Supervisory Control of Timed Discrete-Event Systems under Logical and Temporal Specification

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione e del Controllo

Author: **Giosuè Basso**

Student ID: 964037
Advisor: Prof. Luigi Piroddi
Co-advisors: None
Academic Year: 2022-23

Abstract

The supervisory control (SC) of timed discrete events systems (TDESs), modelled using time Petri nets (TPNs), has seen much success employing a novel type of forward reachability graph: the modified state class tree (MSCT). This graph combines the logical and temporal features of the underlying TPN, allowing for a more straightforward check on the specifications. The required specifications regard both the logical aspects -reaching a target marking(s) while avoiding forbidden marking(s)-, and the temporal aspects - arriving and/or departing within a specific time window. The SC approach is based on the formulation of integer linear problems (ILPs), built starting from the MSCT, which are solved to obtain the largest firing time interval (FTI) for each enabled controllable transition. These FTIs ensure that the specifications are met independently of the behaviour of the system, which can include uncontrollable transitions.
This thesis describes and comments a set of algorithms that:

- build the MSCT,
- prune it of the nodes that are not of interest for the control task,
- formulate the ILPs
- calculate, if present, the solution.

Subsequently, a case study is analysed to test the algorithms.

Keywords: Petri Nets, Time Petri Nets, Timed Discrete Event Systems, Supervisory Control, Modified State Class Tree, Partial Modified State Class Tree.

Abstract in lingua italiana

Il controllo supervisivo di sistemi a eventi discreti temporizzati, schematizzati usando reti di Petri temporizzate, ha riscosso successo utilizzando un nuovo tipo di grafo di raggiungibilità: l'albero modificato delle classi di stati. Questo grafo combina le caratteristiche logiche e temporali della rete di Petri temporizzata che modella, semplificando il controllo delle specifiche. Queste specifiche riguardano sia gli aspetti logici -raggiungere una o più marcature obiettivo evitando quelle proibite- che quelli temporali -arrivare (partire) a (da) una marcatura nei limiti di una finestra temporale specificata. L'approccio di controllo supervisivo si basa sulla formulazione di problemi di programmazione lineare, costruiti basandosi sull'albero modificato delle classi di stati, la cui soluzione restituisce la più grande finestra di scatto per ogni transizione abilitata controllabile. Queste finestre di scatto garantiscono che le specifiche vengano rispettate indipendentemente dall'evoluzione del sistema, che può includere transizioni non controllabili.
Questa tesi descrive e commenta un insieme di algoritmi che:

- costruisce l'albero modificato delle classi di stati,
- lo pota dei nodi che non interessano la funzione di controllo,
- formulano i problemi di programmazione lineare,
- calcolano, se esitente, la soluzione.

Successivamente, gli algoritmi sono testati con un esempio informativo.

Parole chiave: Reti di Petri, Reti di Petri temporizzate, Sistemi a eventi discreti temporizzati, Constrollo supervisivo, Albero modificato delle classi di stati, Albero modificato parziale delle classi di stati.

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

Introduction **1**

1 Literature Review and Theoretical Foundations **5**
- 1.1 Overview of Timed Discrete-Event Systems . . . 6
  - 1.1.1 Petri Nets . . . 7
  - 1.1.2 Time Petri Nets . . . 9
- 1.2 Control of TDES . . . 11
- 1.3 State of the Art . . . 12
  - 1.3.1 State Class Graph . . . 12
  - 1.3.2 Modified State Class Graph . . . 13

2 Problem Statement **21**
- 2.1 The Control Specifications . . . 21
- 2.2 The Control Action . . . 22
- 2.3 Control Synthesis Overview . . . 23

3 Methodology and Algorithms **27**
- 3.1 Declaration of the Data . . . 27
- 3.2 Construction of the MSCT . . . 28
  - 3.2.1 Boundedness Assurance . . . 30
  - 3.2.2 Structural Matrices of the MSCT . . . 32
- 3.3 Path-finding Algorithm . . . 40
  - 3.3.1 Remarks about the Breadth-First Search Algorithm . . . 40
- 3.4 Pruning of the MSCT . . . 42

3.5 Path Constraints 44
3.6 Calculation of the Solution 47
3.6.1 Minimisation Problem 48
3.6.2 Maximisation Problem 49
3.6.3 Computation of the FTIs 51

4 Case Study **53**
4.1 Graphical Representation 55
4.2 Results 56
4.3 Pruning of non-Compliant Paths 61

5 Absence of Guaranteed Solutions **63**
5.1 Relaxation of the Temporal Specifications 63

6 Conclusion and Future Work **69**
6.1 Summary of the Thesis 69
6.2 Limitations and Future Research Direction 70

Appendix **73**

Bibliography **79**

Introduction

In the modern world, automation has become an essential driving force behind the optimization and reliability of numerous industrial processes. From manufacturing plants to transportation systems, the seamless coordination of complex operations is critical for achieving optimal performance and ensuring safety. Timed Discrete Event Systems (TDES) provide a powerful framework for modeling and analyzing such dynamic processes, characterized by discrete events occurring in a continuous time domain.
The supervisory control of TDES involves the design and implementation of control strategies to regulate and coordinate the behavior of these systems. It aims to ensure that the system adheres to predefined logical specifications, while also satisfying temporal constraints and optimizing its operational efficiency. Achieving these objectives requires a deep understanding of the underlying theoretical foundations and the development of effective control methodologies. This thesis focuses on investigating and advancing the field of supervisory control of TDES with a specific emphasis on integrating logical and temporal specifications. The integration of logic and time is a fundamental challenge in the domain of automation, as it necessitates the orchestration of system behaviors according to logical constraints and temporal deadlines. The effective synthesis of such control strategies is crucial for achieving safe, reliable, and efficient operations in real-world applications.

The primary objective of this thesis is to explore novel approaches and algorithms that enable the effective supervision and control of TDES by seamlessly combining logical and temporal specifications. By addressing this critical research gap, this work aims to contribute to the advancement of control theory, providing practical solutions for managing complex systems where precise timing and logical coherence are of utmost importance.
To achieve these objectives, the thesis will delve into the theoretical foundations of TDES, drawing upon established frameworks such as *Automata Theory* and the extensive theory about *Petri Nets*, which already includes timed features in the form of *Time Petri Nets*. Additionally, the thesis will cover existing control methodologies and algorithms that have been proposed for supervisory control of TDES, identifying their strengths and limitations.
Furthermore, this research will involve the development and implementation of innova-

tive control strategies, leveraging recent advancements in computer science, optimization techniques, and machine learning. By integrating logical and temporal reasoning into these strategies, this work aims to enhance the capabilities of supervisory control systems, enabling them to effectively adapt to dynamic environments, unforeseen events, and changing operational requirements. More specifically, the goal is to build on top of what is the state of the art concerning the control of TDES, as explained in the work [2]. Namely, provided the control specifications that the system has to abide to, it is this work's task to formulate a set of algorithms that return the control action able to guarantee the compliance of the system to said specifications. In some cases the desired behaviour cannot be guaranteed even though the system allows for feasible solutions. The novel problem that is shaped by the aforementioned situation is of great interest and still not tackled by scientific work. It favours the development of new strategies to handle the absence of assurances, leading to the use of heuristics. The motivation comes from the fact that the objective is not assuring that a solution is reached, but taking the decisions that get the system as close as possible to the desired behaviour. It is an inherently difficult task to treat, both as far as complexity and size of the problem are concerned, so the scope of this thesis is going to be limited to laying the theoretical foundation for the heuristic control of TDES.

The findings of this thesis will have potential practical implications for a wide range of industries, including manufacturing, transportation, power systems, and more.
Through rigorous investigation, experimentation, and analysis, this research endeavors to provide valuable insights and methodologies that contribute to the advancement of the topic at hand. By unraveling the intricacies of this domain, this thesis aims to facilitate the design and deployment of sophisticated control systems that are capable of managing complex operations in real-time, further advancing the capabilities of this field.

Overview of the Thesis Structure

Considering the vastness of the topic that is going to be treated in this work, the first chapter will be dedicated to analysing the preliminary notions that allow to understand the thesis development, providing both the nomenclature and the information needed to navigate it. The aim is to offer a comprehensive literature review that examines the foundational concepts of the field. It delves into the models and formal representations of TDES in the form of *Time Petri Nets*, providing an understanding of the underlying theoretical foundations; Additionally, the theories and approaches of supervisory control, with a particular focus on existing methods and algorithms applicable to TDES, are

explored.
The state of the art is going to be treated next, concentrating on the methodologies that manage to represent timed systems unifying the logical and temporal specifications. The focal point of this section is going to be the *modified state class graph* (MSCG), one of the most powerful tools at our disposal in the study of TPN as it opens the door to a more effective comprehension of the relationship between the logical and temporal constraints that the net is subjected to, even as it evolves.

The second chapter uses the theoretical information of the previous one to precisely and mathematically define the problem at hand.
Firstly a formal description of the specifications is provided in the form of a *generalised timed state sequence* (GTSS), which summarises all of the requirements that the problem needs to fulfill. Secondly, the focus is going to move on how the control action influences the system, distinguishing between the effects that different events have on the system in the framework of control. Namely if they can be prohibited from happening or forced to happen.
Lastly, an overview of the control methodology is discussed, providing the backbone structure for the approach utilised to control the system. Moreover, the strengths and weaknesses of an online and offline approach to the control algorithm are going to be analysed.

Chapter three examines the state-of-the-art approaches, identifying the research gaps and challenges that arise while adapting the theory to a computer program. The developed algorithms are described, providing a step-by-step explanation of the computational techniques employed. These algorithms leverage the MSCG representation, incorporating logical and temporal constraints into the control synthesis process in two separate steps, without loss of generality. The chapter presents the theoretical justifications and algorithmic details, highlighting the novel aspects of the proposed approach that is built on top of existing methods.

In chapter four a case study is considered in order to show the correct functioning of the explained procedures. It consists of a simple material handling system specifically conceived to test and demonstrate the most meaningful aspects of the theory treated up to this point.
The chapter ends with the discussion of the results that have been obtained, putting particular emphasis on the instances where procedures used by the program may differ with respect to the ones studied in [2]. The hope is that this example will provide an all rounded explanation able to convey the inner workings of the algorithms developed in this thesis.

Chapter five delves into the case where the control specifications cannot be guaranteed, nevertheless useful information can be provided. This problem is of particular interest considering how, as to date, it has yet to be tackled by scientific literature.
Different approaches are proposed and discussed, analysing their positive and negative characteristics. Subsequently, the policies that derive from the solutions of the optimisation problems that stem from them are evaluated, understanding the framework and applications they are better suited for.

Chapter six concludes the thesis by summarizing the key contributions and findings of this work. It highlights the significance of the results, emphasizing the potential impact on real-world applications. The chapter also outlines potential future research directions, suggesting avenues for further exploration and improvement.

Finally, the thesis includes an appendix with material of interest and a comprehensive reference section that cites all the sources consulted during the research step.

1 | Literature Review and Theoretical Foundations

In order to understand *time Petri nets*, referred by using the acronym TPN, it is useful to start by talking about *discrete-event systems* and *timed discrete-event systems* first. Discrete event systems (DES) and timed discrete event systems (TDES) are fundamental concepts in the field of control and automation. They provide powerful frameworks for modeling, analysing, and controlling complex systems that operate based on events and the sequencing of these events.

A DES can be defined as a dynamic system that evolves over time through a sequence of discrete events. These events occur at distinct points in time and can trigger changes in the system's state or behavior. Examples of discrete events include sensor activations, button presses, or the arrival of a message in a communication network. The system's behavior is determined by the order and timing of these events, and the interactions between them.
TDESs extend the notion of DESs by incorporating time into the modeling and analysis process. In the TDES framework, events occur within a specified time window. By considering time, TDESs enable the representation and analysis of time-critical systems where the temporal aspects of events and system dynamics are crucial. The modeling and analysis of DESs and TDESs are typically performed using formal methods and mathematical techniques. Various formalisms have been developed to represent and analyze these systems. *Petri nets* (PNs) and *time Petri nets* (TPNs), for DESs and TDESs respectively, are frequently employed formalisms that provide abstractions and mathematical frameworks for capturing the system's structure, behavior, and temporal properties, creating a practical and easily understandable interface with the problem at hand.
The analysis of DESs and TDESs encompasses various aspects, including reachability analysis, performance evaluation, verification of system properties, and synthesis of control strategies. By utilising these analysis techniques, it is possible to gain insights into system behavior, identify potential bottlenecks or issues, optimise system performance, and design effective control strategies.

Considering the scope of this work, only a part of these tools will be discussed.

1.1. Overview of Timed Discrete-Event Systems

A DES is a dynamic system which evolves in accordance with abrupt occurrences of physical events [15]. If the events of the system may occur only within a designated time interval, the framework that needs to be used is that of a TDES. While the addition of timing constraints to the model makes for a more powerful tool, the added complexity is significant.
A model can be developed starting from the 5-tuple

$$G_{act} = (\Sigma_{act}, A, \delta_{act}, a_0, A_m)$$

where Σ_{act} is a finite alphabet of events (list of events that can occur). A is the set of activities, whose elements are indicated by a_i -it is important to remark how, in this interpretation, events are instantaneous while activities have a duration in time-. δ_{act} is the activity transition function $\delta_{act} : \Sigma_{act} \times A \to A$ so, to show the change of activity that a transition implies, the notation is: $a^{'} = \delta_{act}(\sigma, a)$. The initial activity is denoted by a_0 and $A_m \subseteq A$ is the subset of marked activities.
Each transition σ is characterised by a lower time bound and an upper time bound, $l_\sigma \in \mathbb{N}$ and $u_\sigma \in \mathbb{N}$ respectively; the former typically represents a delay while the latter a deadline. In other words a transition can fire only after l_σ time instants have elapsed and after u_σ time instants, if it has not fired, it must do so (if it is still enabled). For this reason it is called a *hard deadline*. Events whose deadline is finite are referred to as *prospective* while those which have an infinite upper bound are defined *remote* events. It is assumed that events occur at quasi-random moments of real time $\mathbb{R}^+ = \{t \ / \ 0 \leqslant t \leqslant \infty\}$ [6].
This modelling paradigm is well-suited to describe a variety of systems, both physical and logical. The simplicity of its premises allows to widen the span of its use resulting in a general set of rules that does not sacrifice precision. This being said, depending on the modelling technique used, there can be notable trade-offs: results may end up being convoluted and difficult to interpret, especially if the dimension of the sets constituting the system is significant.
These theoretical foundations are shared by PNs and their time-dependent counterpart, which are of more interest in the current matter of discussion as these two modelling frameworks are particularly well suited to schematise DESs whose states are asynchronous and/or concurrent.

As far as time dependent nets are concerned, two main methods have been developed to handle the time aspects: *timed Petri Nets* and *time Petri Nets*. The former, described in [16], considers transitions as having a finite firing duration and so they are not to be considered instantaneous. This reflects in a change of the classic PN firing rules: the time it takes for transitions to fire has to be accounted for. Secondly, a transition is set to fire as soon as it gets enabled. Timed PNs have mainly been used for performance evaluation tasks.
Time Petri nets (TPNs) on the other hand have found more use considering the fact that they are more general than timed PNs (indeed, TPNs can be used to model timed PNs, but the opposite is not possible). They are going to be the default modelling paradigm in this work and their description, first published in [11, 12], will be extensively treated shortly in the following sections. In order to understand TPNs it is useful to first analyse the simpler, logic-only version, that are PNs, subsequently moving the focus to TPNs.

1.1.1. Petri Nets

A Petri Net is a place/transition net whose structure is defined by the quadruple $N = (P, T, Pre, Post)$ where: P is a set of m places, T is a set of n transitions and $Pre : P \times T \to \mathbb{N}$ and $Post : P \times T \to \mathbb{N}$ are the matrices that specify how the arcs are connected to the places, using the arcs' weight (the number of tokens that are moved by them). For some applications the *incidence matrix* $I = Post - Pre$ may be of use.
In order to completely identify a PN a *marking* is needed, which represent a specific state of the underlying system that the PN models. It is a vector $M : P \to \mathbb{N}$ that assigns to each place a non-negative integer which represents the tokens present in that place. To identify the marking of a selection of places the notation $M(p)$ is used, where p represents the considered places.
With this preliminary knowledge it is useful to define some important notions for Petri Nets. Transitions represent the occurrence of an event at a certain state of the system, so a transition t is said to be *enabled* at a marking M if the root places of the transition -the places connected to the tail of the transition arrow- have enough tokens for t to fire, namely, indicating with RP the set of root places of t, $M(RP) \geq Pre(RP, t)$. The firing of an enabled transition t in M leads to a new marking $M' = M - Pre(\cdot, t) + Post(\cdot, t)$, which is equivalent to $M' = M + I(\cdot, t)$. To denote the set of enabled transitions in M the expression $T_e(M) = \{t \in T/M \geq Pre(\cdot, t)\}$ is used. The core assumption on which Petri nets base their functioning is that, out of all the enabled transitions at a given marking, the one to fire is selected at random. The framework of PNs enables the control action through *controllable transitions*: a transition t is said to be controllable if

its firing can be prevented while it is enabled. Practically, this inhibition is performed using *enabling places*, places that prevent a transition from firing when they possess no tokens. Consequently, if a transition has an enabling place as part of its Pre matrix it is schematised as controllable.
Conversely, a *non-controllable transition* cannot be prevented from firing once it becomes enabled.
Another key element of the nomenclature of PN is *transition sequences*, shortened to TS: a succession of transitions that can fire in the order expressed by the sequence: $s = t_{i1}\ t_{i2}\ ...\ t_{ik}$. If s is enabled in M_0, then this relationship can be written as $M_0[s\rangle$, whereas if s fires from the same marking it is denoted as $M_0[s\rangle M_k$ with M_k being the last marking reached firing s. An interesting property of the firing of a transition sequence is that the final marking can be computed using a single equation: $M' = M + I\sigma$ where $\sigma = \sigma(s)$ is the so called *firing count vector* (FCV) associated to s. To be more specific the j-th element of σ is the number of times that transition j fires in s. A FCV is said to be *admissible* if it admits at least an enabled transition sequence.
A Petri net is said to be *live* if every marking belonging to the net allows to fire every transition of said PN after a suitable transition sequence.
A marking M is said to be *reachable* in $\langle N, M_0\rangle$ if there exists a TS enabled in M_0 that ends in M, $M_0[s\rangle M$; the set of all the markings reachable from M_0 is called the *reachability set* of $\langle N, M_0\rangle$ and is referred to as $R(N, M_0)$. Another important characteristic of Petri Nets is *boundedness*; a place $p_i \in P$ is said to be bounded if and only if p_i has a finite number of tokens in all the reachable markings of the net, namely $\exists k > 0 | M(p_i) \leq k, \forall M \in R(N, M_0)$. A Petri net is bounded if and only if all its places are bounded.
If it exists, a vector $y > 0$ such that $Iy = 0$ is called a *T-invariant*; it is trivial to understand that a transition sequence whose firing count vector is a T-invariant results in a null net marking modification, $M' = M$, taking the PN back to the initial marking.
A key feature of Petri nets is that they can easily be represented graphically. Being a higher level representation technique, a graph ensures a more direct and immediate comprehension, emphasising the way transitions connect places and conveying a general idea of the structure of the system being analysed.

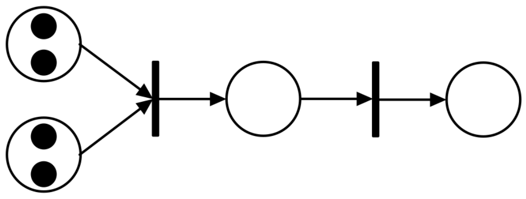


Figure 1.1: A simple example of a marked Petri Net.

Places are represented as circumferences which may contain tokens in the form of black circles; transitions are depicted as orientated arrows with a perpendicular segment in the middle where multiple transitions can be joined together. They may have a number indicating the weight of the transition which, if absent is defaulted to 1, and a uniquely identifying label. [2]

1.1.2. Time Petri Nets

All of the previous considerations are inherited by time PNs too, with some necessary integration that will be discussed shortly. A net of this kind is defined using the couple $N_d = (N, Q)$ where $N = (P, T, Pre, Post)$ is a generic PN and $Q : T \to \mathbb{Q} \times (\mathbb{Q} \cup \{\infty\})$ defines the set of static closed intervals for each transition of the net. More specifically, considering $t_i \in T$, function Q assigns a lower bound $l_i \in \mathbb{Q}$ and an upper bound $u_i \in \mathbb{Q}$ which can also be infinite. Namely $Q(t_i) = (l_i, u_i)$, where $l_i \geq 0$, $u_i \geq l_i$, $l_i \neq \infty$. This means that transition t_i may fire if it has been uninterruptedly logically enabled for a minimum of l_i time units and it must fire after being uninterruptedly logically enabled for u_i time units. A *TPN system* consists of the pairing of a TPN N_τ with an initial marking M_0 at the initial time instant $\tau = 0$, written shortly as $\langle N_\tau, M_0 \rangle$.
A transition is said to be enabled if the preceding place has enough tokens for it to fire and the current time instant is inside the boundaries of the transition's time window. Sometimes it can be useful to indicate the transitions that are only logically enabled and this is achieved using the set $\mathcal{A}(M) = \{t \in T / M \geq Pre(\cdot, t)\}$.
Transitions are distinguished between *controllable transitions* and *non-controllable transitions*, the former type can be prevented from firing until a prescribed moment is reached -its upper bound-, respecting the conditions imposed by the time constraints. The latter type behaves just like uncontrollable transitions in untimed Petri nets. The presence of controllable transitions is necessary for the system to be controllable as the control action revolves around setting the time windows of controllable transitions such as to impose

a certain behaviour, independent of the randomness that characterises non-controllable transitions.
A *time-transition sequence* (TTS) corresponds to a transition sequence, the main difference being that it consists of a sequence of pairs: the transition that fired and the time instant it fired at. They are expressed analogously to the untimed case through $s_\tau = (t_{i1}, \tau_1)(t_{i2}, \tau_2)...(t_{ik}, \tau_k) \in (T \times \mathbb{R}_0^+)^*$, where τ_j is the time of firing of t_{i_j} $(j = 1, 2, ..., k)$ with $\tau_1 \leq \tau_2 \leq ... \leq \tau_k$. A TTS s_τ enabled in M_0 is denoted $M_0[s_\tau\rangle$ while the firing is $M_0[s_\tau\rangle M_k$ where M_k is the final marking reached by s_τ. This implicitly changes the definition of reachability, which has to account for time conditions as well: given a net $\langle N_\tau, M_0\rangle$, a generic marking M is said to be reachable if there exist a TTS such that $M_0[s_\tau\rangle M$. Equivalently to their time-independent counterpart, the set of all reachable markings from M_0 is called the *timed reachability set*, referred to by the expression $R_\tau(N_\tau, M_0)$. It is easy to conclude that $R_\tau(N_\tau, M_0) \subseteq R(N, M_0)$ where N is the untimed version of N_τ, considering that the addition of time specifications may result, in the most permissive hypothesis, in no loss of reachability. As a consequence the purely logical reachability is a *necessary condition* to have timed reachability.
The notion of *boundedness* does not change for TPNs, remarking the fact that the boundedness of the untimed counterpart is a *sufficient condition* for the boundedness of $\langle N_{tau}, M_0\rangle$, [9].

Theoretical Assumptions

It is important to explicitly remark the theoretical assumptions that have been chosen for this study of TPNs. Clearly, this work being the continuation of [2], the assumptions are shared.
The first assumption is that of *single server semantics*, which means that, independently of the enabling degree (how many times a single transition would be able to fire at a given marking), only one transition can fire at a given time instant; hence the operations that transitions represent must be executed one at a time by a single operation unit or server [4]. Thus, this assumption only mildly restricts the operations that can be modeled considering how transitions fire instantly and concurrent states can be added as a workaround to represent parallel operations.
Secondly, the *enabling memory policy* states that transitions have no memory of any previous enabling. This is to say that a transition that first gets disabled by the firing of another transition and then becomes enabled again has no memory of the previous enabling, which in turn results in its time-tracking timer restarting from 0, [17].
The final assumption can be disregarded for what concerns the work of this thesis, but it

is stated anyway for the sake of completion. It is assumed that the system cannot execute idle loops in 0 time. This would create infinite trajectories taking 0 time to traverse. Again, the limitations this implies are minimal as systems can be adapted in order to exclude the aforementioned kind of loops.

1.2. Control of TDES

Let us now review the theory concerning the control of TDES, as described in [6], where a general approach is proposed, essential to the understanding of supervisory control as a whole.

In order to apply TDESs as models for supervisory control problems, it is crucial to define how the transitions of TDES can be controlled by an external agent or supervisor. From a theoretical standpoint, it is natural and advantageous to establish two criteria for the control approach:

1. control should, at most, restrict uncontrolled behavior and never expand it
2. controlled behavior, while adhering to a specification constraint, should allow for optimization by maximizing permissiveness.

Drawing from the concepts presented in [14] and [18], the initial focus is posed on identifying the counterpart of the so called controllable events, which refers to transitions that can be disabled. In essence, the fact that an event can be disabled implies that it can be indefinitely prevented from occurring. Keeping in mind the first criterion, this suggests that only remote events may fall into this category. If a potential event were disabled, it could be prohibited from occurring even when it becomes imminent, and no competing event is eligible to preempt it i.e. fire before it. The behaviour emerging from such a scenario would only be justified by the presence of a control action.

To better address these considerations a new subset is introduced, denoted as $\Sigma_{hib} \subseteq \Sigma_{rem}$, which denotes the set of *prohibitible events*. The control mechanism allows the supervisor to remove a prohibitible event from the current list of eligible transitions at a specific state S within the supervised TDES. Naturally, just like in the original model, the erased event can be reinstated if and when G revisits state S later on.

In a timed environment another category naturally arises, i.e. that of *forcible events*. They belong to a new subset $\Sigma_{for} \subseteq \Sigma_{act}$ which comprises the events that can be forced to happen, a sort of opposite concept with respect to prohibitible events.

It should be noted how there is no defined relation between Σ_{for} and any of Σ_{hib}, Σ_{rem} or Σ_{spe}, so an event in Σ_{rem} can belong both to Σ_{for} and Σ_{hib}.

Another useful concept to define is that of *uncontrollable events* and the set Σ_{unc} they are part of. More specifically

$$\Sigma_{unc} := \Sigma_{act} - \Sigma_{for} = \Sigma_{spe} \cup (\Sigma_{rem} - \Sigma_{hib})$$

is the set of eligible events that are impossible to erase by the control action. As a direct consequence the set of *controllable events* Σ_{con} can be defined as:

$$\Sigma_{con} := \Sigma - \Sigma_{unc}.$$

It should be noted how a forcible event can be controllable or uncontrollable: an uncontrollable forcible event cannot be directly prevented from occurring by means of disablement (for instance the event of a plane landing can be delayed but not prevented indefinitely from happening).
By the hypothesis formulated above, it is possible to describe the *supervisor* of a system G as the decision maker that, at a given node n, selects a nonempty subset of eligible events that respect the precedent assumptions.

1.3. State of the Art

Recently, research regarding TDESs focused on the development of hybrid models that combine the logical and temporal aspects of the systems to be studied.
It is useful to remark the notation that will be used from now on regarding graphs. A graph G is composed of a set of nodes $N = n_1, n_2, ..., n_i$ and a set of edges $E = e_1, e_2, ..., e_k$, namely $G = (N, E)$. Each edge connects two nodes, $e(n_x, n_y)$ where, in the case of oriented graphs which is of interest for this thesis, n_x indicates the node connected to the tail of the arrow representing the edge while n_y indicates the node connected to the head of said arrow.

1.3.1. State Class Graph

A cornerstone regarding the aforesaid subject is presented in [5]. The paper presents an enumerative analysis technique which conjunctively schematises the behaviour and analyses the properties of a time-dependant system. The cited work takes inspiration from [10], where a reachability analysis methods for untimed PN is proposed.
Novel elements are also presented: the *state* and the *state class* of a TPN, indispensable tools to move the framework to a shared paradigm between the timed and logical

specifications. A state class, also referred to as class, of a time PN is described as the pair $C = (M, \Theta)$, where M is a reachable marking and Θ represents the firing intervals, or domain, of the enabled transitions at that particular marking. Θ can be seen as a $n \times 2$ matrix where row i indicates the i-th enabled transition at marking M and the two columns contain the lower and upper bounds of the same transition i. The state of a TPN is $S = (M, \gamma)$ where M identifies a reachable marking and γ is the analogous of Θ, with the key difference that it keeps track of the amount of time elapsed since the transitions in it got enabled. It is easy to notice that for a given class, the number of states is infinite, as an infinitesimal change of time technically constitutes a new state. For this reason a state can be conveniently seen as an "instance" of the class it belongs to.
A *state class graph* (SCG) is made of nodes that represent classes and edges that connect said classes. Edges also represent the transition that fired from the root node to the new one. Using a bounded SCG -considering a finite part of a SCG in case it is made of infinite nodes and edges- of a TPN enables to more clearly check the behaviour of the system that the TPN represents.
While undoubtedly being groundbreaking work, some shortcomings are present; the aim of this graph is more to illustrate how the underlying TPN works and less to actively act on it. The absence of a framework to describe the evolution of the temporal bounds of the transitions limits the depth of the analysis, especially regarding control, that can be performed on the considered net. For these reasons the concept of SCG has been enhanced, as described in the following section.

1.3.2. Modified State Class Graph

In [1, 3] the SCG is revised and takes the form of the *modified state class graph* or MSCG. There are two main differences that define the MSCG with respect to the SCG:

1. The introduction of labels associated with transitions, which is a relatively minor addition that manages to summarise the time characteristics of the transitions that fire.

2. The introduction of timing variables and constraints associated with the edges of the net.

These changes enable to estimate the system's state at a specific time instant τ based on a timed observation consisting of labeled occurrences at specific time points. The concept underlying the definition of the MSCG revolves around representing the firing domain within a class by incorporating the firing intervals of transitions entering that class. Each arc exiting or entering a class is associated with a variable. By introducing these modifications to the graph, the number of nodes in the MSCG is always equal to or greater than the number of nodes in the SCG. However, the property of finiteness for bounded TPNs is preserved.

The notion of class previously introduced is also the foundation for the MSCG. The graph allows to condense all the states that share a marking into a single class which constitutes a node of the MSCG. The MSCG framework enables the use of better suited functions for the study and manipulation of the underlying net. It should be pointed out that, for the scope of the treatment at hand, the nets that are to be used are unlabeled, i.e. the edges of the MSCG are labeled just with t_i, the transition that fired, and a constraint on the time identified by the so called *firing time variable* (FTV) Δ_i to go from the node connected to the tail of the edge to the one connected to the head of said edge. More specifically, indicating with M_{root} the marking of the parent node, said constraint takes the form $\Delta_i \in [l_i^*,\ u_i^*], \forall t_i \in \mathcal{A}(M_{root})$. No additional elements are added to the label of edges. All things considered, it is a minor simplification that results in no loss of generality or precision.

To be more exhaustive let us consider a TPN system $\langle N_\tau, M_0 \rangle$, where $N_\tau = (N, Q)$ is a time PN and M_0 is its initial marking. The MSCG is an oriented graph whose nodes are the classes created based on the TPN characteristics, namely, as described previously, a unique combination of a reachable marking $M \in R_\tau(N_\tau, M_0)$ and the set of inequalities stemming from the enabled transition at that marking, the so called *domain* of the class. Two classes are said to be *equivalent* if they have the same marking and domain. It is good practice to label the edges of the MSCG as follows: $(t, \Delta \in [l^*, u^*])$, where $t \in T$ is the transition that fired, leading to the class pointed by the arrow of the edge and $\Delta \in [l^*, u^*]$ is the FTV that indicates the time that elapsed from the enabling to the firing of t. The notation used to indicate the transition from a class C_k to the class C_j by the firing of t_i is $C_K[t_i > C_j$, while, to refer to the preceding or successive classes of C_k, $\bullet C_k$ and $C_k\bullet$ are used respectively, [2, 3].

Construction of the MSCG

The construction of the MSCG has been thoroughly studied and analysed in past literature, notably in [3] and [2], and so the algorithms that implement this procedure have already been proven to be theoretically sound. Nonetheless, a small summary of the aforementioned work is of order.

The first step in building a MSCG is to create the initial class which is characterised by the initial marking M_0 and the set of inequalities associated to it at the initial time instant

$\tau = 0$, which are referred to as Θ_0. Starting from the initial class the algorithm iteratively explores new classes and builds the domain corresponding to the enabled transitions of their marking, getting new reachable markings as a result. Flags are used to keep track of new and explored nodes. It is important to notice how not all logically enabled transitions of a given class may be able to fire. As a matter of fact transitions whose lower bound is greater than the minimum upper bound at that specific class cannot possibly fire.
The algorithm shown below was formulated in [3] and summarises the steps that are needed to build the MSCG.
Some remarks about the pseudo-code are of order. At **line 1** the first class C_0 is created and tagged; **line 5** represents a critical step as it ensures that only the transitions that can actually fire are taken into consideration, while simultaneously imposing that the lower bound is always greater or equal than 0. Next, at **line 6** the marking reached via the firing of t_i is defined as M_q. Subsequently the computation of the domain is performed, which is going to change based on the fact that the considered transitions have just been enabled or were already enabled in the previous class (**lines 8-13**). Finally at **line 14** the subsequent node (class) is created and the edge connecting it to its predecessor is labeled. This label contains the transition that fired, an optional tag for it, and the time elapsed since said transition got enabled (with the corresponding time interval).

Algorithm 1.1 Construction of the MSCG

input: A labeled TPN System

output: The corresponding modified state class graph

```
1: Initialisation: The root node $C_0$ is labeled with the initial marking $M_0$ and the corresponding set of inequalities $\Theta_0$ defined as follows: $\forall t_i \in \mathcal{A}(M_0)$ let $l_i^0 \leq \theta_i \leq u_i^0$ where $l_i^0 = l_i$ and $u_i^0 = u_i$. Tag the root node as "new".
2: while a node tagged "new" exists do
3:   Select a node $C_k$ tagged "new".
4:   for all $t_i \in \mathcal{A}(M_k)$ do
5:     if $max\{0, l_i^k\} \leq min_{j:\ t_j \in \mathcal{A}(M_k)}\{u_j^k\}$ where $l_i^k$ $(u_j^k)$ is the lower (upper) bound associated with $t_i$ $(t_j)$ at class $C_k$ then
6:       Let $M_q = M_k + I(\cdot, t_i)$ be the marking reached from $M_k$ firing $t_i$.
7:       for all transitions $t_r \in \mathcal{A}(M_q)$ do
8:         if $t_r \in \mathcal{A}(M_k)$ i.e. $t_r$ was already enabled at class $C_k$ and $M_k - Pre(:, t_i) \geq Pre(:, t_r)$ then
9:           let $l_r^q = l_r^k - \Delta_i, \quad u_r^q = u_r^k - \Delta_i$
10:        else
11:          let $l_r^q = l_r, \quad u_r^q = u_r$
12:        end if
13:      end for
14:      Add a new node $C_q$ labeled with marking $M_q$ and a set of inequalities $\Theta_q$
15:      defined as follows:
16:      $\forall t_r \in \mathcal{A}(M_q)$, let $max\{0, l_r^q\} \leq \theta_r \leq u_r^q$.
17:      Add an edge from $C_k$ to $C_q$ labeled:
18:      "$t_i$, $\mathcal{L}(t_i)$, $\Delta_i \in [max\{0, l_i^k\}\ , min_{j:\ t_j \in \mathcal{A}(M_k)}\{u_j^k\}]$".
19:      if There already exists a node equivalent to $C_q$ in the tree then
20:        Tag node $C_q$ as "duplicate".
21:      else
22:        Tag node $C_q$ as "new".
23:      end if
24:    end if
25:  end for
26:  Untag node $C_k$
27: end while
```

Example

Let us now consider the example used in [1, 3] to show how the construction of the MSCG works.

The TPN to be analysed is the following:

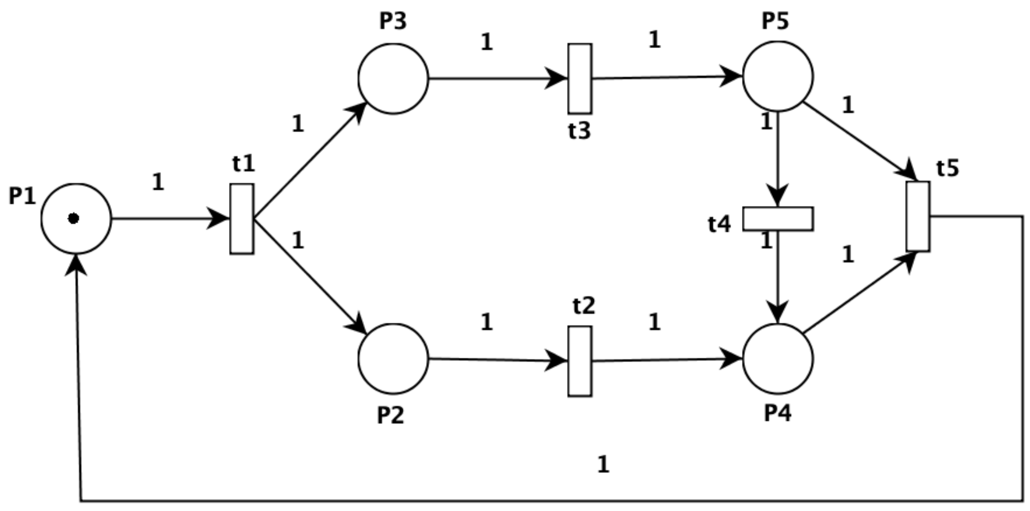


Figure 1.2: The labeled TPN of the example.

Where the initial marking is $[1\ 0\ 0\ 0\ 0]^T$ and the transitions have the following characteristics:

Transition	Time window	Label
t1	(0, 1)	a
t2	(0, 2)	ε
t3	(1, 3)	ε
t4	(1, 5)	ε
t5	(2, 3)	b

Table 1.1: Characteristics of the Transitions of the TPN.

It should be specified that ε denotes the absence of a label for the transition, implying, for instance, that there is not a sensor that detects the associated event when it happens.

The MSCG obtained by the application of **Algorithm 1.1** is:

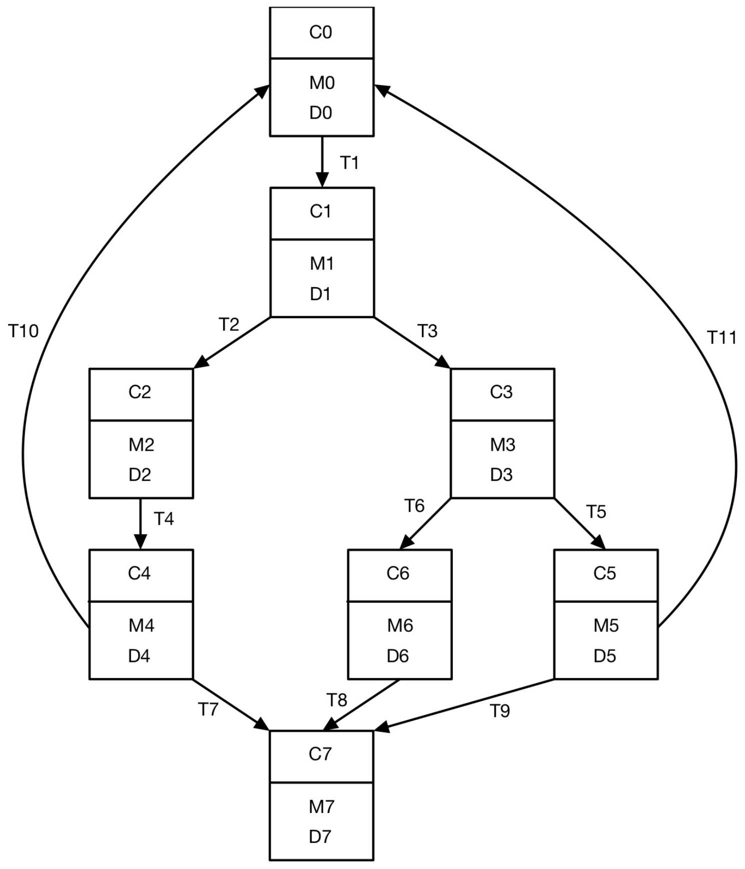


Figure 1.3: The MSCG of the example.

where the labels of the net elements are summarised in the following tables.

Class	Marking	Domain
C0	[1 0 0 0 0]	$0 \leq \theta_1 \leq 1$
C1	[0 1 1 0 0]	$0 \leq \theta_2 \leq 2$ $1 \leq \theta_3 \leq 3$
C2	[0 0 1 1 0]	$0 \leq 1 - \Delta_2 \leq \theta_3 \leq 3 - \Delta_2$
C3	[0 1 0 0 1]	$0 \leq 0 - \Delta_3 \leq \theta_2 \leq 2 - \Delta_3$ $1 \leq \theta_4 \leq 5$
C4	[0 0 0 1 1]	$1 \leq \theta_4 \leq 5$ $2 \leq \theta_5 \leq 3$
C5	[0 0 0 1 1]	$0 \leq 1 - \Delta_2 \leq \theta_4 \leq 5 - \Delta_2$ $2 \leq \theta_5 \leq 3$
C6	[0 1 0 1 0]	$0 \leq 0 - \Delta_3 - \Delta_4 \leq \theta_2 \leq 2 - \Delta_3 - \Delta_4$
C7	[0 0 0 2 0]	$\emptyset$

Table 1.2: Labels of the nodes of the MSCG.

Edge	Fired transition	Label	FTV window
T1	t_1	a	$\Delta_1 \in [0, 1]$
T2	t_2	ε	$\Delta_2 \in [0, 2]$
T3	t_3	ε	$\Delta_3 \in [1, 2]$
T4	t_3	ε	$\Delta_3 \in [\max\{0, 1 - \Delta_2\}, 3 - \Delta_2]$
T5	t_4	ε	$\Delta_4 \in [1, \min\{2 - \Delta_3, 5\}]$
T6	t_2	ε	$\Delta_2 \in [\max\{0, 0 - \Delta_3\}, \min\{2 - \Delta_3, 5\}]$
T7	t_4	ε	$\Delta_4 \in [1, 3]$
T8	t_2	ε	$\Delta_2 \in [\max\{0, 0 - \Delta_3 - \Delta_4\}, 2 - \Delta_3 - \Delta_4]$
T9	t_4	ε	$\Delta_4 \in [\max\{0, 1 - \Delta_2\}, \min\{5 - \Delta_2, 3\}]$
T10	t_5	b	$\Delta_5 \in [2, 3]$
T11	t_5	b	$\Delta_5 \in [2, \min\{5 - \Delta_2, 3\}]$

Table 1.3: Labels of the edges of the MSCG.

It is useful to point out how to interpret the labels of the elements that constitute the graph. Let us consider the instance of $T6$: from class $C3$, marked with [0 1 0 0 1], transition t_2 fires, leading to class $C5$, marked with [0 0 0 1 1]. The FTV corresponding to $T6$ must be in the time window $[\max\{0, 0 - \Delta_3\}, \min\{2 - \Delta_3, 5\}]$ considering how the domain of $C3$ is structured.
It is clear how treating the control of a TPN by the means of its MSCG results in remarkable advantages. Time dependencies are explicitly expressed, allowing for the seamless tracking of both the logical and temporal evolution of the system.

2 | Problem Statement

Having acquired the necessary theoretical foundation, it is now possible to properly state the problem that this work addresses. More specifically, considering the substantial overlap with the problem treated in [2], the same formulation for the problem can be used. In order to provide an exhaustive description, it is divided into three sections: *Control Specifications*, *Control Action* and finally an overview of the *Control Synthesis*; this ensures that a detailed outlining of the problem is laid out, which in terms conveys a clearer image of the requests that need fulfilling, subsequently providing a strong framework to base the solution on.

2.1. The Control Specifications

It can be assumed that, for any given problem of the kind that this work aims to address, the provided information consists of a *Time Petri Net System.* This incorporates, as said in the previous chapter, a marked TPN, which describes the logical relationships between the events that are being modelled and the states of the system in the form of marked places, in conjunction with the time windows for each transition that constitutes the system. In addition to these components, further data are required to formulate the *Control Specifications* that are to be fulfilled by the solution returned by the program. These are: the *Target Marking Set* $\mathcal{L}_i \subseteq R_\tau(N_\tau, M_0)$, namely a non-empty set of reachable markings; the *Set of Forbidden Markings* $\mathcal{F}_i \subseteq R_\tau(N_\tau, M_0)$ which, as the name suggests, must be avoided throughout the evolution of the system and that can be empty. Lastly, the *Absolute Departure Time Interval* (ADT) $I_i^D = [l_i^D, u_i^D]$ with $l_i^D \in \mathbf{R}_0^+ \cup \{\infty\}, u_i^D \in \mathbf{R}_0^+ \cup \{\infty\}$, which defines the allowed time span for leaving $\mathcal{L}_i$, and the *Absolute Arrival Time Interval* (AAT) $I_i^A = [l_i^A, u_i^A]$ with $l_i^A \in \mathbf{R}_0^+, u_i^A \in \mathbf{R}_0^+ \cup \{\infty\}$, which defines instead the time window in which $\mathcal{L}_i$ is to be reached.
These specifications can be conveniently condensed into one formulation, the so called *Generalised Timed State Sequence* (GTSS), which takes the form of a sequence of 4-tuples $(\mathcal{L}_i, \mathcal{F}_i, I_i^A, I_i^D)$. To be more specific, the initial 4-tuple is slightly different as it is composed of the initial marking $\mathcal{L}_0$, the initial time instant τ_0 and a departure time

interval I_0^D and so it can be represented with $(\mathcal{L}_0, \emptyset, \tau_0, I_0^D)$. It is useful to point out how a path to a target marking can have intermediate points that effectively divide it into smaller paths, allowing for a more precise statement of the specifications. An n-step GTSS takes the form of:

$$g = (\mathcal{L}_0, \emptyset, \tau_0, I_0^D)\ (\mathcal{L}_1, \mathcal{F}_1, I_1^A, I_1^D)\ ...\ (\mathcal{L}_n, \mathcal{F}_n, I_n^A, I_n^D)$$

where $\mathcal{L}_0 = \{M_0\}$ represents the initial marking, τ_0 is the initial time instant and I_0^D specifies the initial ADT interval. In order for the GTSS to be *consistent* - and therefore executable - the upper bounds need to be greater than the lower bounds and so $l_i^A \leq l_i^D$, $u_i^A \leq u_i^D, i = 1,\ ...,\ n$, and $l_{i-1}^D \leq l_i^A$, $u_{i-1}^D \leq u_i^A$, $i = 1, ..., n$ must be verified.

2.2. The Control Action

Controlling a TPN bears little to no resemblance to controlling continuous or discrete time physical systems. The "degrees of freedom" on which to act consist of the decision on when to fire the controllable transitions, in accordance to the time windows prescribed by the net. This control action goes beyond the disablement of controllable transitions until their deadlines are met -practically preventing them from firing-, it can also involve partially disabling a transition for a portion of its allowable Firing Time Interval (FTI). In order to express this partial disabling, a time-varying control function is introduced. This function acts as a restriction on the firing regions associated with the enabled controllable transitions in the current marking, allowing for the implementation of partial disabling actions.
Considering a generic state of the TPN at hand $S_k = (M_k, \Phi_k)$, the control action is defined as a time-dependent function of the system state: $\mathcal{F}(S_k, \tau_k) = \hat{\Phi}_k$, where

$$\hat{\Phi}_k = \{\hat{l}_i^k \leq \phi \leq \hat{u}_i^k, \forall i \in I_e(S_k)\},$$

where the hat-variables are the new time bounds defined by the control function $\hat{\Phi}$ at the state S_k and time instant τ_k, and $I_e(S_k)$ identifies all the enabled transitions at state S_k. The expressions $\hat{l}_i^k \geq l_i^k$ and $\hat{u}_i^k \leq u_i^k$ need to be verified if the considered transitions are controllable, as the control window can at most be as wide as the originally allowed time window; conversely, the time bounds of uncontrollable transitions cannot be effected by the control action.

Two types of events can be distinguished:

- *Prohibitible Events*: if $\hat{l}_i^k = \hat{u}_i^k = \infty$ for a transition t_i, then the control action completely disables t_i.
- *Forcible Events*: if $\hat{l}_i^k = \hat{u}_i^k = \delta$ for a transition t_i, then t_i must fire exactly at time $\tau_k + \delta$.

By applying the control law $\mathcal{F}$ to a TPN system $\langle N_\tau, M_0 \rangle$ a system denoted by $\langle N_\tau, M_0, \mathcal{F} \rangle$ is obtained. Naturally, the set of reachable markings of the new system will be a subset of the reachable markings of the uncontrolled net, namely: $R_\tau(N_\tau, M_0, \mathcal{F}) \subseteq R_\tau(N_\tau, M_0)$.

2.3. Control Synthesis Overview

In order to compute the control action in a generic state S_k, a worst-case scenario is assumed. This means that the FTIs of the enabled controllable transitions must be defined in a way that ensures that the GTSS can be satisfied for all possible firings of uncontrollable transitions. This guarantees that the system can always follow a legal trajectory, provided that the controllable transitions enabled in S_k are fired within their prescribed FTIs, and subsequent firings of controllable transitions meet the calculated conditions at τ_k.

This approach accounts for all possible system evolutions in the computation of the control action. To perform the search for all TTSs that move the system from one marking to another, the *modified state class tree* (MSCT) of the system (equivalent to the MSCG but without loops) is used. This choice aims at simplifying the construction of the graph, since loops are computationally intensive to identify, as will be highlighted exhaustively in the following chapter. Computing the full MSCT is often demanding and unnecessary, especially since only a smaller portion of the MSCT is relevant to the given GTSS. To address this, a *partial MSCT*, shortened to PMSCT, is constructed. The core difference is that the PMSCT encompasses only and all legal TTSs, and, by doing so, the computational burden associated with generating and processing the MSCT is significantly reduced. The allowed FTIs of the controllable transitions are then calculated based on this graph.
To be more specific, a step-by-step procedure to calculate the control function in any given state is summarised below; this procedure takes inspiration from the one introduced in [2], adapting the parts that required a change in the approach.

Here are the four steps:

- **Step 1**. A first MSCT is constructed while simultaneously defining it mathematically. This step ends either if the target marking is reached or if the time condition of the GTSS is breached.
  All the classes of the MSCT have their flag initialised as forbidden.
- **Step 2**. The complete MSCT is analysed and the paths leading to the target markings are calculated. These paths are all logically legal as they already belong to the MSCT of the underlying net.
  The classes that belong to at least one trajectory get their forbidden flag removed.
- **Step 3**. The PMSCT is obtained by pruning the MSCT previously calculated, i.e. by removing all the classes flagged as forbidden (and the arcs connecting them). The mathematical definition of the MSCT is also adapted to reflect the PMSCT. Additional constraints are added so that the evolution of the system remains within the PMSCT.
- **Step 4**. The mathematical definition of the PMSCT is used to create an optimisation problem from which the admissible FTIs are calculated for each path. If all paths allow for an FTI to be calculated, then the intersection of the intervals is the final solution. Otherwise, the paths that do not admit a solution are flagged and discarded, and step 3 and 4 are iterated.

A crucial consideration concerning when the control function should be invoked is of order.
Since the control action cannot influence the firing of uncontrollable transitions, it is computed only when the system enters a class where at least one controllable transition is enabled. As long as the system remains in the same class, time continues to pass without any changes in the constraints on the controllable transitions. Hence, there is no need to invoke the procedure again until a transition actually fires.
Although it may seem logical to anticipate the computation of the control action before the actual enabling of a controllable transition, this approach is not pursued here due to the tendency to produce over-conservative conditions. As a matter of fact, prior to the firing of an uncontrollable transition, one must consider the possibility of said transition firing at any point in time within its corresponding FTI. However, once the transition has fired, the remaining portion of its FTI becomes irrelevant. Consequently, the constraints taken into account during the preceding computation of the control action can be relaxed,

potentially resulting in larger FTIs for the controllable transitions that still need to be fired in order to fulfill the GTSS. This justifies the recalculation of the control action after the firing of any uncontrollable transition, as long as at least one controllable transition is enabled in the resulting state.

Algorithm 2.1 outlines how the control function is computed. This requires a TPN in addition to its initial state and time and the prescribed GTSS. Whenever a solution is present, the algorithm follows the evolution of the system and recalculates the FTIs of the enabled controllable transition each and every time a class changes. Once the GTSS is completed the process stops.
The updated constraints are then fed to the controller that will actually decide which controllable transition to fire and when.

Algorithm 2.1 Online Approach for the Control Algorithm

Input: $\langle N_\tau$, M_0, $S_0 = (M_0, \Gamma_0)$, τ_0, g.

Output: The FTI for the Controllable Transitions.

```
1: $S_{curr} = (M_{curr}, \Gamma_{curr}) := S_0$; $\tau_{curr} := \tau_0$
2: **for all** $t_i \in T_e(M_{curr})$ **do**
3:   **if** $t_i \in T_c$ **then**
4:     compute $\mathcal{F}_t(S_{curr}, \tau_{curr}) = [\hat{l}_i^{curr}, \hat{u}_i^{curr}]$;
5:     **if** *a solution does not exist* **then**
6:       **exit**;
7:     **end if**
8:   **else**
9:     $\hat{l}_i^{curr} = l_i^{curr}$; $\hat{u}_i^{curr} = u_i^{curr}$;
10:  **end if**
11: **end for**
12: Feed $\mathcal{F}_t(S_{curr}, \tau_{curr})$ back to the controller;
13: **while** *$g$ is not completed* **do**
14:   **if** $obs = (T_{obs}, \tau_{obs}) \neq \emptyset$ i.e. a new set of events is observed **then**
15:     $S_{prev} = (M_{prev}, \Gamma_{prev}) := S_{curr}$;
16:     $\tau_{prev} := \tau_{curr}$;
17:     $\sigma_{obs} := \sigma(T_{obs}$;
18:     $M_{curr} := M_{prev} + C\sigma_{obs}$;
19:     $\tau_{curr} := \tau_{obs}$;
```

Online Approach for the Control Algorithm

```
20:         for all $t \in T_e(M_{curr})$ do
21:             if $t \in T_e(M_{prev}) \setminus T_{obs}$ then
22:                 $\Gamma_{curr}(t) := \Gamma_{prev}(t) + (\tau_{curr} - \tau_{prev})$;
23:             else
24:                 $\Gamma_{curr}(t) := 0$;
25:             end if
26:         end for
27:         $S_{curr} := (M_{curr}, \Gamma_{curr})$;
28:         for all $t_i \in T_e(M_{curr})$ do
29:             if $t_i \in T_c$ then
30:                 compute $\mathcal{F}_t(S_{curr}, \tau_{curr}) = [\hat{l}_i^{curr}, \hat{u}_i^{curr}]$;
31:             else
32:                 $\hat{l}_i^{curr} = l_i^{curr}$; $\hat{u}_i^{curr} = u_i^{curr}$;
33:             end if
34:         end for
35:         Feed $\mathcal{F}(S_{curr}, \tau_{curr})$ back to the controller;
36:     end if
37: end while
```

3 | Methodology and Algorithms

This chapter will follow the procedures through which the solution is calculated, from defining the problem data to the calculation of the final FTIs. It focuses on how the algorithms were developed and how they work, presenting a pseudo-code for each algorithm. The pseudo-code provides meaningful insights about the implementation of the algorithms, highlighting the critical points that might prevent them from functioning properly.

3.1. Declaration of the Data

Recalling the definition of a Petri Net, two $p \times t$ matrices -referred to as Pre-matrix and Post-matrix- are required, where p is the number of places of the net and t its number of transitions. Both matrices are populated using the weights associated to the arcs connecting places and transitions. The initial marking M_0 is a $p \times 1$ vector indicating how many tokens are present in each place at the initial time instant.
The Q matrix provides the information about the time windows for each transition, so it is composed of t rows and two columns to express the lower and upper bounds of the transition. Next, a $t \times 1$ vector is needed to specify whether the transitions are controllable or non-controllable. This is accomplished using a 1 to indicate the former type and a 0 for the latter.
Lastly, information about the GTSS is required, including the initial markings set, the target markings set and the forbidden markings set, all composed of $p \times 1$ vectors. The ADT and AAT intervals constitute the time requirements and take the form of two 1×2 vectors.

3.2. Construction of the MSCT

It is not uncommon for TPNs to present cycles in their structure, i.e. an admissible TS whose starting and final markings are the same. The presence of cycles in TPNs has interesting consequences for the MSCGs constructed from them, which will be analysed below.

The presence of loops in MSCGs requires a function that recognises when classes are equivalent. Two classes are equivalent when they possess the same marking and equivalent domains -two domains are equivalent when they represent the same time intervals for the enabled transitions at that marking. Thus, the aforementioned function has to verify this pair of conditions, a simple task as far as the marking is concerned, but a complex one for what regards the equivalence of the domains. This complexity is the main reason why, as cited in the previous chapter, the modified algorithm developed for this thesis creates a tree, which does not have loops and so develops only in one direction. Algorithms that construct trees only analyse each node once greatly simplifying the procedure and lowering the computational cost.

Example

Let us consider the cyclic TPN:

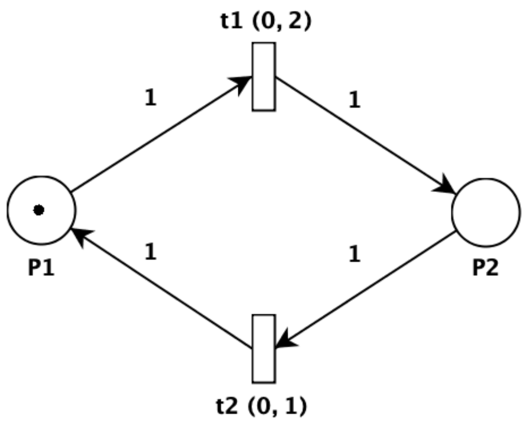


Figure 3.1: A simple cyclic TPN.

Its corresponding MSCG, according to algorithm 1.1, will be:

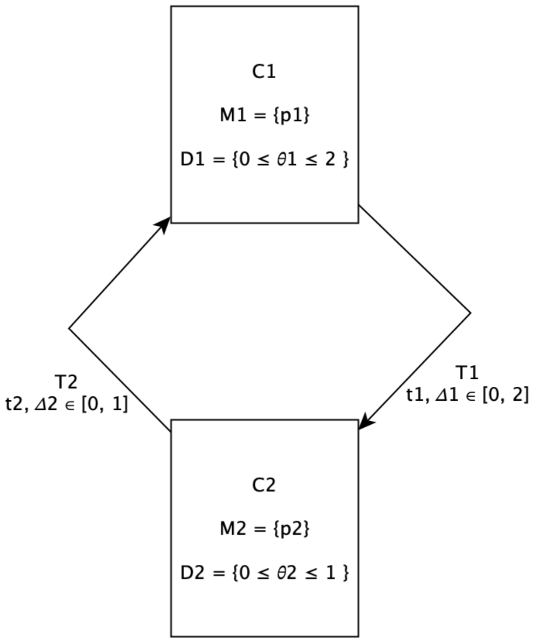


Figure 3.2: The corresponding MSCG.

where, for a given edge i of the MSCG, the label is composed of T_i which is the identifying number of the edge; the transition that fired as t_k; and $\Delta_i \in [lb_i, ub_i]$ which is the time window of the FTV.

The graph can be converted into a tree by avoiding the step which checks whether the class to be added is equivalent to one that is already part of the graph or not. The result is presented below.

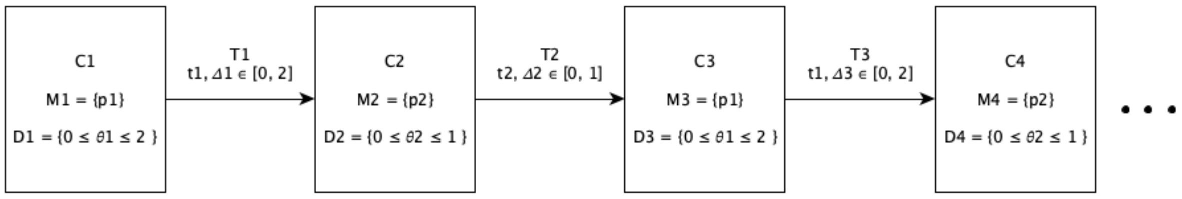


Figure 3.3: The corresponding MSCT.

3.2.1. Boundedness Assurance

Opting for a tree structure, while simplifying the construction of the graph, leads to a severe drawback referred to as *state explosion*. This happens because every new entry introduces a new class, thus causing an exponential growth of the number of nodes of the tree.

A solution is implemented to prevent the tree from becoming too large or even unbounded -a condition that would cause the program to never stop. By labeling the edges with the cumulative time that it takes to traverse them, the exploration of a path can be interrupted once a maximum allowed time of exploration is reached. This parameter is referred to as the *maximum time limit* (MTL).

In accordance with the policy of maximal permissiveness, all paths that might fulfill the specifications must be considered. For this reason, the cumulative time value that edges are labeled with is the sum of the lower bounds of the transitions that have fired in the path, adjusted for the amount of time transitions have been enabled for.

This operation is summarised by the following recursive algorithm, in which the edges are labeled as the tree is explored. The key aspect is the discounting of the lower bound of the time elapsed since the fired transition was first enabled (**line 6**).

Dot notation is used to identify the relationship that classes have in the tree: given a generic class C, $C.prev$ identifies the parent class of C, while $C.succ$ indicates its successive class. This does not lead to ambiguity thanks to the tree's inherent structure: every node of the tree has only one parent.

Algorithm 3.1 MTL edge labeling (MTL_el)

Input: The MSCT up to the newly added class C, its parent class $C_{parent} = C.prev$, the fired transition t, and the initially defined lower bound of t: lb_t.

Output: the time label for the considered edge t_min.

```
1: if $t \in \mathcal{A}(M_{parent})$ then
2:     let $C_{parent\_parent} = C_{parent}.prev$ be the parent node of $C_{parent}$;
3:     if $t \in \mathcal{A}(M_{parent\_parent})$ then
4:         let $t\_min = lb_t - MTL\_el(t, C_{parent\_parent}, lb_t)$;
5:     else
6:         let $t\_min = lb_t - lb^{parent}_{scatt}$;
7:     end if
8: else
9:     let $t\_min = lb_t$;
10: end if
```

As a last operation, the label is calculated as the sum of t_min and the label of the edge leading to the parent node C_{parent}, which represents the time elapsed from the initial time instant to reach C_{parent}. It must be noted how, if t_min is not positive, the new label will be equal to the label of the previous edge.

At **line 2**, C_{parent_parent} refers to the parent node of the considered parent node. This, in conjunction with the recursive nature of the algorithm, allows for the algorithm to explore the tree upwards, considering all the classes in which the fired transition has been enabled.

Line 4 is where the algorithm is invoked again with a different input: the use of the C_{parent_parent} node enforces that the exploration moves up the tree.

The lb^{parent}_{fired} at **line 6** symbolises the initially defined lower bound of the transition that fired in the parent class considered in the iteration of the algorithm.

The actual enforcement of the MTL criterion is performed as the tree is constructed: edges whose cumulative time is less than the value of the MTL are explored and added to the tree, otherwise they are discarded.

It can be deduced that a tree whose construction was stopped because of this criterion will have its leaf-nodes -the terminal nodes of the tree- reachable within MTL time instants from τ_0.

3.2.2. Structural Matrices of the MSCT

Considering how the value of every FTV of the edges of the graph must stay inside the limits defined by their firing window, effectively, these windows represent the conditions that the FTVs must satisfy at all times. Firing windows can be interpreted as two inequalities that define the range of values that each FTV can take: every FTV must be greater or equal to its lower bound and less or equal to its upper bound in order to be legal. Collecting the inequalities of all the FTVs of a net into a system, referred to as the *system of structural inequalities*, creates a linear problem that must be always verified in order to move in the space of the MSCT.
The system of structural inequalities can be represented using the standard formulation of a linear problem of inequalities $A \cdot \Delta \leq b$, where matrix A is the *structural matrix*, Δ is the vector containing the FTVs and b is the *structural vector of known terms*. It should be pointed out how, throughout the work of this thesis, the linear problem will be addressed using two structural matrices, one for the "greater or equal" inequalities and one for the "less or equal". This is solely for practical reasons, as in order to solve the system, the two matrices must be merged together, after changing the sign of the "greater or equal" structural matrix and vector of known terms.
It follows that the structural linear problem has a one-to-one correspondence to its graph, and mathematically defines the system being studied.
Below, two frameworks for building the system of inequalities are presented. One uses *symbolic variables* as an intermediate step to obtain the structural matrix, while the other builds it directly.

Structural Inequalities using Symbolic Variables

The basic reasoning behind this approach is to create a mathematical object for the FTVs of the system, adding the structural constraints to said objects as the tree is explored and enlarged.
Let us consider a generic edge of a MSCT T_i, class C_j connected to its tail and class C_k connected to its head. T_i is labeled with t_h, the transition that fired, and Δ_i, its FTV. D_j is the domain of C_j, consisting of n inequalities of the kind $lb_q \leq \phi_q \leq ub_q$ for $q = 1, 2, ..., n$ -one for each enabled transition in C_j-.

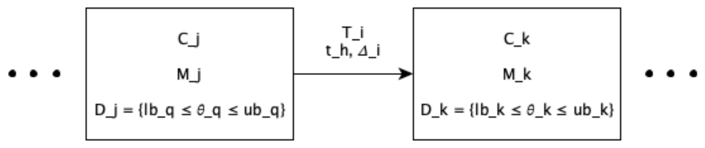


Figure 3.4: A generic section of a MSCT.

The time window of Δ_i must be:

$$\max\left\{0, lb_h - \delta_{previous\ i}\right\} \leq \Delta_i \leq \min_{q:t_q \in D_k}\left\{ub_q - \delta_{previous\ q}\right\}$$

where $\delta_{previous\ i}$ and $\delta_{previous\ q}$ represent the numerical components for the time elapsed since the considered transition first became enabled. Naturally, the transitions enabled in C_j will have $\delta_{previous} = 0$.
The left bound employs the "maximum" operator to ensure that Δ_i is greater or equal to 0. The right bound employs the "minimum" operator to obtain the minimum value of all the upper bounds of D_j, adjusted for the amount of time they have been enabled for. This is because, in order for a specific transition to fire, its upper bound must be less than or equal to the minimum upper bound of the domain it belongs to. Otherwise, recalling the basic assumptions of Petri Nets, the transition whose upper bound was reached will fire in its stead.
Using symbolic variables to model the FTVs of the system means that the program creates a mathematical object for each and every FTV of the system. Subsequently, the aforementioned constraints are enforced each time a new class is added to the MSCT. When the tree is complete, a function converts the relationships between the FTVs into the structural matrix of the corresponding linear problem and its structural vector of known terms.
This implementation is the most direct translation of **algorithm 1.1**, so the same pseudocode can be used as reference to understand its workings. This similarity makes this approach the more intuitive to be implemented of the two, but its benefits are limited -the underlying premise makes it severely inefficient and slow. As a matter of fact, the program is required to manage a mathematical object for every FTV of the MSCT. Moreover, multiple transitions might be enabled in the same class, therefore multiple FTVs are present in a single domain. This requires large memory usage and effectively prevents the practical use of this approach.

Direct Method to obtain the Structural Inequalities

The core idea behind this method is to update the structural matrices as the MSCT is constructed. This is possible thanks to the fact that, for each new edge being added, the resulting constraints never violate the ones already part of the system. Moreover, when transitions stay enabled over multiple classes, they depend on the constraints that are already part of the system.
Throughout this thesis the two structural matrices that are built while exploring the tree are referred to as VM -for the "greater or equal" inequalities- and Vm -for the "less or equal" inequalities. The two vectors of known terms are called m for the one corresponding to VM and n for the one corresponding to Vm. Consequently, recalling the standard structure for a system of inequalities, the system matrix takes the form of $A = [-VM; Vm]$, while the vector will be $b = [-m; n]$.

Let us consider the generic section of a MSCT represented by figure 3.4, recalling how the time window of the FTV of the edge must belong to the time window

$$\max\{0, lb_h - \delta_{previous\ i}\} \leq \Delta_i \leq \min_{q:t_q \in D_k} \{ub_q - \delta_{previous\ q}\}$$

In order to obtain the actual linear inequalities to be added to VM and Vm some manipulation of the bounds of the window is necessary, as both extremities depend on the result of non-linear operators.
Exploiting the properties of the "maximum" operator, the following expression is true:

$$x \geq max\{a_1, a_2, ...a_n\} \ \rightarrow \begin{cases} x \geq a_1 \\ x \geq a_2 \\ \vdots \\ x \geq a_n \end{cases}$$

and the same stands for the analogous case

$$x \leq min\{a_1, a_2, ..., a_n\} \ \rightarrow \begin{cases} x \leq a_1 \\ x \leq a_2 \\ \vdots \\ x \leq a_n \end{cases}$$

As far as the lower bound is concerned, there will always be two terms inside the "maximum" operator, therefore, for each FTV of the MSCT, two linear inequalities are added to VM. This allows for the prediction that VM will be a $2d \times d$ matrix, recalling how d is the number of FTVs of the system.
The minimum operator that provides the value of the upper bound has a non-predictable number of terms in it, so it is not possible to predict the number of rows Vm will have. Nevertheless, it is safe to assume that it will be greater than or equal to the number of FTVs of the system, as it will always contain at least one term as the MSCT is being constructed. The considerations regarding the number of rows of the two matrices apply to their respective vectors of known terms as well.

The second meaningful change performed on the expressions that define the time window is division of the triangular inequalities into the two basic inequalities they are made of.

$$lb_i - \delta^i_{previous} \leq \Delta_i \leq ub_j - \delta^j_{previous} \;\rightarrow\; \begin{cases} \Delta_i + \delta^i_{previous} \geq lb_i \\ \Delta_i + \delta^j_{previous} \leq ub_j \end{cases}$$

where, for the sake of brevity, only one "less or equal" inequality is taken into consideration.
VM and Vm will be populated by the coefficients of the FTVs -which can be either 0 or 1-, while m and n will be filled with the corresponding lower and upper bounds of the initially declared time windows. Two additional vectors, the id vectors $idVM$ and $idVm$ for VM and Vm respectively, are necessary. They link rows to their corresponding FTV, allowing for the searching of a row based on the FTV of interest.
As a consequence of this structure, the time dependent part of the inequalities will be completely contained within the structural matrices, so, when a new class is added to the tree, its domain will only require the initially defined firing windows contained in matrix Q. This new concept of domain is referred to as the *temporary domain* of a class.

Below, the algorithm that manages the update of the structural matrices is presented. It should be specified that e_i is a generic edge of the tree, C_{parent} is the class connected to its tail and $C_{sibling}$ is the one connected to its head.

Algorithm 3.2 Update of the Structural Matrices

Input: C_{parent}, $C_{sibling}$, e_i, VM, Vm, m, n, $idvM$, $idvm$.

Output: The updated structural matrices VM, Vm and vectors of known terms m, n.

```
1: for all $t_j \in \mathcal{A}(M_{sibling})$ do
2:    if $t_j \in \mathcal{A}(M_{parent})$ & $max\{0, lb_j^{sibling}\} \leq min_{k:\ t_k \in \mathcal{A}(M_{sibling})}\{ub_k^{sibling}\}$ then
3:       add a row to $VM$ and $idvM$ using policy 1;
4:       for all $t_k \in \mathcal{A}(M_{sibling})$ do
5:          if $t_k \in \mathcal{A}(M_{parent})$ then
6:             add a row to $Vm$ and $idvm$ using policy 1;
7:          else
8:             add a row to $Vm$ and $idvm$ using policy 2;
9:          end if
10:      end for
11:      add the arc corresponding to the new row;
12:      update the total minimum time;
13:   else
14:      if $max\{0, lb_j^{head}\} \leq min_{k:\ t_k \in \mathcal{A}(M_{head})}\{ub_k^{head}\}$ then
15:         add a row to $VM$ and $idvM$ using policy 2;
16:         for all $t_k \in \mathcal{A}(M_{sibling})$ do
17:            if $t_k \in \mathcal{A}(M_{parent})$ then
18:               add a row to $Vm$ and $idvm$ using policy 1;
19:            else
20:               add a row to $Vm$ and $idvm$ using policy 2;
21:            end if
22:         end for
23:         add the arc corresponding to the new row;
24:         update the total minimum time;
25:      end if
26:   end if
27: end for
```

Policy 1

```
1: new id vector row(1) = new FTV id;
2: new matrix row(1:end) = prev. row(2:end); {prev. row is the row corresponding to $\Delta_i$ of $e_i$}
3: new matrix row(end + 1) = 1;
4: new known term row(1) = corresponding temporary domain entry;
```

Policy 2

```
1: new id vector row(1) = new FTV id;
2: new matrix row(end + 1) = 1;
3: new known term row(1) = corresponding temporary domain entry;
```

The condition at **line 2** and **line 18** is analogous to the one inside the *if* clause at **line 5** of algorithm 1.1: it checks whether the transition being considered can actually fire, i.e. that its lower bound is less than or equal to the smallest upper bound of the considered temporary domain D_{head}.
The characterising difference between the two policies is a consequence of the considered transition having been previously enabled (*policy 1*) or not (*policy 2*). In the first case, an additional step is required to account for the contribution of the time elapsed since said transition was enabled. This information is contained in the row of the structural matrix corresponding to the FTV of the edge e_i: Δ_i. Therefore, to identify which row refers to said FTV, the id vector of the appropriate structural matrix is checked for the corresponding FTV id, consequently obtaining the number of the desired row.
These operations are performed following the instructions of **line 1-2** of *policy 1*. **Line 3** completes the row by setting the place value relative to the current FTV equal to one. Lastly, **Line 4** fills the place of the vector of known terms with the corresponding value of the temporary domain.
Policy 2 is equivalent to *policy 1*, the sole difference being the absence of the second step of *policy 1*, in view of the enabling of the transition that is being considered in C_{head}.

Example

It is useful to illustrate an example of the tree and structural matrices being updated. For practical reasons the example in **chapter 1** is used, specifically the addition of the edges stemming from $C3$ to the MSCT. The only change implemented here is a switch to uniquely defined FTVs, achieved by using their edge's number as subscript for the corresponding FTV.

In this case $C_{parent} = C1$, $C_{sibling} = C3$, with their markings and domains named analogously; $e_i = T3$ meaning that $\Delta_i = \Delta_3$. Following the algorithm, we start with the first transition of $D3$: t_2, which was already enabled in $C1$. In order for t_2 to fire, the corresponding FTV, Δ_5, must be greater or equal to the lower bound of t_2 and less than or equal to all of the upper bounds of the transitions in $D_{sibling}$. Hence, accounting for Δ_3, the inequalities at play are:

$$\begin{cases} \Delta_5 \geq 0 \\ \Delta_5 + \Delta_3 \geq 1 \\ \Delta_5 + \Delta_3 \leq 2 \\ \Delta_5 + \Delta_3 \leq 5 \end{cases}$$

where the first inequality is omitted in the final version of VM since the "greater than or equal to 0" condition will be enforced when calling the function that solves the linear problem (all FTVs are non-negative). It should be noted how the second inequality will be added to VM, and the remaining two will be added to Vm.
The fact that t_2 was enabled in $C1$ means that *policy 1* must be used to update VM and the first new row of Vm:

```
1: idvM(new) = [5];
2: VM(new) = [0 0 1];
3: VM(new) = [0 0 1 0 1];
4: m(new) = 1;
```

```
1: idvm(new) = [5];
2: Vm(new) = [0 0 1];
3: Vm(new) = [0 0 1 0 1];
4: n(new) = 2;
```

where *idvM* and *idvm* are the two vectors containing the id of the FTV of the corresponding structural matrix row.
For the second new row of Vm, *policy 2* is used:

```
1: idvm(new) = [5];
2: Vm(new) = [0 0 0 0 1];
3: n(new) = 5;
```

The second enabled transition in $D3$ is t_4, which was not enabled in $D1$. Therefore, Δ_6 being the FTV of the new edge, the system of structural inequalities to be added takes the form:

$$\begin{cases} \Delta_6 \geq 0 \\ \Delta_6 \geq 1 \\ \Delta_6 + \Delta_3 \leq 2 \\ \Delta_6 \leq 5 \end{cases}$$

Policy 2 must be chosen for the update of VM and of the second new row of Vm:

```
1: idvM(new) = [6];
2: VM(new) = [0 0 0 0 0 1];
3: m(new) = 1;

1: idvm(new) = [6];
2: Vm(new) = [0 0 0 0 0 1];
3: n(new) = 5;
```

while *Policy 1* must be used to update the first new row of Vm as seen below:

```
1: idvm(new) = [6];
2: Vm(new) = [0 0 1];
3: Vm(new) = [0 0 1 0 0 1];
4: n(new) = 2;
```

With both procedures being defined, it is possible to assess their performances. To do so, the example presented in **chapter 1** is considered once again. To quantitatively determine the effectiveness of the two approaches, the functions that implement them are timed. Considering the presence of a loop in the TPN, the MTL parameter is necessary to stop the exploration of the tree -the chosen value is 10 time instants.
For the first method, using symbolic variables, it takes 8.526973 seconds to complete the task. Using the second method, directly computing the elements of the structural problem, the time required is cut to 0.147066 seconds. The difference is staggering, and

so justifies the decision to only utilise the direct approach from here onward.

3.3. Path-finding Algorithm

The task performed by the path-finding algorithm is to find all the paths that lead from the starting node to the target one(s). This task must be performed as it is necessary to identify all the legal trajectories of the tree, both from a logical and from a temporal perspective. Hence, taking into consideration that the check on the temporal aspects of the specification is going to be performed at a later stage, the path-finding algorithm must provide all the logically compliant trajectories.
There is a substantial amount of research concerning path-finding algorithms, with optimal algorithms existing for a variety of tasks. As far as the problem at hand is concerned -finding a path between two nodes of a graph- there are two main options to choose from: the *breadth-first search* (BFS) and the *depth-first search* (DFS). Both options completely explore the graph, ensuring the exhaustiveness of the solution they provide. The differentiating characteristic is that the former analyses all the nodes at the present depth first, subsequently moving on to a lower level, while the latter, starting from a root node, explores as far as the graph reaches, backtracking once a leaf-node or an already explored one is found. Both approaches require extra memory to function, a *queue* (FIFO policy) in the case of the BFS and a *stack* for the DFS.
The two algorithms work in similar manners and neither presents any noticeable advantages when applied to the case at hand. This being said, BFS is preferred because DFS does not guarantee the completeness of the solution. In fact, DFS can enter a part of the graph that functions as a trap and thereby does not allow the algorithm to explore the remaining areas of it. BFS will eventually find a goal state if it exists in the graph.

3.3.1. Remarks about the Breadth-First Search Algorithm

Some of the most important aspects of BFS, already studied extensively in past literature, are summarised in the following section.
The complexity of the BFS algorithm can be expressed as $o(|V|+|E|)$, where $|V|$ represents the number of vertices and $|E|$ represents the number of edges in the graph. In the worst case scenario, every vertex and every edge will be explored. It is important to note that the complexity of $o(|E|)$ can vary between $o(1)$ and $o(|V|^2)$, depending on the sparsity of the input graph. When the number of vertices in the graph is known beforehand and additional data structures are used to track which vertices have been added to the queue, the space complexity can be expressed as $o(|V|)$. This is in addition to the space required

for storing the graph itself, which can vary depending on the chosen representation. [7, 13] In algorithm analysis, BFS is assumed to operate on finite graphs represented explicitly using adjacency lists, adjacency matrices, or similar representations, so there does not need to be an adaption of the data that constitutes the MSCT.

The pseudo-code of BFS as described in [7] is:

Algorithm 3.3 Breadth-First Search Algorithm

Input: A graph G, a starting node *root*, a target node *goal*.

Output: The goal state. The parent links trace the path back to the root node.

```
1: Let $Q$ be a queue;
2: Label $root$ as explored;
3: Add $root$ to $Q$;
4: while $Q$ is not empty do
5:    let $v$ be the first element of $Q$;
6:    remove $v$ from $Q$;
7:    if $v == goal$ then
8:       return $v$;
9:    end if
10:   let $w$ be the nodes reachable from v;
11:   for all $w$ do
12:      if $w$ is not labeled as explored then
13:         label $w$ as explored;
14:         let $v$ be the parent node of $w$;
15:         add $w$ to $Q$;
16:      end if
17:   end for
18: end while
```

A couple of remarks are of order. In **line 5**, the first element of the queue is the first that was added to it, since the policy of a queue is FIFO (First In First Out). The *if* clause at **line 12** is superfluous in the case of a tree structure, as loops are not present.
The way BFS is implemented for this work requires the net, the starting marking and the target marking as input. The output consists in all the paths that start in the starting marking and end in the target one. Each path is expressed as the sequence of classes it involves.

3.4. Pruning of the MSCT

Having established information about the net and the logically legal trajectories on it, its simplification can be performed. The tree will be reduced in size by pruning all of the branches that do not lead to the target marking(s). It is an exceptionally important step because it allows for a drastic decrease in the size of the graph, which in turn will make the subsequent procedures faster and more efficient.
The goal is to obtain the so called *partial modified state class tree* (PMSCT), which is the tree that only includes the legal trajectories of the MSCT. To be more specific, the pruning operation treated in this section returns the tree that contains only the logically legal trajectories. Further pruning will be performed once the temporal specifications are taken into consideration, resulting in the final PMSCT of the underlying system.
The pruning algorithm consists of a two-step procedure:

- The marking and deletion of the classes and edges that do not belong to the logically legal paths.
- The deletion of the rows and columns of the structural matrices that correspond to the deleted edges.

In this way, only the elements of the system that play a role in the behaviour described by the specifications are present.
The first step of the pruning process is the flagging of the nodes of the tree. Classes are marked with one of three flags: 0 for *safe classes*, 1 for *critical classes* and -1 for *forbidden classes*. A safe class is a class whose sibling classes are either a target class or all themselves safe classes. Forbidden classes are defined as those whose sibling classes are all forbidden. Lastly, critical classes have at least one safe sibling class, and at least one forbidden class. For this reason, they cannot have less than two classes stemming from them. Classes are initialised as forbidden when they are added to the graph.
For reasons that will be made clear in the following section, critical classes require all of their sibling classes to be present in the tree, therefore the PMSCT that the algorithm creates will contain forbidden classes.

Let us now consider the pseudo-code that describes the pruning algorithm, recalling how the dot notation is used to refer to the attributes of classes. Therefore, given a class C, $C.marking$ refers to its marking, $C.tempdomain$ refers to its temporary domain and $C.flag$ to its flag.

Algorithm 3.4 Pruning Algorithm

Input: The MSCT G, its structural matrices *VM, Vm, m, n, idvM, idvm*, the set of forbidden markings F and the logically legal paths P.

Output: The PMSCT and its structural matrices.

```
 1: for all paths $p \in P$ do
 2:     for all nodes $n \in p$ do
 3:         let $n_i$ be the considered node and $n_i.succ$ the set of nodes that have $n_i$ as their parent node;
 4:         if size($n_i.succ$) $> 1$ then
 5:             $n_i.flag =$ critical;
 6:         else
 7:             $n_i.flag =$ safe;
 8:         end if
 9:     end for
10: end for
11: for all nodes $n \in G$ do
12:     let $n_i$ be the considered node and $n_i.marking$ its marking;
13:     if $n_i.marking \in F$ then
14:         $n_i.flag =$ forbidden;
15:     end if
16: end for
17: for all nodes $n \in G$ do
18:     let $n_i$ be the considered node and $n_i.succ$ the set of nodes that have $n_i$ as their parent node;
19:     if all($n_i.succ.marking$) $=$ safe then
20:         $n_i.flag =$ safe;
21:     end if
22: end for
23: for all edges $e(n_i, n_k) \in E$ do
24:     if $n_i.flag = forbidden$ & $n_k.flag = forbidden$ then
25:         let $e.flag = forbidden$;
26:     end if
27: end for
28: for all edges $e \in E | e.flag = forbidden$ do
29:     delete the row and column of VM, Vm, m, n, idvM, idvm corresponding to the current edge's delta;
30:     delete the current edge;
31: end for
```

Where from **line 1** to **line 22** the flagging of the net is performed, and subsequently, from **line 23** to **line 27**, the deletion is performed.

3.5. Path Constraints

The PMSCT that was obtained in the previous section is the actual space of the control problem: it represents only the areas of interest of the MSCT. Recalling how it is made up of all the classes that lead to the target class -from a logical stand point- it is of primary importance to ensure that the evolution of the net is kept inside the limits of the PMSCT. This task is achieved by adding constraints that prevent the firing of transitions that lead to forbidden classes. These constraints will be added in correspondence to the critical classes, identified in the flagging step performed in algorithm 3.4.
The main idea behind the implementation of these constraints is to force the transitions that lead to safe classes to preempt the firing of those that lead to forbidden classes, in a way disabling them independently of their controllability.
To facilitate the understanding behind how the constraints work, it is useful to consider a generic example of a portion of PMSCT that requires constraints to be added:

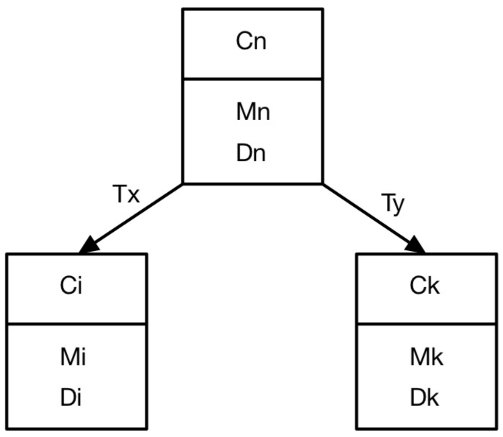


Figure 3.5: A generic critical node.

It is assumed that C_n is the critical class, C_i is the safe class and C_k is the prohibited one; with a little abuse of notation, transition T_x is referred to as safe and T_y as prohibited. The time windows are assumed to be $\Delta_x \in [lb_x, ub_x]$ and $\Delta_y \in [lb_y, ub_y]$ for T_x and T_y respectively.

Four options arise when considering the controllability of the transitions that fire in T_x and T_y, three of which are important as far as the introduction of the constraints is concerned:

1. The prohibited transition is controllable,
2. The prohibited transition is non-controllable while the other one is controllable,
3. Both transitions are non-controllable.

The last option is not of interest since both transitions are non-controllable and so no behaviour can be forced on the evolution of the system.
Considering the first option, the constraint that must be added is $\Delta_x < ub_y$. This is because the controllable prohibited transition can be prevented from firing until it reaches its upper bound, unless of course another transition fires before that time instant, therefore a "less than" operator is enough to ensure the condition. It should be noted that the only modification applied to the constraint on Δ_x, that is already be part of the structural system of inequalities, is that of a "less than" rather than a "less or equal" inequality. For this reason, the row of matrix Vm, which represents the "less or equal" constraint, corresponding to Δ_x will be used to build the constraint on the path. A last remark should be spent on the implementation of strictly "less than" inequalities, as the function to solve the linear problem only supports "less than or equal to" inequalities. The compliance of the function with strict inequalities operators will be ensured by adding to the inequality of interest a fictitious variable greater than zero, which is not subjected to any constraints, and is minimised.
In the second case the constraint that enforces the avoidance of the prohibited class is $\Delta_x < lb_y$. Let us analyse how to obtain it, starting from the information present in the structural matrices and vectors of known terms.
Recalling how, generally speaking and assuming that FTVs are positive, the lower bound of a FTV is $lb_{initial} - \delta_{previous}$, where $lb_{initial}$ refers to the initial lower bound, and $\delta_{previous}$ is the vector of FTVs that fired as the current transition has been enabled. Therefore the constraint can be expressed as:

$$\Delta_x < lb^y_{initial} - \delta^y_{previous}$$

So, rearranging it as to bring $\delta^y_{previous}$ on the left side, we obtain the row of VM and of m corresponding to Δ_y, with the difference that the place corresponding to Δ_p is set to 0 and the one of Δ_x is set to one.

To clarify, let us present the algorithm that manages this operation.

Algorithm 3.5 Paths constraints algorithm.

Input: A PMSCT G and its structural matrices VM, Vm, m, n.

Output: The set of constraints associated to the legal paths.

```
1:  for all nodes $\in G$ flagged as critical do
2:      let $es$ be the set of edges of the considered node that lead to safe classes;
3:      let $ef$ be the set of edges of the considered node that lead to forbidden classes;
4:      for all $ef$ do
5:          let $tf$ be the transition that fired in the considered forbidden edge;
6:          if $tf$ is controllable then
7:              for all $es$ do
8:                  let $\Delta_s$ be the FTV of the considered safe edge;
9:                  cond(new) $= Vm(\Delta_s)$;
10:                 known term $= n(\Delta_s)$;
11:             end for
12:         else
13:             if not all $es$ are non-controllable then
14:                 for all $es$ do
15:                     let $\Delta_f$ be the FTV of the considered safe edge;
16:                     cond(new) $= VM(\Delta_f)$;
17:                     exchange the values of the places of $\Delta_s$ and $\Delta_f$;
18:                     known term $= m(\Delta_f)$;
19:                 end for
20:             else
21:                 mark the current class as prohibited;
22:             end if
23:         end if
24:     end for
25: end for
```

Where $Vm(\Delta_s)$, $VM(\Delta_f)$, $n(\Delta_s)$ and $m(\Delta_f)$, at **line 9**, **line 10**, **line 15** and **line 18** respectively, refer to the row(s) of the structural elements whose id vector's row(s) has the number of the needed FTV.

It should also pointed out that, if the command at **line 21** is reached, the whole flagging-pruning-constraints addition operations have to be performed again, as the PMSCT will be different.

3.6. Calculation of the Solution

The last operation to be performed is the calculation of the FTIs for the enabled controllable transitions of the considered class of the PMSCT so as to guarantee the verification of the specifications.
This task is achieved by solving the appropriate linear systems of inequalities, built by merging the aforementioned sets of constraints with a new set that will be described in this section. Moreover, from the solutions of the resulting linear problems, a value for the FTIs of the enabled controllable transitions is calculated.
The solving of the linear system of inequalities will be performed by a pre-existing function. It should be pointed out that, according to the approach described in **chapter 2**, two linear problems will be associated with each path, one to calculate the minimum amount of time required to traverse the considered trajectory, and one to calculate the maximum. Therefore, two additional ILPs -one for each bound of the interval- need to be solved in order to calculate the FTIs of the enabled controllable transition(s) for each path of the tree. The final solution will be obtained by performing the intersection of all the obtained time windows.
Before tackling the calculation of the solution, it is good practice to test whether the system subjected to the constraints admits a solution. This step can be performed by solving the problem created by the structural system of inequalities and the constraints on the paths. If the resulting linear system allows for a solution to exist, then the actual calculation of the solution can be carried out.

Let us revise the nomenclature used from this point onward. Given a path Π, a controllable transition t_i, the state of the system S at the time instant τ, the FTI of the controllable transition t_i being considered is:

$$\mathcal{F}^{\pi}_{t_i}(S, \tau) = [L^i_\pi, U^i_\pi]$$

To ensure that the solution is robust the minimum and maximum time to traverse the path need to be considered: the former is the result from a minimisation problem, while the latter from a maximisation one. In this way, uncontrollable transitions are considered as if they fired at their earliest and latest possible moment respectively, thereby providing the worst-case scenario required to assess the robustness of the solution. It is important to consider the solutions returned by the function as the sum of the FTVs that are involved in the considered path, and not as the single values of the FTVs. An example is presented below.

Example

This example presents the importance of considering the time taken to traverse a path as a whole, and not as the single time windows returned by the function.
A path made up of only two edges is considered, whose FTVs are:

$$\begin{aligned}&\Delta_1 \in [1, 3]\\&\Delta_2 \in [\max\{0, 4 - \Delta_1\}, 5 - \Delta_1]\end{aligned}$$

Calculating the minimum total time amounts to adding together the lower bounds of the FTVs. Therefore:

$$lb_{tot} = \Delta_1^{min} + \Delta_2^{min} = lb_1 + lb_2 = lb_1 + 4 - lb_1 = 4$$

A similar reasoning can be applied to calculate the upper bound, obtaining $ub_{tot} = 5$.
It is possible to misinterpret the single time window of Δ_2 thereby obtaining incorrect values that make the interval invalid. The only result that is of interest is the total time window $\Delta_{tot} = [4, 5]$.

The two optimisation problems mentioned previously are typical: two linear systems of inequalities are built and solved, optimising with respect to their cost function. Their core difference concerns the matrices that define the systems and how they are obtained. As a matter of fact, the calculation of the minimum time it takes to traverse the paths must consider the lower limit of the uncontrollable transitions. Symmetrically, the maximum time requires to consider the upper bound of non-controllable transitions.
Let us now analyse how this behaviour is realised.

3.6.1. Minimisation Problem

At this point, the elements of the structural linear problem and the ones of constraints deriving from the logically legal paths have been calculated. In order to force the function that solves the linear system to consider the uncontrollable transitions as all firing at their earliest available moment, additional constraints are required.
For this task, only one new inequality is added for each non-controllable FTV, as can be seen in the following example in which the generic lower bound of a FTV is considered:

$$\begin{cases} \Delta_i + \Delta^i_{previous} \leq m_i \\ \Delta_i + \Delta^i_{previous} \geq m_i \end{cases}$$

where Δ_i is the considered non-controllable FTV, $\Delta^i_{previous}$ is a vector containing the FTVs which have fired as Δ_i was enabled, and m_i is the known term corresponding to Δ_i, present in the corresponding row of m. It should be noted that if Δ_i has been enabled in a single class, then $\Delta^i_{previous}$ will be equal to 0.
It is evident that Δ_i can only be equal to $m_i - \Delta^i_{previous}$, which is its lower bound. The structural matrix VM and its vector of known terms m are utilised to formulate the new constraint.
This new set of constraints takes the form of a matrix containing the rows of VM corresponding to the uncontrollable FTVs and their relative known terms. Both the matrix and the vector of known terms are multiplied by -1 so as to change the inequality sign to "less or equal".

3.6.2. Maximisation Problem

In order to solve the linear problem as if uncontrollable transitions all fired at their latest possible moment, analogously to the previous case, a new set of constraints is needed.
As previously described, there are multiple inequalities that define the upper bound for any given FTV, which then force the FTVs to be less than or equal to each upper bound of the transitions enabled in their parent class. The task this section aims to address is forcing the upper bound of the time window to be equal to the lesser of all the values that identify the upper bound corresponding to the considered uncontrollable FTV.
Mathematically, a generic FTV of the PMSCT Δ_n, is subjected to:

$$\begin{cases} \Delta_i \leq n_{i_1} \\ \Delta_i \leq n_{i_2} \\ \vdots \\ \Delta_i \leq n_{i_k} \end{cases}$$

where $n_{i_1}, n_{i_2}, ..., n_{i_k}$ are the entries of n corresponding to Δ_i, the considered non-controllable FTV.
The problem at hand is essentially developing a set of constraints that ensures that $\Delta_n = \min\{n_1, n_2, ..., n_k\}$ is verified. A mathematical workaround to linearise the previous expression exists, and it consists of the addition of the following constraints:

$$\begin{cases} \Delta_i \geq n_{i_1} - M \cdot y_1 \\ \Delta_i \geq n_{i_2} - M \cdot y_2 \\ \vdots \\ \Delta_i \geq n_{i_k} - M \cdot y_k \\ y_1 + y_2 + ... + y_k = k - 1 \end{cases}$$

Where M is an arbitrarily big parameter, and $y_1, y_2, ..., y_k$ are binary variables. The last constraint ensures that only one of those variables is equal to 0, as k is the number of binary variables to be used. This works because when the binary variables are equal to 1, the right side will always have a negative value, satisfying the structural constraints. A binary variable can be equal to 0 only in the case of the smallest known term, which is the only instance in which a binary variable set to 0 will not violate the structural constraints.

The constraints added for each non-controllable FTV are based on the corresponding rows of Vm, with the addition of the aforementioned components. This modification not only adds constraints to the linear system that defines the problem to be solved, but also variables; more specifically, k new columns are added to the matrix that represents the system.

After obtaining the elements of the two linear systems, the actual linear problem can be solved.

The cost function to be used in both instances is a $z \times 1$ vector, where z is the second dimension of the matrix being considered. In the case of the minimisation problem it will contain a 1 in correspondence to the FTVs involved in the path being considered and to the fictitious variables that enforce the strictly "less than" constraints. Symmetrically, recalling how the function utilised to solve the linear problems only supports "less or equal" inequalities, the cost function used for the maximisation problem will have -1 in the places of the appropriate FTVs. Technically, only the controllable FTVs are being optimised, considering that the non-controllable ones are fixed to take the value of their upper and lower bounds, but no significant drawbacks derive from the optimisation with respect to all the FTVs of the path.

3.6.3. Computation of the FTIs

It is now possible to obtain the values for the bounds of the FTI of the controllable transition under examination.
To do so, two additional optimisation problems must be solved for each path, one for the lower bound and one for the upper bound of the FTI that is being analysed. As far as the minimisation problem is concerned, recalling how the AAT vector defines the time window of arrival at the target marking, and so $AAT(1)$ refers to the lower bound of said window, it is structured as such:

$$\begin{gathered}\min\{\Delta_c\} \\ s.t.\ \Delta_c + \Delta_{nc} \geq AAT(1)\end{gathered}$$

Where Δ_c represents the FTV of the controllable transition whose FTI is being calculated, and Δ_{nc} indicates the sum of the FTVs of the uncontrollable transitions of the path i.e. the results of the optimisation problems treated in subsections 3.6.1 and 3.6.2. This separation is possible because the controllable transitions are treated as free parameters, while uncontrollable transitions are evaluated with a worst-case scenario policy so as to enforce the robustness of the result, and therefore represent a fixed amount of time.

Symmetrically, the optimisation problem for the upper bound of the FTI will be:

$$\begin{gathered}\max\{\Delta_c\} \\ s.t.\ \Delta_c + \Delta_{nc} \leq AAT(2)\end{gathered}$$

Where $AAT(2)$ is the upper bound of the time window of arrival.
The solution of these two optimisation problems defines the FTI of the controllable transition being examined, which in turn ensures that the specifications are met. In the event of the resulting FTI being invalid, i.e. its lower bound is greater than its upper bound, or either one of them cannot be calculated, the path must be discarded and the constraints preventing it from being taken are added.
Lastly, for the sections of the GTSS following the first one, the *absolute departure time* (ADT) is used to validate the FTI of the first transition exiting the initial class. If the FTI cannot guarantee the condition posed by the ADT, the path is prohibited and the tree is pruned.

Example

Let us consider a simple FTI calculation example. The simplified net being considered (considering all uncontrollable transitions as a whole) is the following:

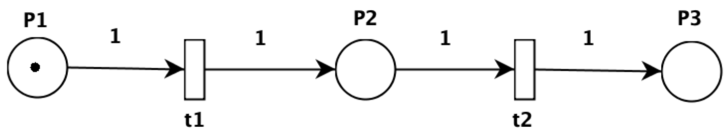


Figure 3.6: A simplified TPN.

Where t_1 is associated to a remote and controllable event, while t_2 is associated to a prospective and uncontrollable event with its time window being (2, 4). Considering an AAT of (3, 5.5), the two optimisation problems become:

$$\begin{array}{cc} \min\{\Delta_c\} & \max\{\Delta_c\} \\ s.t.\ \Delta_c + \Delta_{nc} \geq 3 & s.t.\ \Delta_c + \Delta_{nc} \leq 5.5 \end{array}$$

Which, evaluating for the extreme values that Δ_{nc} can take, become:

$$\min\{\Delta_c\} \qquad\qquad \max\{\Delta_c\}$$
$$s.t. \begin{cases} \Delta_c + 2 \geq 3 \\ \Delta_c + 4 \geq 3 \end{cases} \qquad s.t. \begin{cases} \Delta_c + 2 \leq 5.5 \\ \Delta_c + 4 \leq 5.5 \end{cases}$$

From the first problem we obtain $\Delta_c \geq 1$ and from the second one $\Delta_c \leq 1.5$, so the FTI of t_1 is [1, 1.5].

If, once the FTIs have been calculated for all the legal paths of the graph, all of them return a valid time window, the final result is obtained by performing the intersection of all FTIs. The intersection operator effectively enforces that the solution respects the specifications independently of the system evolution.

4 | Case Study

In order to verify the algorithms developed for this thesis a case study is considered. The example that was chosen is the one studied in [2], which provides a proven result which can be compared to the outcome obtained with the algorithms described in **Chapter 3**. The considered TPN is the following:

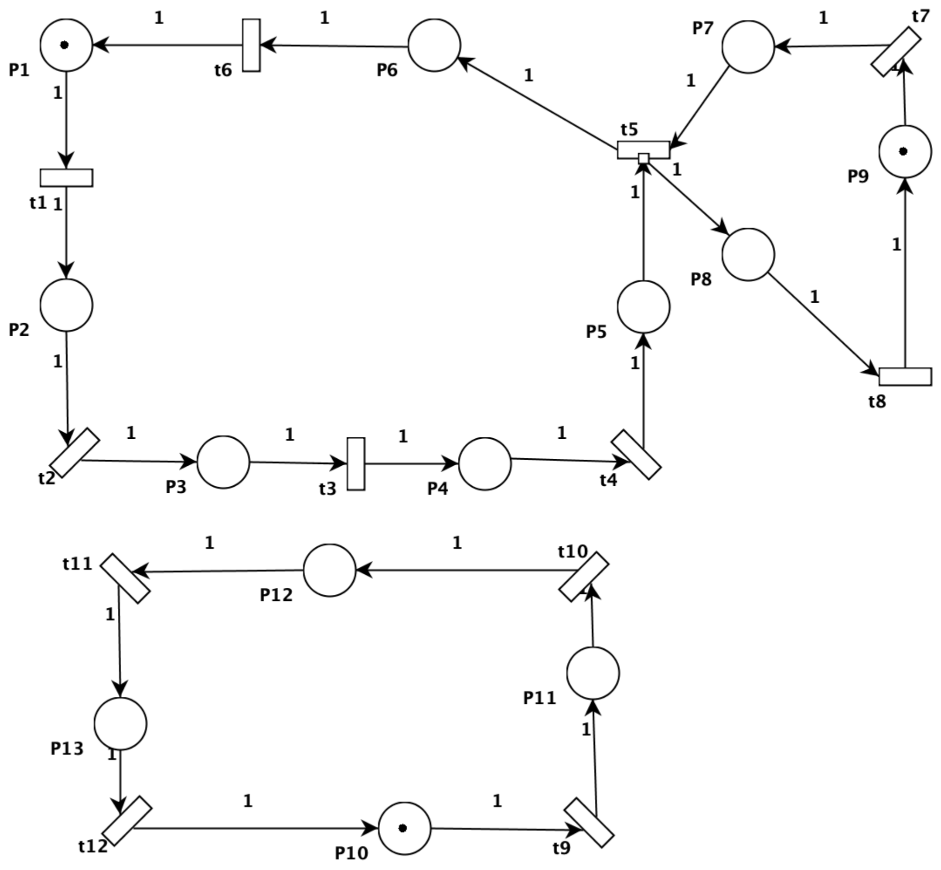


Figure 4.1: The TPN considered for the Case Study.

It schematises a simple material handling system made up of two automated guided vehicles (AGVs) -places 1 to 6 and 10 to 13- and a workstation -places 7 to 9.

Transitions t_1, t_7 and t_9 are associated to remote and controllable events. Therefore, their time windows are set from 0 to infinity. Transition t_6 is associated to a prospective and controllable event with a FTI from 3 to 4. The remaining transitions are associated to prospective and uncontrollable events, with their respective firing windows described in the following table:

Transition	Time Window
t_2	(1, 2)
t_3	(1, 2)
t_4	(1, 1)
t_5	(1, 1)
t_8	(7.5, 8.5)
t_{10}	(3, 4)
t_{11}	(3, 4)
t_{12}	(2, 2)

Table 4.1: Pure Time Windows of the Uncontrollable Transitions.

The specifications for the control problem are summarised by the GTSS:

$$g = (M_0, \emptyset, [0,0], []) \ (\mathcal{L}_1, \mathcal{F}_1, [3, 5.5], [3, 6.5]) \ (\mathcal{L}_2, \mathcal{F}_2, [9, 13.5], [])$$

where $\mathcal{L}_1 = \{M_1\}$ with $M_1 = p_5 + p_7 + p_{10}$, $\mathcal{F}_1 = \{M/M(p_3) + M(p_{12}) > 1\}$, $\mathcal{L}_2 = \{M_{21}, M_{22}\}$ with $M_{21} = p_1 + p_8 + p_{13}$, $M_{22} = p_6 + p_9 + p_{13}$ and $\mathcal{F}_2 = \{M/M(p_2) = 1 \vee M(p_9) = 1\}$.
In other words, the system has to move from its initial state, characterised by marking M_0 and the initial time instant τ_0, to marking M_1, which represents that an object is ready to be delivered by AGV1 to the input buffer of the workstation, while AGV2 is (or remains) idle. This change has to occur between 3 and 5.5 time instants. At the same time, the markings that have $M(p_3) + M(p_12) > 1$ must be avoided, as they represent a zone that is shared by the two AGVs and where only one can stay at a time. Subsequently, the system leaves M_1 after at least 3 and at most 6.5 time units to reach either of the two markings of $\mathcal{L}_2$, which represent that either AGV1 or the workstation are in their home positions, while the other two elements are one step from their respective home positions. During this process, any marking with $M(p_2) = 1$ or $M(p_9) = 1$ must be avoided, because only one agent between AGV1 and the workstation can reach its initial state. The set $\mathcal{L}_2$

must be reached between 9 and 13.5 time instants.
For the sake of brevity, the solution of this example only revolves around the first step of the GTSS.

4.1. Graphical Representation

Throughout this example, graphical representations will be used to provide a functional way to represent time PNs and MSCTs. The modelling software chosen to convert the higher level data -matrices and structures- to images is Graphviz; it uses a proprietary programming language which allows for the easy automation of the file writing process, with the sole trade-off consisting in the transition labels being less descriptive in order to maintain a higher level of readability. The representation of the TPN used for the example made using Graphviz is provided below.

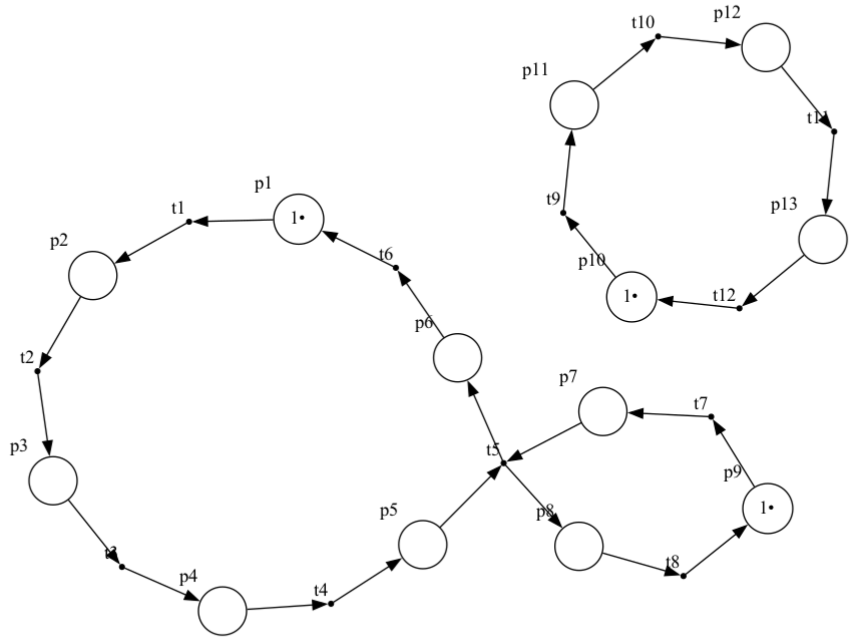


Figure 4.2: Graphical representation of the case study TPN using Graphviz.

Some graphical modifications with respect to the canonical way of representing TPNs have been made so as to fit with the Graphviz framework. Places are represented as circles labeled using a *p* and their identifying number; they can have tokens inside, which are represented as dots preceded by the number indicating how many tokens are present in the considered place. Transitions are drawn as dots and are labeled with a *t* followed by the number of the transition.

4.2. Results

The first step is to apply algorithm 1.1 with the variations introduced in algorithm 3.2. Considering how the underlying TPN allows for several loops to form, the state explosion phenomenon is inevitable. As a consequence, the exploration of the graph and construction of the structural matrices is stopped via the MTL parameter, in this case set to 5.5 time instants in view of the temporal specifications of the GTSS.
The resulting MSCT has 910 nodes, of which 377 are leaf-nodes, and 909 edges, and its structural matrices VM and Vm are 909×909 and 2534×909 respectively. The tree contains 377 paths, one for each leaf-node, which extend from a minimum of 6 classes to a maximum of 18.
First, the paths leading to the classes containing the target marking must be detected, using algorithm 3.3. Considering the first step of the GTSS, i.e. the paths that take the net from the initial marking $M_{initial} = p_1+p_9+p_{10}$ to the target one $M_{target} = p_5+p_7+p_{10}$, the sequences of classes that fulfill the aforementioned logical criterion are:

$$\begin{aligned}
\pi_1 &= \{C_1 \rightarrow C_2 \rightarrow C_5 \rightarrow C_{13} \rightarrow C_{31} \rightarrow C_{63}\}\\
\pi_2 &= \{C_1 \rightarrow C_3 \rightarrow C_8 \rightarrow C_{20} \rightarrow C_{44} \rightarrow C_{83}\}\\
\pi_3 &= \{C_1 \rightarrow C_2 \rightarrow C_6 \rightarrow C_{16} \rightarrow C_{38} \rightarrow C_{75}\}\\
\pi_4 &= \{C_1 \rightarrow C_2 \rightarrow C_5 \rightarrow C_{14} \rightarrow C_{34} \rightarrow C_{69}\}\\
\pi_5 &= \{C_1 \rightarrow C_2 \rightarrow C_5 \rightarrow C_{13} \rightarrow C_{32} \rightarrow C_{65}\}
\end{aligned}$$

where the trajectories can be grouped based on the controllable transition that fires in C_1:

$$\begin{aligned}
\Pi_1 &= \{\pi_1, \pi_3, \pi_4, \pi_5\}, && \text{if } t_1 \text{ fires}\\
\Pi_2 &= \{\pi_2\}, && \text{if } t_2 \text{ fires}
\end{aligned}$$

With the information of the logically legal trajectories being available, a first pruning of the MSCT can be performed using the procedure highlighted in algorithm 3.4. The pruning consists of discarding all the nodes that are not part of at least one of the five trajectories presented above. Below, a picture of the PMSCT is presented, where classes are coloured in yellow if they are critical, green if they are safe and red if they are forbidden.

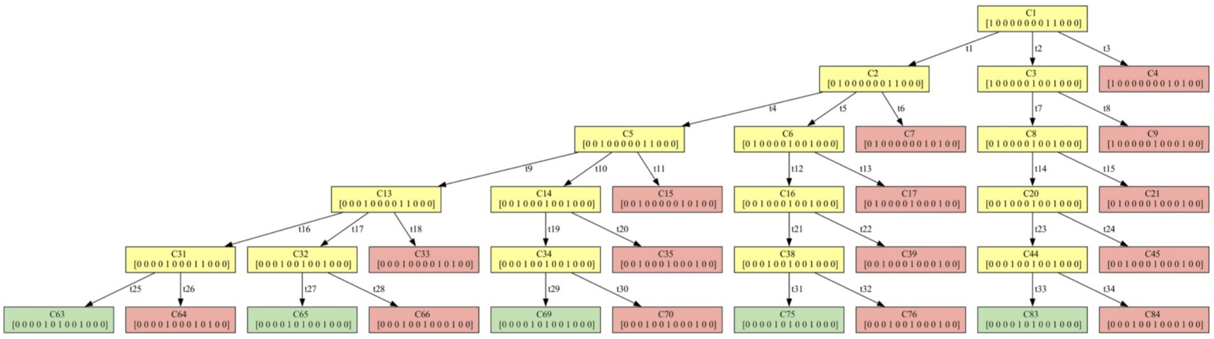
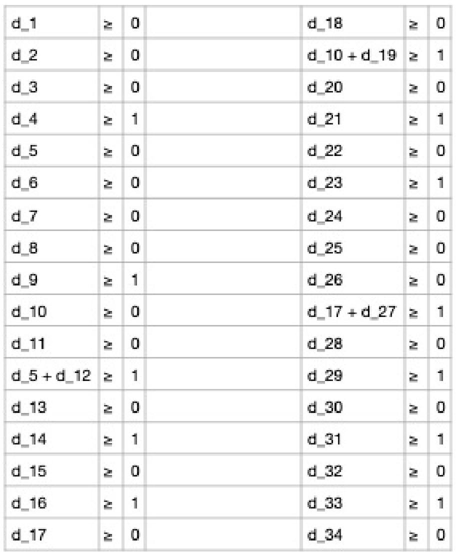


Figure 4.3: The first version of the PMSCT.

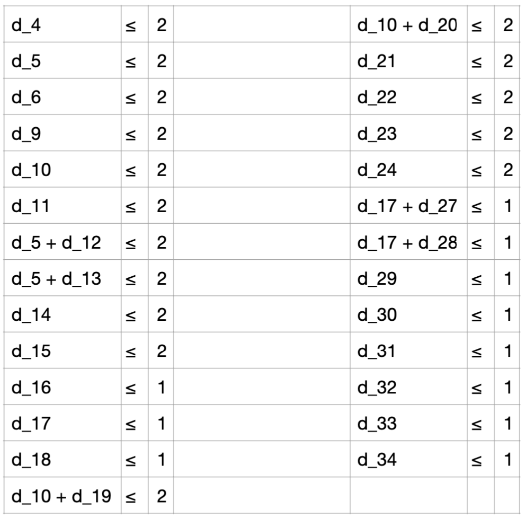
The corresponding tree has 35 nodes and 34 edges, a reduction of more than 96.1% for both elements compared to the full MSCT. Its structural matrices were also significantly reduced: VM is 34×34 while Vm is 80×34.

The inequalities that form the structural problem of the pruned tree are:



d_1	≥	0		d_18	≥	0
d_2	≥	0		d_10 + d_19	≥	1
d_3	≥	0		d_20	≥	0
d_4	≥	1		d_21	≥	1
d_5	≥	0		d_22	≥	0
d_6	≥	0		d_23	≥	1
d_7	≥	0		d_24	≥	0
d_8	≥	0		d_25	≥	0
d_9	≥	1		d_26	≥	0
d_10	≥	0		d_17 + d_27	≥	1
d_11	≥	0		d_28	≥	0
d_5 + d_12	≥	1		d_29	≥	1
d_13	≥	0		d_30	≥	0
d_14	≥	1		d_31	≥	1
d_15	≥	0		d_32	≥	0
d_16	≥	1		d_33	≥	1
d_17	≥	0		d_34	≥	0

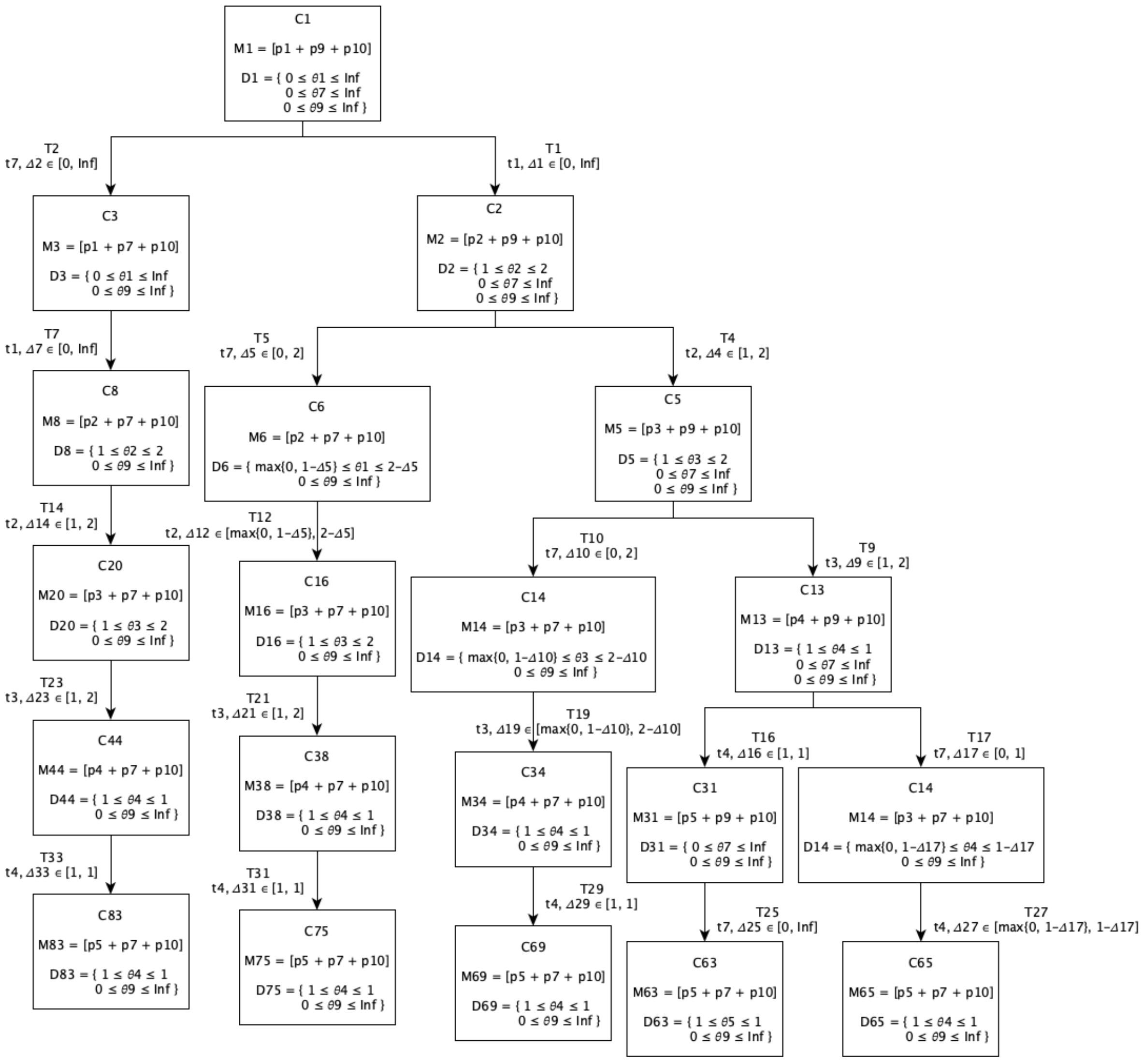
Figure 4.4: The inequalities of VM and m.



d_4	≤	2		d_10 + d_20	≤	2
d_5	≤	2		d_21	≤	2
d_6	≤	2		d_22	≤	2
d_9	≤	2		d_23	≤	2
d_10	≤	2		d_24	≤	2
d_11	≤	2		d_17 + d_27	≤	1
d_5 + d_12	≤	2		d_17 + d_28	≤	1
d_5 + d_13	≤	2		d_29	≤	1
d_14	≤	2		d_30	≤	1
d_15	≤	2		d_31	≤	1
d_16	≤	1		d_32	≤	1
d_17	≤	1		d_33	≤	1
d_18	≤	1		d_34	≤	1
d_10 + d_19	≤	2				

Figure 4.5: The inequalities of Vm and n.

A more descriptive version of the PMSCT is presented below, where forbidden classes have been omitted and the labels of the edges have been added.



The fully labeled PMSCT of the case study.

The structure of the labels of the edges is as follows:

$$Ti$$
$$tn,\ \Delta i \in [lb_i,\ ub_i]$$

Where Ti identifies the edge, tn the transition that fired and Δi indicates the FTV corresponding to its edge, accompanied by its firing interval.

At this point it is possible to construct the matrix of the constraints that keep the evolution

of the net inside the limits of the PMSCT, by applying algorithm 3.5. The matrix that is obtained is a 19×34 matrix even though there are only 15 critical classes. This is a consequence of the fact that more than one constraint can be added at a given class, depending on the number of its enabled transitions.
The resulting constraints end up not adding any actual limitation to the FTVs since they force them to be less than infinity.

It is now possible to compute the solutions of the two ILPs that provide the time window necessary to traverse each logically legal trajectory. Both are defined by the set of structural constraints and the set of constraints that keeps the evolution inside the PMSCT. To calculate the lower bound of the window, the set of constraints that considers the minimum fire time of uncontrollable transitions is added, while for the upper bound of the window, the set of constraints that considers uncontrollable transitions as though they fired at their latest is added. The results are:

$$\Pi_1 \rightarrow [3, 5] \ \forall \pi \in \Pi_1$$
$$\Pi_2 \rightarrow [3, 5] \ \forall \pi \in \Pi_2$$

where the lower and upper bounds are the values of the cost functions of the minimisation and maximisation problem, respectively. Considering the fact that both controllable transitions are remote, the two time windows coincide with the time it takes for all uncontrollable transitions to fire.

At this point, the optimisation problems described in section 3.6.3 are solved using $AAT = [3, 5.5]$. The final value for the FTIs of t_1 and t_7 are:

$$\mathcal{F}^{\pi}_{t_1}(S, \tau) = [0, 0.5], \text{ and } \mathcal{F}^{\pi}_{t_7}(S, \tau) = [0, \infty], \forall \pi \in \Pi_1$$
$$\mathcal{F}^{\pi}_{t_1}(S, \tau) = [0, \infty], \text{ and } \mathcal{F}^{\pi}_{t_7}(S, \tau) = [0, 0.5], \forall \pi \in \Pi_2$$

As it is possible to obtain a valid FTI for each of the paths, none need to be discarded, therefore no additional constraints are required.
The final solution, resulting from the intersection of the previous result, is:

$$\mathcal{F}^{\pi}_{t_1}(S, \tau) = [0, 0.5],$$
$$\mathcal{F}^{\pi}_{t_7}(S, \tau) = [0, 0.5]$$

It follows that the specifications of the first section of the GTSS are respected if t_1 and t_7 fire in the time window [0, 0.5].

4.3. Pruning of non-Compliant Paths

In order to highlight all the features of the set of algorithms that was developed in this thesis, we here slightly modify the temporal specifications in order to show what happens when non-compliant paths are obtained.
Suppose that the GTSS takes the following form:

$$g' = (M_0, \emptyset, [0,0], []) \; (\mathcal{L}_1, \mathcal{F}_1, [2, 4.5], [3, 6.5])$$

where the AAT has been decreased by 1 time instant, from [3, 5.5] to [2, 4.5], making the requirement stricter.

The first iteration of the solving process does not change: first, the MSCT is constructed -along with its structural matrices and vectors of known terms-, then its logically legal paths are found and the graph is pruned, obtaining the PMSCT. Subsequently, the constraints that keep the evolution of the system inside the PMSCT are formulated and added to the system of inequalities. Lastly, the time windows of the collected uncontrollable transitions are calculated, which, considering that the system was not modified, are equivalent to the previously analysed ones.
Despite there being a theoretically possible solution, as the time window of the specification overlaps with the one of the path, none can be guaranteed. As a matter of fact, uncontrollable transitions might require 5 time instants to fire, and the modified deadline is only 4.5 time instants. Consequently, all trajectories will be labeled as non-compliant. By removing the trajectories that breach the time specifications from the set of logically legal paths, the set of logically compliant trajectories for the second iteration is obtained; in this case the said set is empty. The flagging process is repeated, which, as there are no legal trajectories, will mark every class as forbidden, therefore no nodes belong to any path. As a consequence, the PMSCT will decay into a tree without nodes or edges. Clearly, a system of this kind does not admit any solution.

5 | Absence of Guaranteed Solutions

It is not unusual for systems to have trajectories that are not compliant with respect to the desired timed behaviour. It is still of great relevance to be able to obtain information about the system and instructions that specify which course of action provides the best outcome.
This is, to date, an uncharted segment of the supervisory control of TPNs; this thesis aims to outline the problem, defining some preliminary strategies to tackle it.

A path is non-compliant if either its logical or its temporal characteristics breach the specifications. Considering the former case, it means that the set of target markings does not belong to any class of the MSCT, hence there is no action that can help modify this situation. The specifications are simply incompatible with the considered TPN, as the marking refers to a state that cannot be reached by the system.
Of far more interest is the case of trajectories that do not satisfy the temporal boundaries of the GTSS. A TTS is non-admissible if its time window needed to reach/leave the objective does not overlap with the time windows specified by the GTSS. This being the case, it is useful to understand how much the time specifications must be relaxed in order for a solution to be present. This is yet another reason why the check on the timed aspects of the desired behaviour is done as a last step of calculating the solution to the control problem.

5.1. Relaxation of the Temporal Specifications

Whenever a trajectory breaches the conditions imposed by the specifications, useful information can still be obtained. As mentioned previously, the order in which the checks on the specifications are performed is key to retrieving information even when the considered TTS is not compliant. Once a logically legal trajectory is found, solving its linear system of inequalities provides the time interval within which said trajectory will reach the target marking. This interval, besides checking the compliance of the path, can be

used to address the relaxation of the temporal specifications.
This kind of analysis is useful in a broader context than just non-compliant controlled systems: it is possible to go beyond the scope of supervisory control and address the quality of the solution that is obtained.
For instance, let us assume that a compliant trajectory, both logically and temporally, has a small FTI; this will affect the final solution that is the result of the intersection between all the FTIs of all the trajectories. By excluding one legal, but temporally restrictive trajectory, the overall behaviour might be improved, granting a wider time window for the transitions to fire in. The trade off of such an operation is the addition of new constraints that prevent the discarded path from being taken, so they might restrict the behaviour of the system in an undesirable way.

In order to implement operations of the sort described in the previous section, a new parameter must be considered. Its task is to describe how much the time window of the considered trajectory deviates from that of the specifications. Given two generic time windows $W_{target} = [lb_t, ub_t]$ and $W_{path} = [lb_p, ub_p]$, where the former refers to the specifications -therefore it can be assumed to be either the AAT or the ADT- and the latter represents the minimum and maximum time it takes for a trajectory to reach the target. The *time performance parameters* (TPPs) of a trajectory are defined as:

$$\begin{cases} \Gamma^i_{lb} = ub_p - lb_t \\ \Gamma^i_{ub} = ub_t - lb_p \end{cases}$$

where Γ^i_{lb} indicates the lower bound TPP of the generic path i and Γ^i_{ub} the corresponding upper bound TPP. Their formulation stems from the requirement that the two time windows overlap for at least one time instant in order for the solution to be met.
It should be noted that the above definition ensures that the TPPs assume negative values when the specifications are not met. The motivation for this modification arises from the need to have a consistent and easily detectable characteristic that conveys whether the path is compliant or not. It is important to note that TPPs cannot both be negative at the same time, as that would imply that either of the time windows have no meaning.
Furthermore, the magnitude of the TPP expresses the extent of the deviation, enabling the application of quantitative analysis to the control problem.

A direct use for TPPs is the relaxation of the temporal specifications. It should be recalled that, in the framework explained in the previous chapters, if a path is non-compliant, it is discarded and the tree is pruned to reflect that. This completely excludes the path from the solution space. Clearly, if there is not a single compliant path the procedure will not

return any solution because there is no FTI that guarantees that the specifications are met.
The introduction of TPPs requires an additional step to be implemented: non-compliant trajectories are evaluated based on their TPPs, and, in accordance to the behaviour that is of interest for the user, the AAT and/or ADT time windows can be widened. More specifically, if any negative TPPs are present after the time properties of the paths are evaluated, they can be used as reference for how much the time specifications need to be relaxed to achieve a solution.

Example

Let us consider the case presented in **chapter 4** where the specifications are not met, recalling how $W_{path} = [3, 5]$ and $W_{target} = [2, 4.5]$. It is possible to evaluate the TPPs of the paths by applying the aforementioned definition to these time windows:

$$\Gamma^1_{lb} = 5 - 2 = 3 \ , \ \Gamma^1_{ub} = 4.5 - 3 = 1.5$$

Both TPPs are greater than 0, so the paths admit a solution. To know whether said solution can be guaranteed, additional calculations are required, namely solving the ILPs that return the FTIs of the controllable transitions, if they exist.

As previously mentioned, TPPs can also be used to address the quality of the solution when a solution exists. To be more specific, let us consider the case of a set of compliant trajectories, i.e. such that their TPPs are all greater than or equal to zero. The worst trajectory, i.e. the one with the smallest FTI, can be discarded, improving the behaviour of the overall system. To do so, a statistical analysis must be performed and a threshold value on which to base the discarding must be provided.

Example

To better understand the concept introduced above, let us consider the following TPN.

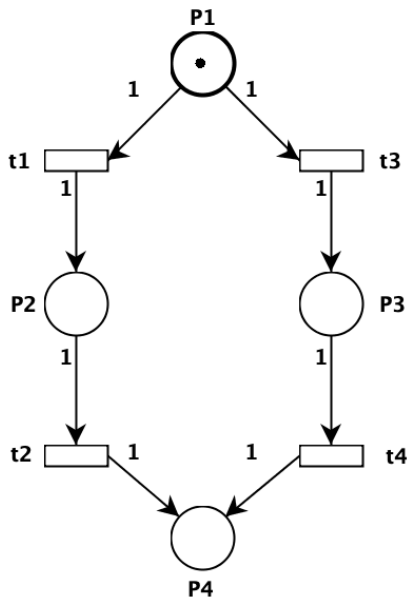


Figure 5.1: Quality of the solution example.

where transitions $t1$, $t2$ and $t3$ are associated to remote and controllable events (their firing window is $(0, \infty)$), while transition $t4$ is associated to an event that is prospective and non-controllable. The static firing interval of $t4$ is $(2, 3)$ and the AAT is $(2, 3)$.
It is evident how both trajectories -$\pi_1 = \{t1, t2\}$ and $\pi_2 = \{t3, t4\}$- are solutions to the problem resulting in the FTIs:

$$\begin{cases} FTI_{t_1}^{\pi_1} = (2,3) \\ FTI_{t_3}^{\pi_1} = (0,\infty) \\ FTI_{t_1}^{\pi_2} = (0,\infty) \\ FTI_{t_3}^{\pi_2} = (0,0) \end{cases} \rightarrow \quad \begin{matrix} FTI_{t_1} = (2,3) \\ FTI_{t_3} = (0,0) \end{matrix}$$

The FTI of $t3$ is very stringent, forcing $t3$ to fire at the moment it is enabled if π_2 is taken. Preventing $t3$ from firing eliminates the stringent FTI at no additional cost, since no constraints were required to be added. Consequently, π_1 becomes the only available choice, but with a much wider FTI.
While being a toy example, it serves as a proof of concept for the kind of analysis that

can be performed on TPNs, based on the quality of the solution.

It should be pointed out that the time specifications relaxation framework helps mitigate the effects that the unpredictability of uncontrollable transitions has, with regard to the overall time it takes to traverse a path. As a matter of fact, returning a solution that guarantees that the relaxed time boundaries are respected might result in the evolution of the net being compliant to the non-relaxed time specifications.

6 | Conclusion and Future Work

6.1. Summary of the Thesis

This thesis set out to codify the algorithms developed in previous scientific literature, notably [2, 3, 8]. It developed a set of functions that, with a TPN system and a GTSS as input, operates on them with the task of returning the FTIs that ensure that the specifications are fulfilled, if they exist.

Firstly, the procedure to construct the MSCT of the net was developed, expressing the MSCT both as a data structure and as a system of inequalities i.e. the structural system of inequalities.

The next step was to create the algorithms which would return the partial MSCT, a task that required the finding of all of the paths that logically lead to the target marking(s), flagging them accordingly, and removing them from the MSCT. This was also reflected in the modification applied to the structural matrices, so as to keep their one-to-one correspondence to the graph they describe.

Subsequently, the problem of keeping the evolution of the system within the boundaries defined by the PMSCT was tackled, resulting in the addition of constraints to the ILP that defines the system at hand. At this point it was possible to solve the aforementioned optimization problem, obtaining the span of time it takes to traverse the paths on the tree, which was subsequently used to calculate the FTI of the transition(s) to be controlled. In the event that a trajectory could not fulfill the time specifications, it would be flagged, removed, and additional constraints would be added as to prohibit the evolution of the system to take said trajectory.

Lastly, the final result was calculated by performing the intersection of all the FTIs of the controllable transitions, guaranteeing that the solution respects the specifications independently of the evolution of the system.

6.2. Limitations and Future Research Direction

A significant limiting factor has been the depth with which the physical systems have been modeled. It has been assumed that they provided no information other than those necessary to build a TPN. This lack of assumptions ensures that the modelling of any DES can be performed. The trade off for such an approach is the absence of specificity, which can drastically decrease the possibilities of control, as will be discussed below.

Having a more detailed description of the physical system being analysed means that it is possible to use statistical tools to model uncontrollable transitions, enabling the manipulation of their firing windows. More specifically, a probability density function could be assigned to every non-controllable transition of the net, enhancing the descriptiveness of the events which the transitions represent. With the additional information that this framework provides, further analyses are possible.
The time window can be reduced based on how conservative the solution needs to be, information which would need to be provided by the user.

Example

To better illustrate the concepts outlined in the previous paragraph, let us consider the example of an uncontrollable transition which models the arrival of some goods to a facility.
The delivery can take from a minimum of 20 minutes -when there is no traffic on the road- to a maximum of one hour -if the truck has an accident along the way-. It is clear that the upper bound is an extreme value that does not happen often, so it would be reasonable to consider a smaller time window for the uncontrollable transition.

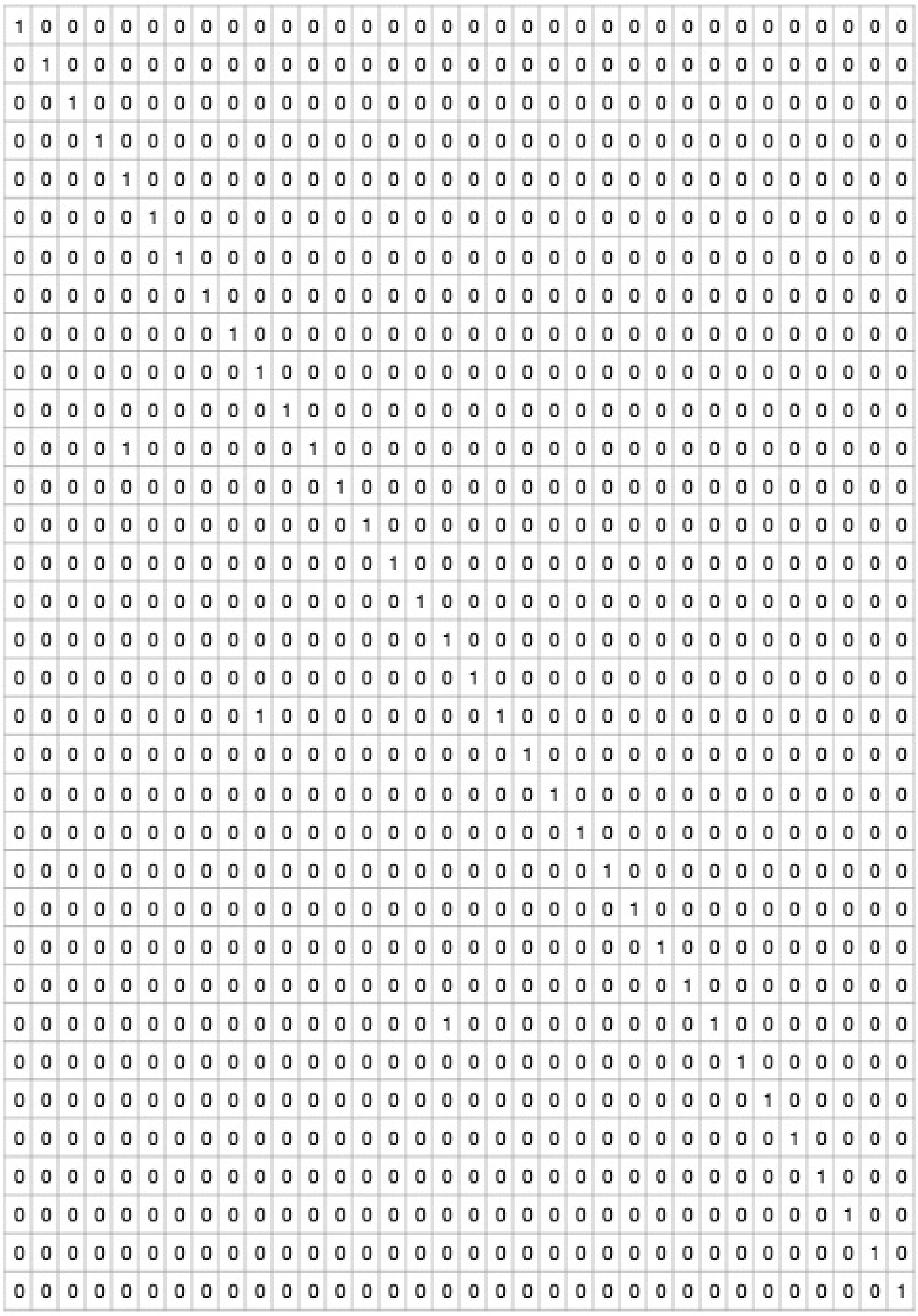
Broadening the scope of this method by considering multiple non-controllable transitions results in a higher degree of complexity that is difficult to address. This is because uncontrollable transitions are likely to be correlated, and the optimisation problems that return the solution is going to have an increased amount of degrees of freedom to manage. Every uncontrollable transition must have its firing window analysed and adjusted, while still being compliant to the parameter dictating how conservative the adjusting needs to be.
It should also be pointed out that the addition of probabilistic features to non-controllable transitions does not in any way affect the approach of the control. On the contrary, it

has the potential to produce positive consequences, which better the precision and the quality of the results of the control action.

Refining how models are built, employing statistical and probabilistic tools to conjunctively enhance the precision with which systems are studied, is one of the main areas that future work can focus on. Considering the areas where this theoretical knowledge can have a more natural and direct application -for instance traffic control or the control of industrial processes-, the increase in the precision of the solutions will have a substantial impact on how effective the application of these control strategies will be. The subsequent increase in efficiency and safety crystallises the cardinal ideals of engineering as a profession, and as a moral compass, staying true to the never ending search for improvement.

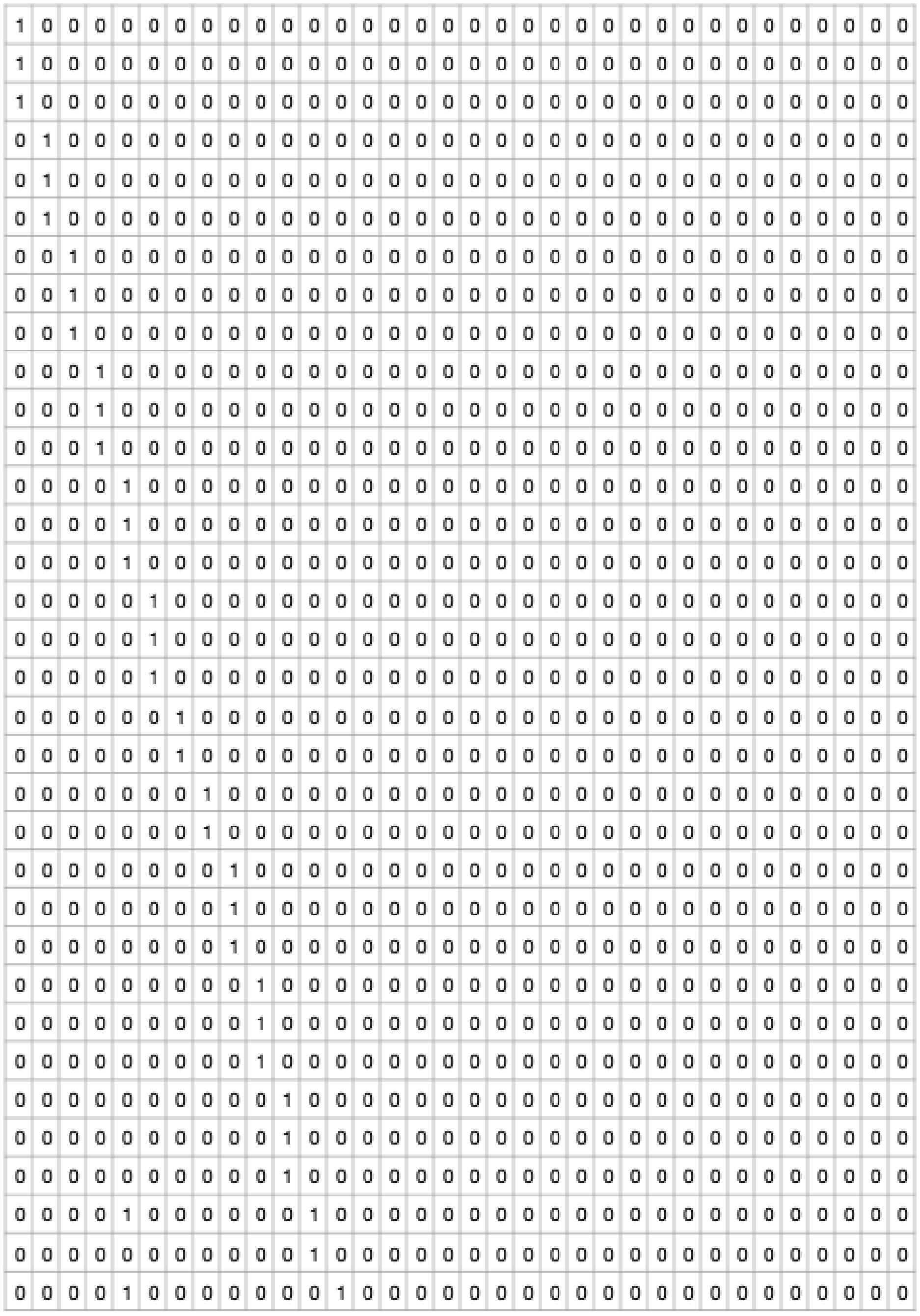
Appendix

Here are collected the figures of chapter 4 that might be of interest for the reader, but that are too cumbersome to present in the main part of the thesis. The structural matrices and vectors of known terms of the complete graph are not reported since their large dimensions make them not readable.

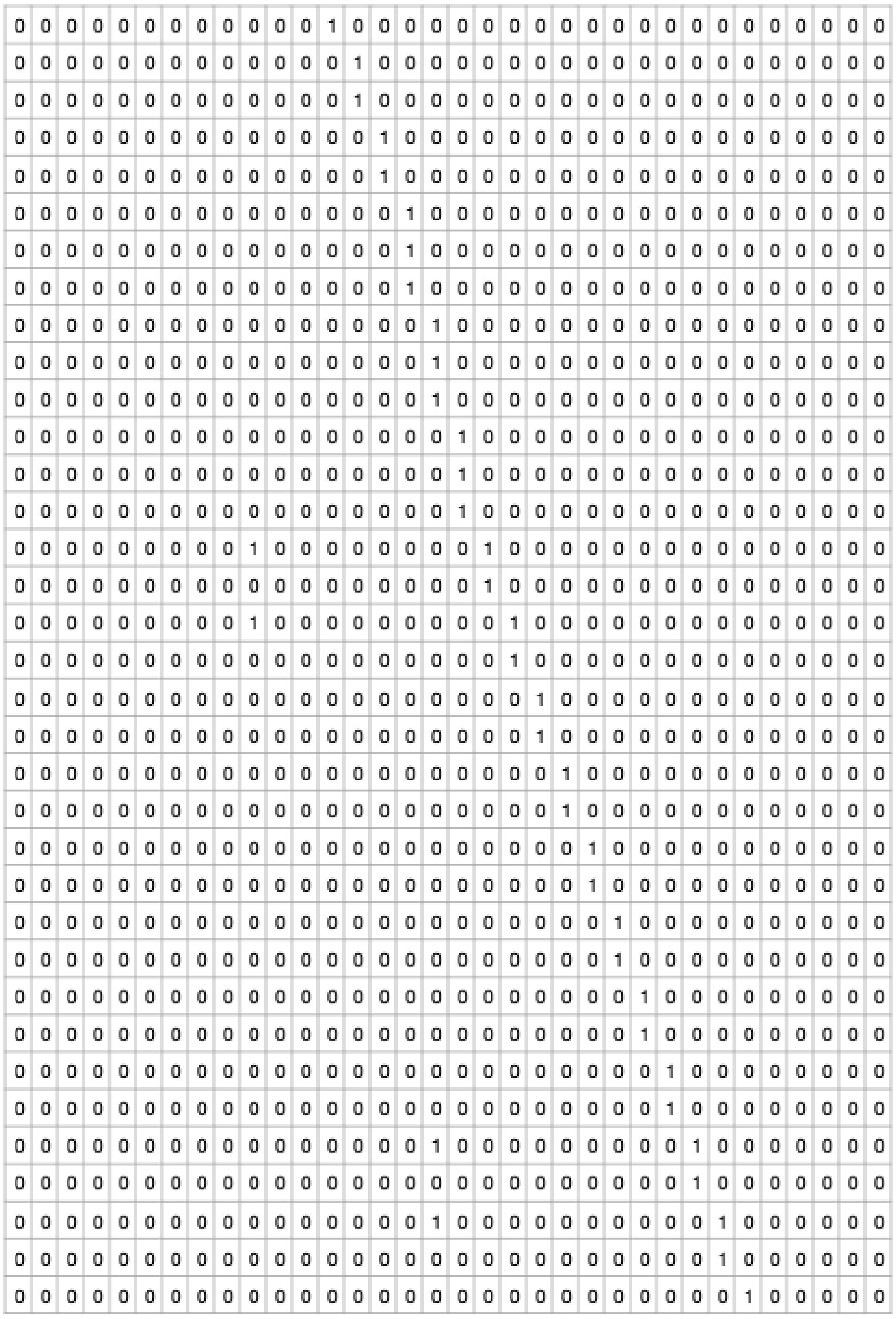


1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

The pruned *VM* matrix of the case study.



1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

The pruned Vm matrix of the case study.

0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0

The pruned transposed vector of known terms m of the case study.

∞	∞	∞	∞	∞	∞	∞	∞	∞	2	∞	∞	2	∞	∞	2	∞	∞	∞	∞	∞	∞	2	∞	∞	2	∞	∞	2	∞	∞	2	∞	2
∞	2	∞	2	∞	1	∞	∞	1	∞	∞	1	∞	∞	2	∞	2	∞	2	∞	2	∞	2	∞	2	∞	∞	∞	∞	∞	1	∞	1	∞
1	∞	1	∞	1	∞	1	∞	1	∞	1	∞																						

The transposed pruned vector of known terms n of the case study.

Bibliography

[1] F. Basile, M. P. Cabasino, and C. Seatzu. Marking estimation of time petri nets with unobservable transitions. pages 1–7, 2013.

[2] F. Basile, R. Cordone, and L. Piroddi. Supervisory control of timed discrete-event systems with logical and temporal specifications. *IEEE Transactions on Automatic Control*, 67(6):2800–2815, 2021.

[3] S. C. Basile F., Cabasino M. P. State estimation and fault diagnosis of labeled time petri net systems with unobservable transitions. *IEEE Transactions on Automatic Control*, 60(4):997–1009, 2015.

[4] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. Comparison of different semantics for time petri nets. In *Automated Technology for Verification and Analysis: Third International Symposium, ATVA 2005, Taipei, Taiwan, October 4-7, 2005. Proceedings 3*, pages 293–307. Springer, 2005.

[5] D. M. Berthomieu B. Modeling and verification of time dependent systems using time petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.

[6] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic control*, 39(2):329–342, 1994.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.

[8] Z. He, Z. Li, A. Giua, F. Basile, and C. Seatzu. Some remarks on “state estimation and fault diagnosis of labeled time petri net systems with unobservable transitions”. *IEEE Transactions on Automatic Control*, 64(12):5253–5259, 2019.

[9] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and system Sciences*, 3(2):147–195, 1969.

[10] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and system Sciences*, 3(2):147–195, 1969.

[11] P. Merlin and D. Farber. Recoverability of communication protocols-implications of a theoretical study. *IEEE transactions on Communications*, 24(9):1036–1043, 1976.

[12] P. M. Merlin. *A study of the recoverability of computing systems.* University of California, Irvine, 1974.

[13] E. F. Moore. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, pages 285–292, 1959.

[14] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230, 1987.

[15] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[16] C. Ramchandani. Analysis of asynchronous concurrent systems by petri nets. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC, 1974.

[17] C. Seatzu, M. Silva, and J. H. Van Schuppen. *Control of discrete-event systems*, volume 433. Springer, 2013.

[18] W. Wonham. A control theory for discrete-event systems. In *Advanced Computing Concepts and Techniques in Control Engineering*, pages 129–169. Springer, 1988.