm utilises all of its capacity. To establish the pair-wise shortest path of all arms, exits, and machines, the Floyd-Warshall algorithm [7] is used to create a lookup table. This look-up is defined by function $dist_{min}$: $(A \cup M \cup E) \times (M \cup A \cup E) \rightarrow \mathbb{R}$. For a protocol $prot = (m_1, t_1, t_1') \cdots (m_n, t_n, t_n')$ the amortised travel time cost of the protocol is given by $dist_{seq}(prot) = dist_{min}(m_1, m_2) + dist_{min}(m_2, m_3) + \ldots + dist_{min}(m_{n-1}, m_n)$.

We combine the remaining minimum processing time with the time to traverse protocols using the function

$$min_{times+travel}(\mathcal{C}) = \{t_{min} + dist_{seq}(remain) | (x, s) = \mathcal{C}(id),$$
$$remain = suffix(prot(id), s), (m, t_{min}, t_{max}) \in remain \text{ for all } id \in Id\}$$

Maximum Remaining Processing Time (MRPT) is based on an admissible heuristic presented by B. Huang and Sun [2] and uses the minimum processing time for each products' remaining protocol to estimate the cost of reaching the goal configuration. The heuristic assumes that all products can be processed in parallel, and the transportation between different machines occurs instantaneously. The heuristic is given by

$$MRPT(\mathcal{C}) = max(min_{times}(\mathcal{C})) \tag{9}$$

Maximum Remaining Processing Time w. Transportation (MRPT-T) is based on MRPT but tries to account for the time it takes to reach the different machines. The heuristic is given by

$$MRPT - T(\mathcal{C}) = max(min_{times+travel}(\mathcal{C})) \tag{10}$$

Remaining Processing Time (RPT) uses the minimum processing time for each products' remaining protocol to estimate the cost of reaching the goal configuration. The heuristic prioritise configurations where the sum of remaining processing times are reduced as much as possible.

The heuristic is given by

$$RPT(\mathcal{C}) = \sum min_{times}(\mathcal{C}) \tag{11}$$

Remaining Processing Time w. Transportation (RPT-T) is similar to RPT in that it accounts for an optimistic estimate of the total remaining processing time for a configuration, but tries to consider travel times between machines. RPT-T is defined as

$$RPT - T(\mathcal{C}) = \sum min_{times+travel}(\mathcal{C}) \tag{12}$$

5.2 RPT and RPT-T are not admissible

The heuristic functions RPT and RPT-T are not admissible, and are therefore not guaranteed to find optimal solutions. This is best shown using an example. Imagine a system with two machines $m_1, m_2 \in M_S$, one exit $e \in E$, one arm a with $reach(a) = \{e, m_1, m_2\}$, $A_T(a) = 1$, and $cap(a) = 1$. Let $prot(id_1) = (4, 4, m_1)$ and $prot(id_2) = (4, 4, m_2)$. Let $\mathcal{C}$ be the current configuration reached in time 0, and let id_1 and id_2 be products placed in m_1 and m_2 respectfully.

We see that $RPT(\mathcal{C}) = \sum\{4,4\} = 8$ and that $MRPT - T(\mathcal{C}) = \sum\{4+1, 4+1\} = 10$. However, we see that the execution $\mathcal{C} \xrightarrow{(0,\mathbf{start}(m_1))} \mathcal{C}_1 \xrightarrow{(0,\mathbf{start}(m_2))} \mathcal{C}_2 \xrightarrow{(4,\mathbf{take}(a,\{id_1\},m_1))} \mathcal{C}_3 \xrightarrow{(5,\mathbf{place}(a,\{id_1\},e))} \mathcal{C}_4 \xrightarrow{(5,\mathbf{take}(a,\{id_2\},m_2))} \mathcal{C}_5 \xrightarrow{(6,\mathbf{place}(a,\{id_2\},e))} \mathcal{C}_6$ reaches a goal configuration in just 6 time units.

6 Experimental evaluation of synthesis

In this section, we describe an experimental evaluation of a number of synthesis algorithms and heuristics used to solve different Input Arrival Problems (IAPs). The algorithm from Section 5 and the heuristics from Section 5.1 have been implemented in a `C#` program called CAT-Synth (CAT-Synth). A verifier establishing whether a schedule adheres to Rules 1 to 11 has also been implemented, and every schedule found has been verified. The implementation can be found on GitHub[9] The experiments were run on a Linux machine with the Ubuntu 20.04 operating system. The machine was set to use 2 AMD EPYC 7642 processors. A timeout of 10 minutes were used.

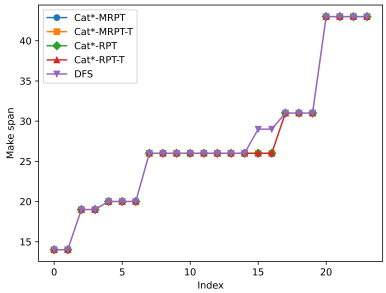
First, we examine which of the heuristics for CAT^* gives the most promising results, in terms of time to synthesise a feasible schedule and number of configurations explored when finding it. We then compare the makespan of the solutions found by CAT^* to that of the an implemented DFS algorithm. The DFS prioritise the immediate execution of action over a delayed execution.

As problem input, we scale the length of the input sequence, the frequency of arrivals, and how many products arrive at the same time. We examine three CAT-FMSs and define protocols and critical sections for them. The CAT-FMSs definitions used in these experiments are described in Appendices A.1 and A.2 and the IAPs instances are described in Appendix A.

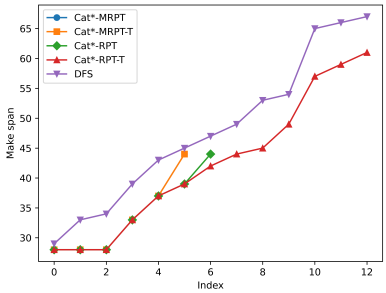
On Figure 4a we see that CAT^* finds schedules with lower makespans in the instances where it is possible. On Figure 4b we see that for all the IAP, the heuristic functions guide CAT^* to solutions with a lower makespan than the DFS. This trends continues on Figure 4c where the found solutions have a substantially lower makespan. However, the number of problems solved by the DFS far surpasses that of CAT^*.

We now select on of the heuristics based on the time it took to find a solution and the amount of solutions found, and then use it as a heuristic for a guided DFS algorithm called CAT^*_{DFS}.

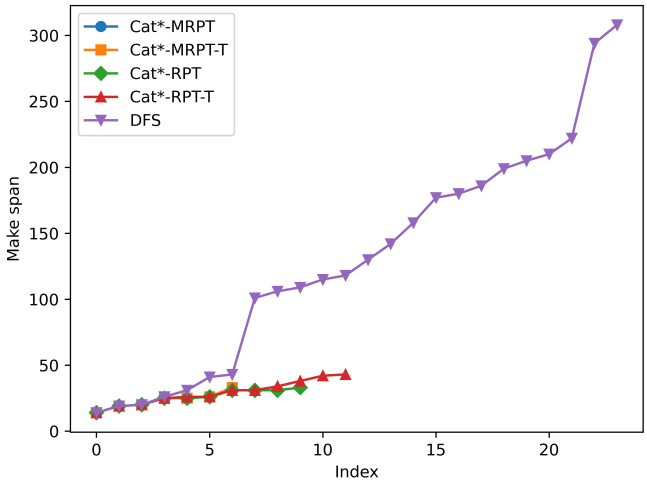
$\mathbf{CAT^*_{DFS}}$ is inspired by B. Huang and Sun [2]. The intuition behind the algorithm is to create a guided depth-first traversal of the CAT-FMS. This is done by maintaining a record of depth for each configuration and selecting a subset of configuration which yields an adequate f-value decided using a constant value ϵ. The deepest configuration in this subset is then examined to generate reachable configurations. Again, a prioritisation of the immediate execution of action is done. A higher ϵ value allows for the exploration of worse configuration, while



(a) Cactus plot showing the makespan of System 1 for each of the different heuristics and DFS



(b) Cactus plot showing the makespan of System 2 for each of the different heuristics and DFS



(c) Cactus plot showing the makespan of System 3 for each of the different heuristics and DFS

Fig. 4: Cactus plots showing the makespan for the three systems defined in Appendices A.1 to A.3

a smaller ϵ will yield results closer to that of CAT^*. The slack in f values are decided by the formula $f(\mathcal{C}) \times (1 + \epsilon)$.

Tables 1 to 4 in Appendix B show the time it took and the number of configurations explored before finding a solution or timing out. The comparison of the different heuristic functions in terms of time and number of configurations explored are shown on Figures 5 and 6 respectfully. We see that, the heuristic RPT-T yields the best results in terms of configurations explored and feasi-

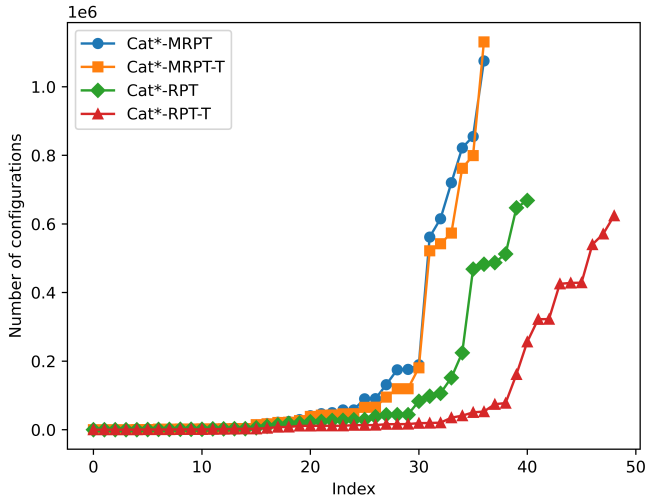


Fig. 5: Cactus plot showing the number of searched configurations for each of the different heuristics.

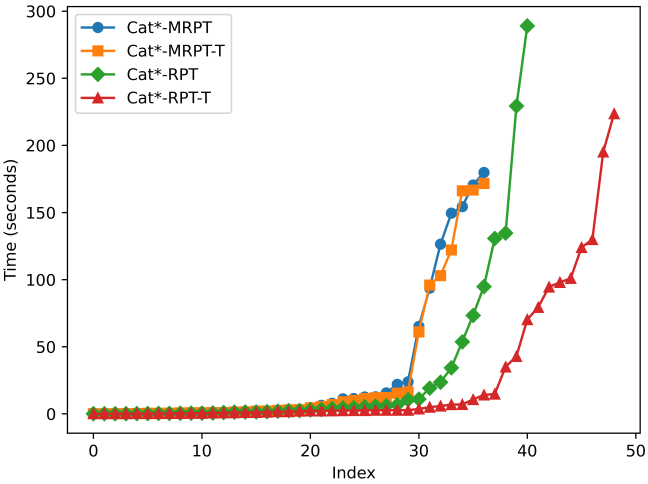


Fig. 6: Cactus plot showing the time before running out of memory, exploring all configurations, or synthesising a feasible schedule.

ble schedules found. In the next experiments, we will use RPT-T to guide the CAT^*_{DFS} algorithm.

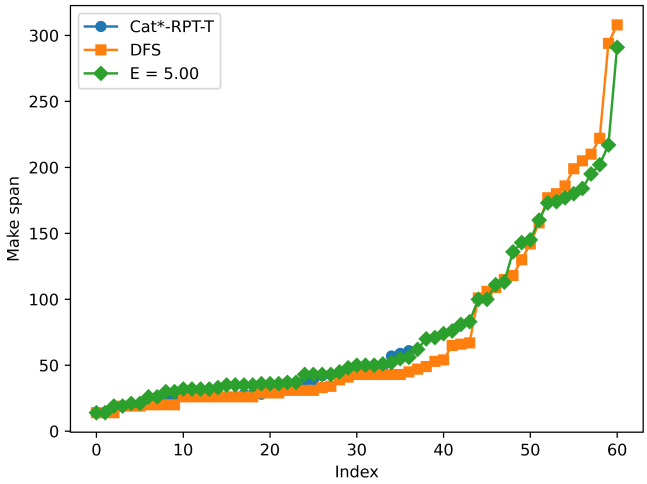


Fig. 7: Cactus plot showing the makespans of synthesised schedules found using DFS, CAT^* and CAT^*_{DFS} with $\epsilon = 5$ using RPT-T.

We now compare CAT^*_{DFS} with different ϵ values to the DFS and CAT^*. For the experiments epsilon values of $1.00, 1.25, 1.50, 1.75, 2.00, 2.25, 2.50, 2.75, 3.00, 3.50, 4.00, 4.50, 5.00$ were used. We found that an ϵ value of 5 is needed to compete with DFS in terms of number problem instances solved.

Figure 7 show the difference in makespan for CAT^*, CAT^*_{DFS} with $\epsilon = 5$, and DFS. Figure 8 shows the time it took for each algorithm to synthesise a feasible schedule. Note that each algorithm does not necessarily solve the same problem instances. We see that for smaller, less complex problems, DFS beats CAT^*_{DFS} in terms of makespan and configurations explored. For complex problems, the solutions found using CAT^*_{DFS} yields remarkably lower makespans, but require a longer time to synthesise the solution.

7 Conclusion

In this paper, we document a formal model which is used to describe Flexible Manufacturing Systems with the same characteristics of the systems SigmaNet ApS develops controllers for. The formal model called CAT-FMS was derived from information gained from a presentation by the CEO of SigmaNet ApS and technical documentation. We define schedules, how these schedules are executed, and how execution changes the state of the CAT-FMS. This is shown

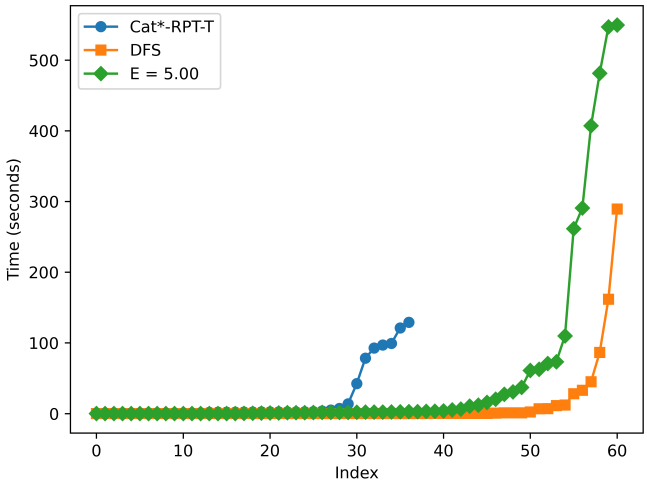


Fig. 8: Cactus plot showing time to synthesise a feasible schedule for DFS, CAT^* and CAT^*_{DFS} with $\epsilon = 5$ using RPT-T.

first through an example, and then formally through semantic rules describing when a schedule is feasible. We then define the Input Arrival Problem and its solution.

The model has been used to develop a verifier in `C#` capable of verifying whether a given schedule is feasible. The verifier has been used to explore and Operation Oblivious Schedule schedules synthesised by SigmaNet ApS's experimental tool ST, and we showed that the synthesised Operation Oblivious Schedule could not always be turned into a feasible schedule.

Additionally, the model has been used to describe an A^* algorithm, and we showed how one can limit the number of generated CAT-FMS configurations to a finite amount. This algorithm has been used to develop a synthesiser called CAT-Synth. The synthesiser integrates with the developed verifier, and implements the three algorithms CAT^*, DFS, and CAT^*_{DFS}, and the four search heuristics defined in Section 5.1.

We then examine the effectiveness of the developed heuristics and algorithms in terms of schedule quality, synthesis time, and configurations explored. This was done by comparing CAT^* with DFS. We found that DFS is able to solve more problems before reaching a timeout, but that the makespans of the solutions are worse. We then showed how CAT^*_{DFS} can use a heuristic and various ϵ-values in a guided depth-first approach and find an equal amount of solutions to that of DFS while improving the quality of the solutions.

8 Discussion & Future works

In Section 4 we verify Operation Oblivious Schedules (OOSs) produced using SigmaNet ApS's tool ST. This was done by creating time-eligible Operation Oblivious Schedules by manually annotating the Operation Oblivious Schedules with **start**(m) and **stop**(m) actions. Annotating the Operation Oblivious Schedules with **start**(m) and **stop**(m) actions by hand reduced the number of OOSs that could be examined, as the process of creating time-eligible OOSs was time consuming. Furthermore, this places limitations on the complexity of systems and problem instances which could be annotated due to human-errors being likely to occur as complexity grows. One could instead automate this annotation. As ST uses an entirely different formalism, the translation between that of ST and CAT-FMS might not be straight-forward. To fully explore the capacities of ST, a translation between the formalism's should be described and implemented.

In Section 6 we saw that a non-admissible heuristic solved more problem instances than that of the admissible heuristic. We suspect that this is due to the used admissible heuristic being designed around a system where all products are inside of the system from the beginning. Our problem model instead inputs products into the system over time, which means that only when all products are placed in the system, the f-values can change. This might severely limit the heuristic function's ability to guide the search early in the exploration. Therefore we suggest that better admissible heuristics should be developed for CAT-FMS. We suggest the examination of heuristics where

- the number of products required before a machine can be started is examined,
- the cost of the entire system is taken into account as this showed promising results when evaluating RPT. However, for this to be admissible one must take into account concurrent behaviour of the system.

Our implementation of CAT^*_{DFS} and DFS prioritise the immediate execution of actions over a delayed execution. During preliminary evaluation of CAT^*_{DFS} and DFS, implementations where the immediate execution of actions were not prioritised over a delayed execution were explored. However, these implementations showed a substantial decrease in the number of solutions found within the time limit. These were therefore not examined further. To better establish the effect of heuristics, a two-level randomisation could be created s.t more experiments could be performed while still finding solutions within the time limit. We imagine a randomisation on the order of actions, while still prioritising immediate execution, to ensure a more random execution comparable to what we have tested.

The experiments performed in Section 6 show that scaling of the IAP for CAT-FMSs is hard, and that the synthesis of schedules leaves much to be desired for these systems. Approaches which could speed up the synthesis of feasible schedules should be explored. One such example could be the decomposition of the IAP into smaller sub-problems, which are solved separately and then merged together into a single solutions.

A Experiment Setups

For the experiments we use subsets of $Id = \{id_1, \ldots id_6, id_7, \ldots, id_{12}\}$ and create several systems. We fix topology, protocols and critical section of the protocol for the systems, and examine the time it take to synthesise feasible schedules, crash, reach a timeout, or examine all configurations. Experiments were performed on the following IAP instances:

1. $(0, \mathbf{arrive}(\{id_1, id_2\}, m_{in}))$
2. $(0, \mathbf{arrive}(\{id_7, id_8\}, m_{in}))$
3. $(0, \mathbf{arrive}(\{id_1\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_2\}, m_{in}))$
4. $(0, \mathbf{arrive}(\{id_7\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_8\}, m_{in}))$
5. $(0, \mathbf{arrive}(\{id_7, id_8\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_9, id_{10}\}, m_{in}))$
6. $(0, \mathbf{arrive}(\{id_1, id_2\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_3, id_4\}, m_{in}))$
7. $(0, \mathbf{arrive}(\{id_1, id_2\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_7, id_8\}, m_{in}))$
8. $(0, \mathbf{arrive}(\{id_1\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_2\}, m_{in})) \cdot (12, \mathbf{arrive}(\{id_3\}, m_{in}))$
9. $(0, \mathbf{arrive}(\{id_7\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_8\}, m_{in})) \cdot (12, \mathbf{arrive}(\{id_9\}, m_{in}))$
10. $(0, \mathbf{arrive}(\{id_1\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_2\}, m_{in})) \cdot (12, \mathbf{arrive}(\{id_3\}, m_{in})) \cdot (18, \mathbf{arrive}(\{id_4\}, m_{in}))$
11. $(0, \mathbf{arrive}(\{id_1\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_2\}, m_{in})) \cdot (12, \mathbf{arrive}(\{id_7\}, m_{in})) \cdot (18, \mathbf{arrive}(\{id_8\}, m_{in}))$
12. $(0, \mathbf{arrive}(\{id_7\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_8\}, m_{in})) \cdot (12, \mathbf{arrive}(\{id_9\}, m_{in})) \cdot (18, \mathbf{arrive}(\{id_{10}\}, m_{in}))$
13. $(0, \mathbf{arrive}(\{id_1, id_2\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_3, id_4\}, m_{in})) \cdot (12, \mathbf{arrive}(\{id_5, id_6\}, m_{in}))$
14. $(0, \mathbf{arrive}(\{id_1, id_2\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_3, id_4\}, m_{in})) \cdot (12, \mathbf{arrive}(\{id_7, id_8\}, m_{in}))$
15. $(0, \mathbf{arrive}(\{id_1, id_2\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_7, id_8\}, m_{in})) \cdot (12, \mathbf{arrive}(\{id_9, id_{10}\}, m_{in}))$
16. $(0, \mathbf{arrive}(\{id_1, id_2\}, m_{in})) \cdot (6, \mathbf{arrive}(\{id_7, id_8\}, m_{in})) \cdot (12, \mathbf{arrive}(\{id_3, id_4\}, m_{in}))$

17. $(0, \mathbf{arrive}(\{id_7, id_8\}, m_{in}))\cdot(6, \mathbf{arrive}(\{id_1, id_2\}, m_{in}))\cdot(12, \mathbf{arrive}(\{id_9, id_{10}\}, m_{in}))$

18. $(0, \mathbf{arrive}(\{id_7, id_8\}, m_{in}))\cdot(6, \mathbf{arrive}(\{id_9, id_{10}\}, m_{in}))\cdot(12, \mathbf{arrive}(\{id_1, id_2\}, m_{in}))$

19. $(0, \mathbf{arrive}(\{id_7, id_8\}, m_{in}))\cdot(6, \mathbf{arrive}(\{id_9, id_{10}\}, m_{in}))\cdot(12, \mathbf{arrive}(\{id_{11}, id_{12}\}, m_{in}))$

20. $(0, \mathbf{arrive}(\{id_1, id_7\}, m_{in}))\cdot(6, \mathbf{arrive}(\{id_2, id_8\}, m_{in}))\cdot(12, \mathbf{arrive}(\{id_3, id_9\}, m_{in}))$

21. $(0, \mathbf{arrive}(\{id_1\}, m_{in}))\cdot(6, \mathbf{arrive}(\{id_2\}, m_{in}))\cdot(12, \mathbf{arrive}(\{id_3\}, m_{in}))\cdot(18, \mathbf{arrive}(\{id_4\}, m_{in}))\cdot$
$(24, \mathbf{arrive}(\{id_7\}, m_{in}))\cdot(30, \mathbf{arrive}(\{id_8\}, m_{in}))$

22. $(0, \mathbf{arrive}(\{id_1\}, m_{in}))\cdot(6, \mathbf{arrive}(\{id_2\}, m_{in}))\cdot(12, \mathbf{arrive}(\{id_3\}, m_{in}))\cdot(18, \mathbf{arrive}(\{id_4\}, m_{in}))\cdot$
$(24, \mathbf{arrive}(\{id_5\}, m_{in}))\cdot(30, \mathbf{arrive}(\{id_6\}, m_{in}))$

23. $(0, \mathbf{arrive}(\{id_1\}, m_{in}))\cdot(6, \mathbf{arrive}(\{id_2\}, m_{in}))\cdot(12, \mathbf{arrive}(\{id_7\}, m_{in}))\cdot(18, \mathbf{arrive}(\{id_8\}, m_{in}))\cdot$
$(24, \mathbf{arrive}(\{id_9\}, m_{in}))\cdot(30, \mathbf{arrive}(\{id_{10}\}, m_{in}))$

24. $(0, \mathbf{arrive}(\{id_7\}, m_{in}))\cdot(6, \mathbf{arrive}(\{id_8\}, m_{in}))\cdot(12, \mathbf{arrive}(\{id_9\}, m_{in}))\cdot(18, \mathbf{arrive}(\{id_{10}\}, m_{in}))\cdot$
$(24, \mathbf{arrive}(\{id_{11}\}, m_{in}))\cdot(30, \mathbf{arrive}(\{id_{12}\}, m_{in}))$

A.1 System 1

Topology $E = \{e\}$, $M_s = \{c\}$, $M_a = \{m_{in}\}$, $A = \{a\}$, $A_T(a) = 1$, $reach(a) = \{e, c, m_{in}\}$, $cap(c) = 4$, $req(c) = \{2, 3, 4\}$.

Protocols $prot(id_1) = prot(id_2) = \ldots = prot(id_6) = (m_{in}, 1, 34)\cdot(c, 3, 4)$, $prot(id_7) = prot(id_8)\ldots prot(id_{12}) = (m_{in}, 1, 34)\cdot(c, 2, 4)$.

Critical section $crit(id_1) = crit(id_2)\ldots crit(id_6) = (0, 1, 20)$ and $crit(id_7) = crit(id_8) = \ldots crit(id_{12}) = (1, 2, 20)$.

A.2 System 2

Topology $E = \{e\}$, $M_s = \{c, d\}$, $M_a = \{m_{in}\}$, $A = \{a\}$, $A_T(a) = 1$, $reach(a) = \{e, c, d, m_{in}\}$, $cap(c) = 4$, $cap(d) = 1$, $req(c) = \{2, 3, 4\}$, $req(d) = \{1\}$

Protocols $prot(id_1) = prot(id_2) = \ldots = prot(id_6) = (m_{in}, 1, 10)\cdot(d, 6, 7)\cdot(c, 3, 4)$,
$prot(id_7) = prot(id_8)\ldots prot(id_{12}) = (m_{in}, 1, 34)\cdot(c, 3, 4)\cdot(d, 6, 7)$.

Critical section $crit(id_1) = crit(id_2) \dots crit(id_6) = (0, 1, 20)$ and $crit(id_7) = crit(id_8) = \dots crit(id_{12}) = (1, 2, 20)$.

A.3 System 3

Topology $E = \{e\}$, $M_s = \{c, d, \}$, $M_a = \{m_{in,m}\}$, $A = \{a_1, a_2\}$, $A_T(a_1) = 1$, $reach(a_1) = \{c, d, m_{in}\}$, $A_T(a_2) = 1$, $reach(a_2) = \{m, d, e\}$, $cap(c) = 4$, $cap(d) = 1$, $cap(m) = 1$, $req(c) = \{1, 2, 4\}$, $req(d) = \{1\}$, $req(m) = \{1\}$

Protocols $prot(id_1) = prot(id_2) = \dots = prot(id_6) = (m_{in}, 3, 34) \cdot (c, 1, 2) \cdot (d, 1, 1) \cdot (m, 1, 2)$
$prot(id_7) = prot(id_8) \dots prot(id_{12}) = (m_{in}, 1, 34) \cdot (c, 1, 2) \cdot (d, 1, 2) \cdot (m, 1, 2)$.

Critical section $crit(id_1) = crit(id_2) \dots crit(id_6) = (0, 3, 15)$ and $crit(id_7) = crit(id_8) = \dots crit(id_{12}) = (0, 1, 5)$.

B Result tables

	Time (seconds)							
	System 1				**System 2**			
Instance	MRPT	MRPT-T	RPT	RPT-T	MRPT	MRPT-T	RPT	RPT-T
1	0.10	0.10	0.09	0.10	0.35	0.43	0.25	0.18
2	0.10	0.10	0.9	0.10	2.00	1.99	1.25	0.86
3	0.11	0.11	0.10	0.11	0.54	0.49	0.30	0.20
4	0.10	0.11	0.10	0.10	2.01	2.13	1.57	1.29
5	1.219	0.95	0.54	0.43	T/O	T/O	T/O	100.91
6	0.79	0.78	0.44	0.36	NS	NS	NS	NS
7	0.93	0.82	0.56	0.43	T/O	T/O	T/O	4.92
8	0.82	0.83	0.70	0.66	4.69	4.54	1.95	1.61
9	1.10	0.99	0.93	0.76	NS	NS	NS	NS
10	2.73	2.59	2.41	2.41	154.38	166.38	10.91	2.06
11	2.93	3.02	2.52	2.42	T/O	T/O	19.16	1.60
12	3.46	3.17	2.87	2.67	T/O	T/O	T/O	42.83
13	6.36	5.47	3.21	2.19	NS	NS	NS	NSF
14	7.90	6.84	3.99	2.25	NS	NS	NS	NS
15	12.58	10.00	4.65	2.43	T/O	T/O	T/O	T/O
16	11.41	9.02	4.43	2.13	T/O	T/O	T/O	T/O
17	15.62	12.16	5.33	2.65	T/O	T/O	T/O	T/O
18	21.98	15.41	5.84	2.68	T/O	T/O	T/O	T/O
19	23.90	16.50	6.75	2.67	T/O	T/O	T/O	T/O
20	149.48	103.01	34.30	14.08	T/O	T/O	T/O	T/O
21	126.36	122.13	94.83	94.50	T/O	T/O	T/O	3.63
22	93.66	95.68	73.32	79.28	T/O	T/O	T/O	7.00
23	170.44	166.13	130.67	123.96	T/O	T/O	T/O	97.90
24	179.87	171.64	134.65	129.89	T/O	T/O	T/O	T/O

Table 1: Evaluation of time before running out of memory, exploring all configurations, or synthesising a feasible schedule per IAP instance, system, and heuristic.

	Time (seconds)			
	System 3			
Instance	MRPT	MRPT-T	RPT	RPT-T
1	0.24	0.25	0.21	0.22
2	1.70	1.43	1.40	1.10
3	0.40	0.39	0.34	0.36
4	1.53	1.47	1.41	1.13
5	T/O	T/O	T/O	T/O
6	12.70	12.02	6.05	6.8
7	T/O	T/O	229.33	70.22
8	11.22	11.44	10.51	10.67
9	65.16	61.04	23.52	5.81
10	T/O	T/O	289.01	T/O
11	T/O	T/O	T/O	34.95
12	T/O	T/O	T/O	223.59
13	T/O	T/O	T/O	T/O
14	T/O	T/O	T/O	T/O
15	T/O	T/O	T/O	T/O
16	T/O	T/O	53.67	14.74
17	T/O	T/O	T/O	T/O
18	T/O	T/O	T/O	195.06
19	T/O	T/O	T/O	T/O
20	T/O	T/O	T/O	T/O
21	T/O	T/O	T/O	T/O
22	T/O	T/O	T/O	T/O
23	T/O	T/O	T/O	T/O
24	T/O	T/O	T/O	T/O

Table 2: Time (seconds) for System 3 across different instances and algorithms

	Number of configurations							
	System 1				**System 2**			
Instance	MRPT	MRPT-T	RPT	RPT-T	MRPT	MRPT-T	RPT	RPT-T
1	79	73	60	52	1603	1724	1028	486
2	79	73	61	53	14890	14887	5764	2822
3	143	138	118	111	2262	1879	1012	526
4	163	148	124	110	18131	17400	9810	4614
5	3567	2676	1497	952	T/O	T/O	T/O	623743
6	2929	2377	1512	945	NS	NS	NS	NS
7	2910	2376	1519	943	T/O	T/O	T/O	50513
8	3064	2842	2750	2381	47004	42965	14332	7959
9	3729	3251	3132	2522	NS	NS	NS	NS
10	21995	21000	18605	16046	1075034	1130524	106207	10997
11	24118	21834	19298	15971	T/O	T/O	97527	7500
12	29526	26560	23394	19023	T/O	T/O	T/O	322043
13	57800	46758	26932	11382	NS	NS	NS	NS
14	57824	46504	27046	11326	NS	NS	NS	NS
15	90154	65952	29007	11374	T/O	T/O	T/O	T/O
16	90442	65450	29784	11190	T/O	T/O	T/O	T/O
17	131293	94981	43275	13224	T/O	T/O	T/O	T/O
18	174676	119764	43858	13427	T/O	T/O	T/O	T/O
19	175741	119492	44070	13302	T/O	T/O	T/O	T/O
20	720021	521560	224138	77645	T/O	T/O	T/O	T/O
21	616056	573206	486981	428581	T/O	T/O	T/O	20445
22	561626	542389	568117	427791	T/O	T/O	T/O	52885
23	821671	762216	647055	539765	T/O	T/O	T/O	425247
24	854911	799345	668347	570704	T/O	T/O	T/O	T/O

Table 3: Evaluation of number of configuration searched before reaching the timeout or establishing no solution exists.

	Number of configurations			
	System 3			
Instance	MRPT	MRPT-T	RPT	RPT-T
1	383	377	295	241
2	3771	3314	2961	1684
3	768	763	584	534
4	3401	3421	2844	1553
5	T/O	T/O	T/O	T/O
6	49969	42697	23977	19480
7	T/O	T/O	482660	161257
8	41099	38953	37881	35010
9	189419	180757	83092	16405
10	T/O	T/O	512257	T/O
11	T/O	T/O	T/O	74206
12	T/O	T/O	T/O	322329
13	T/O	T/O	T/O	T/O
14	T/O	T/O	T/O	T/O
15	T/O	T/O	T/O	T/O
16	T/O	T/O	151614	40844
17	T/O	T/O	T/O	T/O
18	T/O	T/O	446267	T/O
19	T/O	T/O	T/O	T/O
20	T/O	T/O	T/O	T/O
21	T/O	T/O	T/O	T/O
22	T/O	T/O	T/O	T/O
23	T/O	T/O	T/O	T/O
24	T/O	T/O	T/O	T/O

Table 4: Evaluation of number of configuration searched before reaching the timeout or establishing no solution exists.

References

[1] SigmaNet ApS. *SigmaNet — IT and automation software*. Accessed: January 8, 2024. 2024. URL: https://sigmanet.dk/.

[2] Y. Sun B. Huang and Y. M. Sun. "Scheduling of flexible manufacturing systems based on Petri nets and hybrid heuristic search". In: *International Journal of Production Research* 46.16 (2008), pp. 4553–4565. DOI: 10.1080/00207540600698878. eprint: https://doi.org/10.1080/00207540600698878. URL: https://doi.org/10.1080/00207540600698878.

[3] MIRYAM BARAD and DANIEL SIPPER. "Flexibility in manufacturing systems: definitions and Petri net modelling". In: *International journal of production research* 26.2 (1988-02). ISSN: 0020-7543.

[4] Olatunde T. Baruwa, Miquel Angel Piera, and Antoni Guasch. "Deadlock-Free Scheduling Method for Flexible Manufacturing Systems Based on Timed Colored Petri Nets and Anytime Heuristic Search". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45.5 (2015), pp. 831–846. DOI: 10.1109/TSMC.2014.2376471.

[5] Michael G.H. Bell. "Hyperstar: A multi-path Astar algorithm for risk averse vehicle navigation". In: *Transportation Research Part B: Methodological* 43.1 (2009), pp. 97–107. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2008.05.010. URL: https://www.sciencedirect.com/science/article/pii/S0191261508000714.

[6] Bernard Berthomieu and Michel Diaz. "Modeling and verification of time dependent systems using time Petri nets". In: *IEEE transactions on software engineering* 17.3 (1991), p. 259.

[7] Thomas H. Cormen et al. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844.

[8] Google. *The Job Shop Problem*. Accessed: February 7, 2024. 2024. URL: https://developers.google.com/optimization/scheduling/job_shop.

[9] Rasmus Høyer Hansen and Daniel Overvad Nykjær. *CatSynth, GitHub repository*. https://github.com/TheRumle/CatSynth [Accessed: June 4. 2024]. June 2024.

[10] Rasmus Høyer Hansen and Daniel Overvad Nykjær. *Timed Multi-Part Manufacturing Systems*. Jan. 2024. URL: https://kbdk-aub.primo.exlibrisgroup.com/permalink/45KBDK_AUB/a7me0f/alma9921650772605762.

[11] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[12] Panagiotis Kouvelis. "Design and planning problems in flexible manufacturing systems: a critical review". In: *Journal of Intelligent Manufacturing* 3.2 (Apr. 1992), pp. 75–99. ISSN: 1572-8145. DOI: 10.1007/BF01474748. URL: https://doi.org/10.1007/BF01474748.

[13] Heiner Lasi et al. "Industry 4.0". In: *Business & information systems engineering* 6 (2014), pp. 239–242.

[14] Doo Yong Lee and F. DiCesare. "Scheduling flexible manufacturing systems using Petri nets and heuristic search". In: *IEEE Transactions on Robotics and Automation* 10.2 (1994), pp. 123–132. DOI: 10.1109/70.282537.

[15] Doo Yong Lee and Frank DiCesare. "Scheduling flexible manufacturing systems using Petri nets and heuristic search". In: *IEEE Transactions on robotics and automation* 10.2 (1994), pp. 123–132.

[16] Hang Lei et al. "Deadlock-free scheduling for flexible manufacturing systems using Petri nets and heuristic search". In: *Computers & Industrial Engineering* 72 (2014), pp. 297–305. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2014.04.002. URL: https://www.sciencedirect.com/science/article/pii/S0360835214001132.

[17] JianChao Luo et al. "Deadlock-Free Scheduling of Automated Manufacturing Systems Using Petri Nets and Hybrid Heuristic Search". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45.3 (2015), pp. 530–541. DOI: 10.1109/TSMC.2014.2351375.

[18] A.R. Moro, H. Yu, and G. Kelleher. "Advanced scheduling methodologies for flexible manufacturing systems using Petri nets and heuristic search". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065).* Vol. 3. 2000, 2398–2403 vol.3. DOI: 10.1109/ROBOT.2000.846386.

[19] Miguel Mujica, Miquel Angel Piera, and Mercedes Narciso. "Revisiting state space exploration of timed coloured petri net models to optimize manufacturing system's performance". In: *Simulation Modelling Practice and Theory* 18.9 (2010), pp. 1225–1241. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2010.04.010. URL: https://www.sciencedirect.com/science/article/pii/S1569190X10000778.

[20] John F Muth, Gerald Luther Thompson, and Peter R Winters. "Industrial scheduling". In: *(No Title)* (1963).

[21] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[22] Shitong Peng et al. "Petri net-based scheduling strategy and energy modeling for the cylinder block remanufacturing under uncertainty". In: *Robotics and Computer-Integrated Manufacturing* 58 (2019), pp. 208–219. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2019.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0736584518303806.

[23] C. A. Petri. "Fundamentals of a Theory of Asynchronous Information Flow". In: *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 - September 1, 1962*. North-Holland, 1962, pp. 386–390.

[24] Tien-Hsiang Sun, Chao-Weng Cheng, and Li-Chen Fu. "A Petri net based approach to modeling and scheduling for an FMS and a case study". In: *IEEE Transactions on Industrial Electronics* 41.6 (1994), pp. 593–601. DOI: 10.1109/41.334576.

[25] Huanxin Henry Xiong and MengChu Zhou. “Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search”. In: *IEEE transactions on semiconductor manufacturing* 11.3 (1998), pp. 384–393.