POLITECNICO DI MILANO

School of Industrial and Information Engineering

Master of Science in Automation and Control Engineering

On hierarchical routing control in discrete manufacturing plants

Supervisor: Prof. Lorenzo Fagiano

Co-supervisor: Dr. Andrea Cataldo

Master Thesis dissertation of:

Roberto Boffadossi Matr. 905811

Accademic year 2019-2020

Abstract

The interest in automating demanufacturing processes has constantly grown in the recent years, especially for electronic products. The main feature required by this application is *high flexibility* due to the uncertain configuration of the products and the variability of the operations. Specifically, the problem of routing pallets across a plant is considered here. The purpose of this thesis is to improve and implement a new approach to address this problem: the *Hierarchical Routing Control* (HRC). It is based on *Model Predictive Control* strategy acting at supervisory level, combined with a low-level path following approach. A new solution for the path generation problem and path assignment redundancy problem is proposed and additional constraints are introduced in the plant model via new specific sub-routines. The conditions for lockout detection are identified and their implementation allows an *Intelligent Move Blocking.* Then the optimization problem is reformulated and sub-optimal solution methods are defined to reduce the computational time. A data exchange protocol is added to control algorithm in order to communicate with the integrated low-level logic that directly controls the pallet handling system. Finally the developed model is verified to be consistent with a real plant and the HRC strategy reaches high performances with very low computational cost.

Keywords: MPC, discrete manufacturing plant, Hierarchical Routing Control, Move Blocking, path following, lockout detection, handling system, search tree.

Sommario

L'interesse nell'automatizzare i processi di *demanufacturing* è cresciuto costantemente negli ultimi anni, specialmente per i prodotti elettronici. La caratteristica principale richiesta in questo tipo di applicazioni è un elevata flessibilità a causa delle condizioni incerte dei prodotti e della variabilità delle operazioni. Nello specifico è stato considerato il problema di instradamento dei pallet attraverso l'impianto. Lo scopo della tesi è di migliorare e di implementare un nuovo approccio di *Hierarchical Routing Control* (HRC) per affrontare questo problema. Esso si basa sulla strategia di *Model Predictive Control*, che agisce come supervisore, combinata con una logica di basso livello per l'inseguimento dei percorsi. Viene proposta una nuova soluzione per il problema della generazione dei percorsi e della ridondanza nel loro assegnamento; inoltre sono stati aggiunti ulteriori vincoli al modello dell'impianto per mezzo di due nuove sub-routine. Avendo individuato le condizioni necessarie per identificare la formazione di un *lockout*, la loro implementazione permette di impiegare la strategia di *Intelligent Move Blocking*. Successivamente viene riformulato il problema di ottimizzazione e utilizzando metodi sub-ottimi vengono ridotti i tempi di calcolo. L'aggiunta di un protocollo di scambio dati consente all'algoritmo di controllo di comunicare con la logica integrata di basso livello che controlla il sistema di movimentazione dei pallet. Infine, avendo verificato la completa coerenza tra modello e sistema reale, si dimostra che la strategia HRC raggiunge ottime prestazioni con tempi di calcolo molto bassi.

Contents

List of Figures

List of Tables

Acronyms

HRC Hierarchical Routing Control

MPC Model Predictive Control

FHOCFP Finite Horizon Optimal Control Problem

GPFS Greedy Path Following Strategy

MPPA Model Predictive Path Allocation

BZ Buffer Zone

CN Constrained Nodes

FTFS Forbidden Transition Filtering Strategy

AD Approaching Direction

BF Brute Force

IMB Intelligent Move Blocking

IMBSH Intelligent Move Blocking with Sub-Horizon

CP Control Platform

PLLC Plant Low Level Control

CNR Consiglio Nazionale delle Ricerche

Chapter 1

Introduction

In recent years the attention to material recovery and product restoration has increased constantly together with the importance of efficiency and sustainability of manufacturing processes. For this reason de-remanufacturing represents an important field of research. The ability of recycling materials from industrial waste and repairing discharged products represents new business opportunities, increasing energy saving and reducing the environmental footprint.

Quoting from [2]: *De- and remanufacturing includes the set of technologies and systems, tools and knowledge-based methods to systematically recover, reuse, and upgrade functions and materials from industrial waste and post-consumer products, to support a sustainable implementation of manufacturer–centric Circular Economy businesses.*

Specifically **demanufacturing** deals with recovering raw materials and components from discarded products, while **remanufacturing** restores or updates their functions along their life-cycle. In this way, to produce the same goods, a large part of the energy and materials are saved and the product "end-of-life" concept is replaced with "restoration".

Demanufacturing tasks are usually performed manually and so there is a great interest in automating these types of processes, especially for electronics waste.

In the past decades many demanufacturing studies have been developed on electronics products such as television, monitor, personal computer, printed circuit, electronic home appliances, in order to improve materials and reusable components recycling [3].

In order to explore the potentials of this area, in the laboratory of the Institute of Industrial Technologies and Automation (STIIMA), National Research Council (CNR), a pilot plant for automated de-remanufacturing has been set up, in figure 1.1. It has been designed to process End-of-Life electronic boards [4] and consists in:

- Flexible pallet handling system, composed of fifteen transport modules, that allows to move the boards from one machine to another.
- Load/Unload Robot Cell, that positions the electronic board on a pallet and loads it in into the plant. Furthermore the Cell unloads processed pallets from the line and removes the finished boards from them.
- Testing machine, which task is to analyze the state of the electronic board.
- Reworking Machine, used for repairing a board that has been detected to not work properly.
- Discharge machine, that dismantles the non reparable boards.

Firstly a functioning test of the electronic board is performed and it is sent to repair, to discharge or to the robot cell, depending on the result of the test. The complexity of this application is determined by the fact that each processed part has a personalized operations scheduling and the target machines are assigned online.

The main feature required by an automated demanufacturing plant is *high flexibility*. In fact there is a great variability in the condition of products belonging to the same family and for this reason the operations needed by a specific part

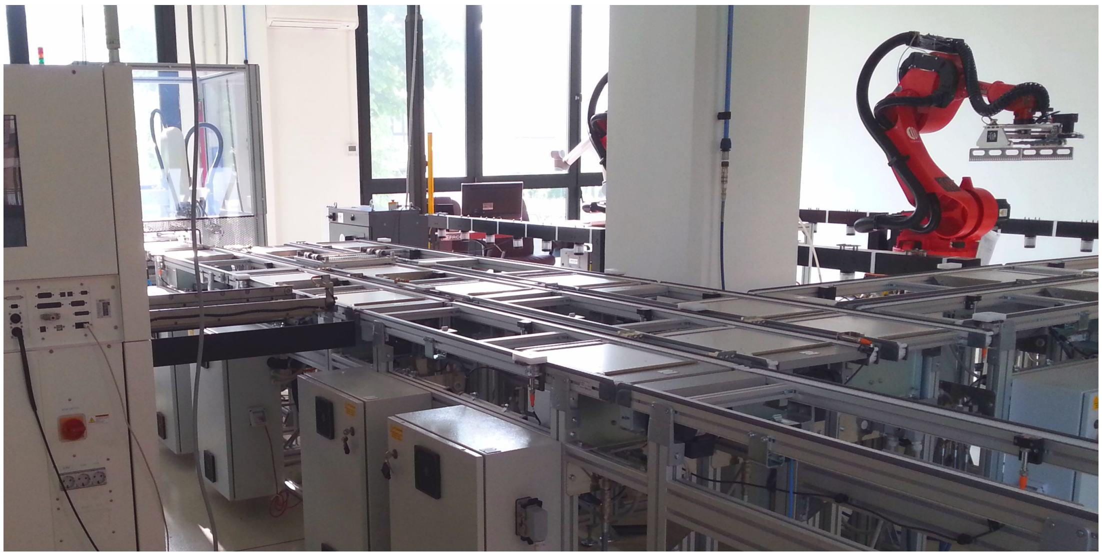


Figure 1.1: Demanufacturing pilot plant in the Institute of Industrial Technologies and Automation (STIIMA), National Research Council (CNR), laboratory, Milano

could be different. The product identification is essential in this application, due to the unknown configuration of the processed product, that can depend on consumer alterations, missing, damaged components or manufacturing errors [5]. This application requires a flexible layout of manipulators, sensors and handling devices, to detect the current state of the product and to address the variability of the operations needed to disassembly or repair it [6].

The purpose of this thesis is to improve, implement and verify a new approach for controlling the pallet routing in a discrete manufacturing plant: the *Hierarchical Routing Control* via *Model Predictive Path Allocation* and *Greedy Path Following*, proposed in [7]. The control algorithm will be set up for the demanufacturing pilot plant in the STIIMA-CNR laboratory previously described. This approach is based on the *Model Predictive Control* (MPC) strategy, which has the advantage of reformulating the control problem into an optimization ones.

1.1 Model Predictive Control

Nowadays MPC is one of the most widespread advanced control strategy in the process industry, because it can be used for a large number of applications and is capable of dealing with complex systems. The MPC control architecture is presented in the following block diagram in figure 1.2.

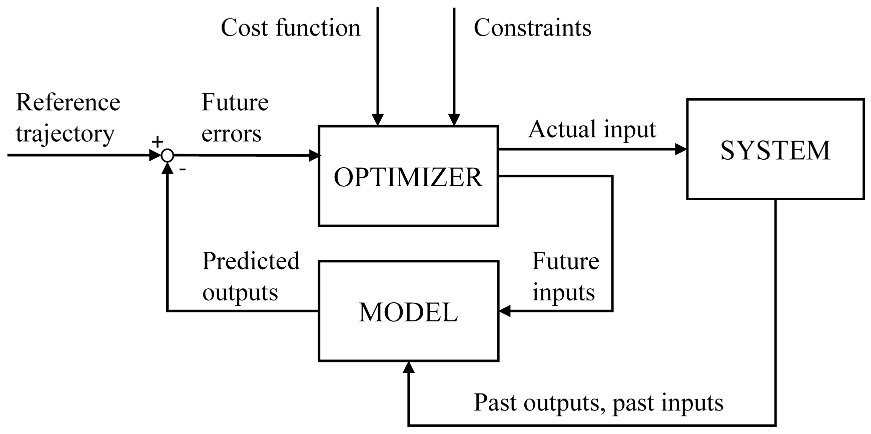


Figure 1.2: MPC scheme

The main features of this method are the following:

- The control problem is formulated as an optimization one, thus the control action is computed minimizing a specific objective function. The advantages of this characteristic are:
  - Designing a multi-objective cost function allows to take into account many different goals, even if conflicting with each other. For example it is possible to find a compromise between minimizing the energy consumption and maximising the plant throughput.
  - In the formulation of the control problem it is possible to explicitly include the state and input constraints.
  - It is suitable for a large range of processes and is an open metodology, improvable with future extensions.

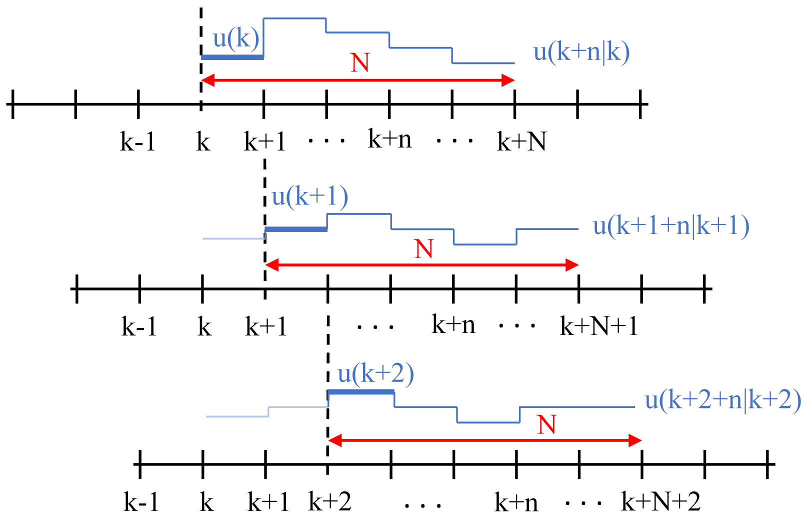

u(k)
u(k+n|k)
N
k-1 k k+1 · · · k+n · · · k+N
u(k+1)
u(k+1+n|k+1)
N
k-1 k k+1 · · · k+n · · · k+N+1
u(k+2)
u(k+2+n|k+2)
N
k-1 k k+1 k+2 · · · k+n · · · k+N+2

Figure 1.3: Receding Horizon principle

- A matematical model, usually in descrete-time, is used to predict the output of the system along a *prediction horizon* $[k, k+N]$, where k is the actual discrete time step and N is the length of the horizon.

- It adopts the so-called *Receding Horizon* (RH) principle: *at each time step k the future control sequences $[u(k), \ldots, u(k+N-1)]$ is computed solving the corresponding Finite Horizon Optimal Control Problem (FHOCP), based on the available process information at time k. Then only the first control action is applied and at the new time instant $k+1$ a new FHOCP is solved along the prediction horizon $[k+1, k+N]$, based on the available process information at time $k+1$* [8]. In this way the control operates in "close loop" and the provided control action is guaranteed to always respect all the system constraints (*recursive feasibility*). The RH principle has been represented in figure 1.3.

The high versatility of the objective function and the ability to deal with complex systems make this control strategy appropriate for a flexible (de-) manufacturing plant. But unfortunately the control problem that has to be solved at each step can be very complex. That depends on the large number of integer or Boolean control inputs and on the presence of many constraints, especially for the temporal-logic constraints that characterize a manufacturing application. Thus the controller has to solve a large combinatorial optimization problem in few seconds to provide the control action within the time step.

The MPC strategy has been frequently adopted in manufacturing plants, but with a different role. Usually it is implemented to compute the optimal online scheduling of the manufacturing system, see [9] [10], to support the supply chain management [11] or to improve the energy efficiency of the plant [12]. In these cases the MPC provides the long-term references to the lower control layer, i.e. the local controllers that establish the detailed operation of the discrete event manufacturing line. Considering the application developed in this thesis, the purpose of the controller is to compute directly the inputs of a pallet handling system that moves the electronic boards to their target machine.

The scheduling of the operations needed by a specific board depends on the state of that board and can not be decided by the controller. Furthermore the target of a board is defined online, because product identification is performed inside the plant at the same time as other operations. In fact the boards are tested by a machine in order to establish if they must be repaired, discharged or unloaded from the line. In the plant there is a single machine for each of these tasks, thus the next target is related only to the product state. The MPC computes the optimal pallet paths combination in order to send each board to the corresponding machine target, maximising the plant performances and avoiding to incur a lockout; moreover it provides the related Boolean control signals that trigger the transitions along the transport modules connecting the machines.

For the considered application, the previous contributions [13] [1] have described the plant by means of a directed graph and the system behaviour has been modeled from an Eulerian point of view, that represents the part movement inside the plant referring to the nodes as control volumes. In this way it has been possible to obtain a Mixed Logical Dynamical (MLD) formulation of the control problem, manipulating the plant linear model and the temporal-logic constraints that limits the part movements. This model has been implemented in a MPC strategy to compute the complex optimal control inputs and high performances has been reached, but the computational cost increases exponentially as function of the number of parts in the plant and the length of the prediction horizon.

The HRC approach [7] has the purpose to reduce the computational cost reformulating the MPC architecture. In fact the point of view is shifted to a Lagrangian one, describing the overall state as the collection of the states of the parts that are being processed by the plant. A sequences is assigned to each part state in order to indicate its path and the MPC controller is designed following a hierarchical structure. Two control layers operates to route the parts: an high-level logic allocates a path to each part solving the optimal control problem choosing from a precomputed set of sequences, a low-level logic moves the parts forward along the assigned path and compute the control inputs.

The validity of HRC approach has not yet been tested on a real case. The main goal of this thesis is to implement this control method in a real plant, including new specific constraints, addressing the problem of sequence generation and providing a new formulation of the optimization problem.

1.2 Thesis contributions

The main contributions developed in this thesis are:

- A new model of the demanufacturing plant in STIIMA-CNR laboratory has been developed using a Lagrangian approach.

- Two new sub-routines have been developed and included in the path following algorithm in order to fully represent the handling system constraints.

- The Lagrangian state has been reformulated to remove redundancy in the path assignment.
- New guidelines have been developed regarding the problem of sequence generation.
- A lockout detection function has been implemented.
- The *Move Blocking* approach has been improved adopting the lockout avoidance backtracking and the *Approaching Direction* (AD) concept. The new strategy has been named *Intelligent Move Blocking* (IMB).
- Several tree search sub-optimal methods have been implemented and a Move Blocking strategy with emptying condition has been developed.
- The IMB has been proved to reach excellent throughput performances with very low computational cost. Moreover, if an aggressive AD is adopted good performances are obtained even without optimization.
- A data exchange protocol has been introduced in the control algorithm to communicate with the real plant and the controller has been validated on a simulator of the STIIMA-CNR pilot plant.

1.3 Thesis structure

The thesis is organized as follows.
Chapter 2 explains the principles of the the Hierarchical Routing Control, introducing the Eulerian and Lagrangian formulations. The low-level logic and the high-level logic are presented and some critical issues are highlighted.
Chapter 3 describes the characteristics and the functioning of the STIIMA-CNR plant and the implementation of the HRC approach. New additional constraints are introduced and two new subroutines are included in the low-level logic in

order to model these constraints.
In Chapter 4 are solved two critical issues: the sequences redundancy in the path allocation algorithm and the sequence generation problem.
Chapter 5 proposes different solution methods for the optimal control problem that characterize the path allocation logic. Moreover two types of lockout are formulated and the related detection conditions are presented.
In Chapter 6 the results of the test are discussed and the different solution methods are compared considering the throughput value and the computational performances.
Chapter 7 describes how the control algorithm is set up for communicating with the plant and a simulator is used to validate the functioning of the controller.
Finally Chapter 8 analyzes the objectives achieved and offers hints for future works.

Chapter 2

Hierarchical Routing Control

The purpose of this thesis is to deepen a new approach developed to solve a real-time optimal routing problem in a discrete manufacturing plant and on adapting it to a real case. The considered problem refers to a flexible demanufacturing plant, where the controller has the task of optimizing the pallet routing through the handing system, because distinct items must be transported to different targets to complete a sequence of jobs. This approach has recently been formalized in the paper [7], leaving some issues unsolved. The main sources of complexity in this kind of problem are represented by the presence of temporal-logic constraints and by the large amount of discrete control inputs.

The previous contributions, see [13] [1], have developed a solution from an Eulerian point of view. In this way the state of the whole plant is considered as the set of the node states of the graph defining the handling system. Therefore the previous works have mathematically represented the system relying on a mixed logical dynamical model. According to a predictive receding horizon strategy, the control algorithm solves recursively the mixed-integer linear programming problem, deriving by the previous assumptions. As results the controller is able to provide the optimal solution at each time steps and the performances are better than the strategy based on a set of heuristic rules. Unfortunately this approach has shown an exponentially increasing computational cost, that depends both on the length of the prediction horizon and on the number of items to be moved.

In the new approach a different perspective of the system has been adopted, the point of view has passed from the Eulerian model to a Lagrangian model, that does not consider the nodes state but the state of each part processed by the plant. Moreover a hierarchical MPC structure has been adopted:

- At a higher level, the control algorithm solves recursively a Finite Horizon Optimal Control Problem (FHOCP) using a receding horizon strategy and assigns a path to each part in the plant, minimizing a defined cost criterion in order to reach the desired performances.
- At a lower level, a path following logic computes the control inputs needed by the parts inside the plant to follow the paths assigned by the high-level controller. At the same time this logic ensures that the control inputs are obtained satisfying all the constraints.

The Hierarchical Routing Control has been implemented and tested on the demanufacturing pilot plant located in the laboratory of Institute of Industrial Technology and Automation, National Research Council, Italy. The plant consists of 4 machines connected by a modular flexible transport line and its purpose is to test, repair or discharge electronics boards. In order to implement the approach to this particular case new constraints have been introduced, modifying the low-level logic. Some improvements have been performed to deal with the computational cost problem and with other several issues, such as defining the creation process of the paths to be assigned, that has not already been automatized.

All the changes and innovations will be discussed in the next chapters, while in this one the approach will be presented in the generalized form, referring to paper [7].

2.1 Eulerian System Model

The first formulation proposed is the Eulerian system model. Even if the control algorithm will operate from the Lagrangian model point of view, it is helpful to recall the Eulerian formulation for the sake of completeness and to introduce some features of the system. The plant is represented as a directed graph composed of a finite number $N_n \in \mathbb{N}$ of nodes and transitions. A Boolean control signal $u_{h,j}(k) \in \{0, 1\}$ models each transition, indicating if a part in the node h at the time k will move to the node j at time $k+1$, where $h, j = 1 \ldots, N_n$. For example, considering the plant represented in figure 2.1, $u_{4,8}(34) = 1$ means that at the time instant 34 a part is leaving the node 4 and will occupy the node 8 at the time instant 35. Then to each node is associated a Boolean variable $z_h(k) \in \{0, 1\}$, that represents the state of the node h at the time instant k: $z_h(k) = 1$ a part is present in the node h at time instant k, $z_h(k) = 0$ the node is empty at that time.

Referring to the whole set of nodes, we consider that N_t out of N_n nodes consist of *transportation nodes*, while the remaining $N_m = N_n - N_t$ denotes the *machines*. The set of machine node is indicated as:

$$\mathcal{M} = \{h : \text{node } h \text{ is a machine}\}.$$

Furthermore two special nodes must be taken into account, the *loading* node and the *unloading* node, denoted respectively with h_l and h_u. These could be associated with a single node, depending on the plant configuration, and are connected to a particular node that represents the outside of the plant. It acts as interface with an input and output buffer for new and finished parts and is generically indicated by the index 0. The control action u_{0,h_l} refers to the command that allows a new part to enter into the loading node from outside the plant, while $u_{h_u,0}$ refers to the command that brings out a finished part from the unloading node. The number of finished parts at time k is denoted with $N_f(k)$

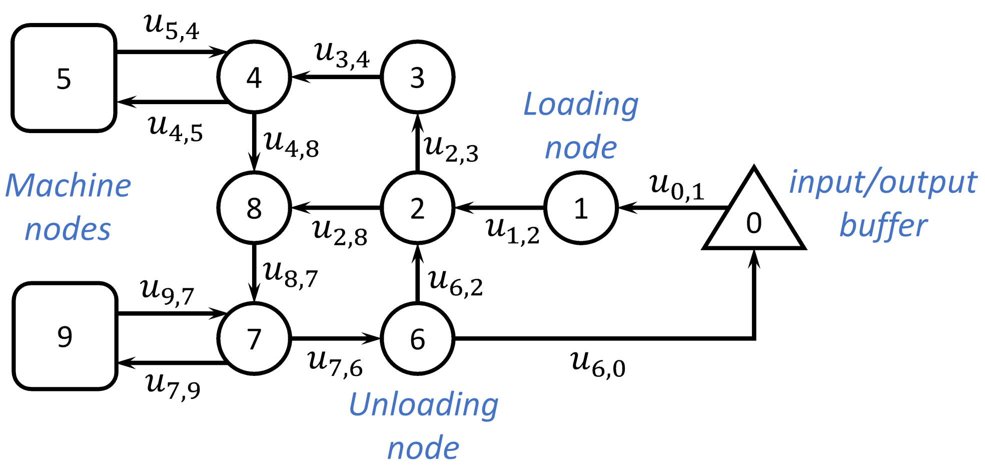


Figure 2.1: Eulerian representation of a generic plant, where: $h_l = 1$, $h_u = 6$, outside node$= 0$, the nodes 5 and 9 are machines (i.e., $\mathcal{M} = \{5, 9\}$)

and corresponds to the number of parts that have been unloaded from the line after being processed by the machines. Finally we can include all the N_u Boolean control signals in a column vector $\boldsymbol{U}(k) \in \{0,1\}^{N_u}$ to represent the whole control action.

Then the nodes indexes are collected into two sets: the *outgoing set* $\mathcal{O}_h$ and the *incoming set* $\mathcal{I}_h$ $h = 1, \ldots, N_n$, defineded as follows:

$$\mathcal{O}_h = \{j : \exists\, u_{h,j}\}, \;\; h = 1, \ldots, N_n$$
$$\mathcal{I}_h = \{j : \exists\, u_{j,h}\}, \;\; h = 1, \ldots, N_n$$

The first one is the set of the indexes related to all nodes that can be reached directly by node h, while the second one is the set of the indexes related to all nodes for which h is a direct destination (both of them including possibly the outside node).

In order to obtain the plant model we define the state of the whole plant as $\boldsymbol{z} = [z_1, \ldots, z_{N_n}]^T$. At each time step the state is updated by means of the column vector $\boldsymbol{v}(k)$. Considering the transitions occurring at time k, it sums or subtracts from the corresponding elements of $\boldsymbol{z}(k)$ the number of parts that are moving in or out a node.

The state update $\boldsymbol{v}(k)$ is computed at each time step as following:

$$\boldsymbol{v}(k) = \begin{bmatrix} \sum_{j\in\mathcal{I}_1} u_{j,1}(k) - \sum_{j\in\mathcal{O}_1} u_{1,j}(k) \\ \vdots \\ \sum_{j\in\mathcal{I}_{N_n}} u_{j,N_n}(k) - \sum_{j\in\mathcal{O}_{N_n}} u_{N_n,j}(k) \end{bmatrix}$$

Finally we can describe the plant behaviour by means of this linear model:

$$\begin{array}{rcrcl} \boldsymbol{z}(k+1) & = & \boldsymbol{z}(k) & + & \boldsymbol{v}(k) \\ N_f(k+1) & = & N_f(k) & + & u_{h_u,0}(k) \end{array} \tag{2.1}$$

The Eulerian formulation describes the part movements inside the plant referring to the nodes as control volumes. Conceptually it can be interpreted as a series of mass conservation equations, because the state indicates how many parts are present in each node at the time k. Taking into account the real application the model is incomplete, thus we have to introduce some operational constraints on control inputs, that must be satisfied at all time steps:

$$\sum_{j\in\mathcal{O}_h} u_{h,j}(k) \leq 1,\ h = 1,\ldots,N_n \tag{2.2a}$$

$$\sum_{j\in\mathcal{I}_h} u_{h,j}(k) \leq 1,\ h = 1,\ldots,N_n \tag{2.2b}$$

$$\sum_{j\in\mathcal{O}_h} u_{h,j}(k) = 0,\ \forall h : z_h(k) = 0 \tag{2.2c}$$

$$\sum_{j\in\mathcal{I}_h} u_{j,h}(k) = 0,\ \forall h : z_h(k) = 1 \wedge \sum_{j\in\mathcal{O}_h} u_{h,j}(k) = 0 \tag{2.2d}$$

The constraints limit the control inputs in order to guarantee that in each nodes can be present one part o none. Specifically constraints (2.2a)-(2.2b) ensure that a part occupying the node h shall reach at most another one, and that only one part shall enter in the node h at the next time step. Moreover the sum of all control signals from an empty node must be zero (2.2c). In conclusion (2.2d) indicates that if the node h is already occupied by a part and it does not become empty in the next step, no one else can reach it.

Because a part has to be held in a machine node until the specific operation is finished, one last group of constraints must be introduced: *the temporal logic constraints*. It is possible to write such constraints defining the time needed by the machine m to end the job by an integer number $L_m \geq 1$ of time steps. Denoting by k_m the time when a new job is started, the constraints become:

$$\sum_{j \in \mathcal{O}_m} u_{m,j}(k) = 0,\ \forall k \leq k_m + L_m, \forall m \in \mathcal{M} : z_m(k) = 1 \tag{2.3}$$

At this point the model is complete and the optimal control problem can be set. The aim of the control policy is to compute at each time step the input vector $\boldsymbol{U}(k)$ minimizing a defined cost criterion and ensuring that all the constraints (2.2)-(2.3) are always satisfied.

In a previous approach [13] [1], the linear model and the constraints have been manipulated in order to obtain a mixed logical dynamical (MLD) formulation of the system model, that allows to solve the problem by means of a large-scale mixed-integer linear program. The MPC algorithm reaches optimal performances, but is limited by the very high computational cost, caused by the complexity of the MLD reformulation.

For that reason a hierarchical control scheme is introduced and a new formulation is presented. The perspective shifts to a Lagrangian model and a two-level approach has been implemented.

2.2 Lagrangian System Model

The major difference between the Eulerian model and the Lagrangian model consists in the fact that the first one approaches the plant from the point of view of the nodes, taking into account their status, while the second one tracks the trajectories of each part in the plant. In order to introduce the Lagrangian model the state of a part has been described adopting the formulation (2.6) reported in the following.

The index $i = 1, \dots, N_p(k)$ denotes the $N_p(k) \in \mathbb{N}$ parts inside the plant at step k. It is important to underline that the number of parts can change during the process execution. Then the sequences (or paths) to be assigned have been indicated by a sets of integers $S = \{1, 2, \dots, N_s\}$, where each element corresponds unequivocally to one sequences. In [7] the formulation of this approach has left open the problem of sequence generation. Indeed the sequences are empirically created by the control designer and not even specific guidelines for the sequence generation have been released. In Chapter 4 this problem will be addressed in order to find some building methods that could be useful for a future automation of the process. Moreover in the same chapter will be solved another problem related to the sequences. Their actual formulation causes a redundancy in the model predictive path algorithm increasing the computational cost.

Then, the Operator $\mathcal{S}(s)$ is defined, which returns $\forall s \in \mathcal{S}$ the sequence that corresponds to the index s. The general structure of each sequence $\mathcal{S}(s)$ is described in the following equation:

$$\mathcal{S}(s) = \left\{ \begin{bmatrix} h_1 \\ g_1 \end{bmatrix}, \dots, \begin{bmatrix} h_p \\ g_p \end{bmatrix}, \dots, \begin{bmatrix} h_{N_s} \\ g_{N_s} \end{bmatrix} \right\} \tag{2.4}$$

The sequence length is defined by N_s and the element position along the sequence is expressed by $p = 1, \dots, N_s$. Each element in the sequence is composed by an array of two values: h_p, g_p. These are two integers indicating a node in the plant, in particular the ordinate set of values h_p corresponds to the list of nodes that must be visited by the part (i.e., the trajectory), while the value g_p, with the same position p in the sequence, denotes the node chosen as the current target to be reached. For example, depending on the specific part routine, g_p could indicates the next machine to be visited or the outside of the plant, in the case the part is finished. The actual sequence formulation has been improved in order to remove the path redundancy, that heavily overloads the computation in the case of long sequences presenting identical segments. In particular a third entry has been added to each element in the sequence:

$$\mathcal{S}(s) = \left\{ \begin{bmatrix} h_1 \\ g_1 \\ f_1 \end{bmatrix}, \dots, \begin{bmatrix} h_p \\ g_p \\ f_p \end{bmatrix}, \dots, \begin{bmatrix} h_{N_s} \\ g_{N_s} \\ f_{N_s} \end{bmatrix} \right\} \tag{2.5}$$

The new value $f_p \in \{1, 0\}$ is a Boolean and its function will be explained more in detail in the Chapter 4. Thanks to this improvement the optimization algorithm avoids to take into account identical scenarios, reducing the computational cost.

In conclusion the plant state of a part i, obtained from the Lagrangian model, has the following structure:

$$\boldsymbol{x}_i(k) = \begin{bmatrix} s_i(k) \\ p_i(k) \\ t_i(k) \end{bmatrix} \tag{2.6}$$

where $s_i(k) \in S$ indicates the sequence assigned to the part i at time k, $p_i(k) \in \mathbb{N}$ indicates the position of the part i along the sequence and $t_i(k)$ counts the elapsed time steps since the part entered the plant. In particular, denoting by $\underline{k}_i$ the step when the part was loaded, $t_i(k)$ is computed by the following expression:

$$t_i(k) = k - \underline{k}_i \tag{2.7}$$

The state variables of all parts are collected in a single vector with a variable length with respect to time (i.e., the length is proportional to the actual number of parts at the time k), that defines the overall Lagrangian state of the whole plant model:

$$X_{N_p(k)}(k) = [\boldsymbol{x}_1(k)^T, \dots, \boldsymbol{x}_{N_p}(k)^T]^T \in \mathbb{N}^{3N_p} \tag{2.8}$$

This kind of formulation are not common in dynamical models, but allows us to simplify the optimal control problem, implementing a hierarchical control, based on a low level path following algorithm and a high-level path allocation algorithm.

2.3 Hierarchical control approach

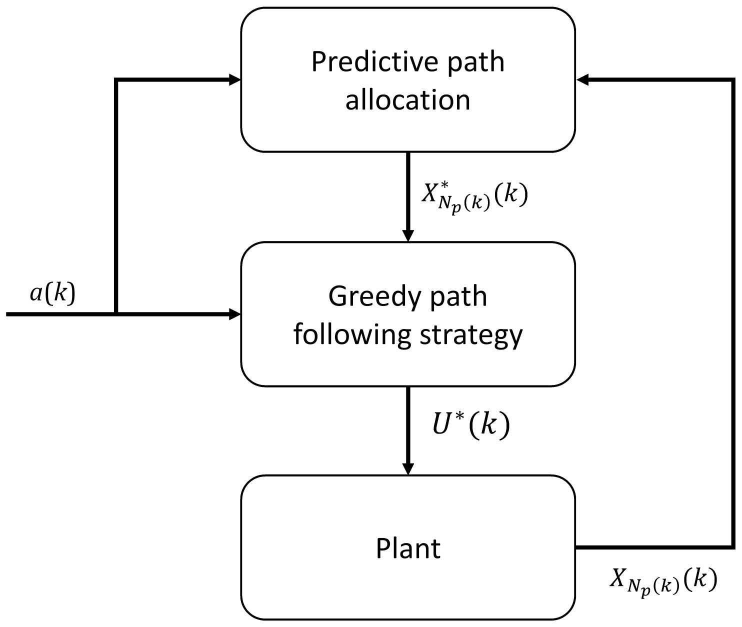


Figure 2.2: Hierarchical Routing Control structure.

The *Hierarchical Routing Control* proposed in the new approach has the aim to simplify the computational complexity of the MPC algorithm. The main idea is to separate the function of ensuring the compliance with the operational constraints, from the function of computing the optimal trajectory for each part in the plant. Furthermore the optimal routing can be found via a simulation-based optimization method. Such control structure is described by the block diagram in figure 2.2. Where $a(k)$ is an exogenous signal, which can be assumed to be a known input or a unknown disturbance; it indicates that at the time k a new part is ready to be loaded into the system.

The two layers of the hierarchical routing control algorithm perform the following tasks:

- a *high-level Model Predictive Path Allocation* assigns recursively to each part a sequence and a position in that sequence, solving a Finite Horizon Optimal

Control problem (FHOCP) with the aim of maximising the efficiency of the plant.

- a *low-level Greedy Path Following Strategy* receives as input the optimal state computed by the FHOCP solver and computes the feasible control inputs needed to move the parts forward along their sequences, satisfying the constraints and solving the conflicts between different part trajectories.

Starting from *low-level Greedy Path Following Strategy* a more detailed explanation of the control algorithms will be now presented, together with some innovations provided by this thesis, in order to comply with the features of the real plant.

2.3.1 Greedy path following strategy

In the following block diagram, fig. 2.3, the structure of the Greedy Path Following Strategy algorithm (GPFS) has been summarized to explain its functioning; for the complete formulation, see [7]. The main task of the algorithm is to compute the control action needed to move the parts forward along their sequence and to update the state of the model. Firstly the actual state is provided to the algorithm that compute the one-step-ahead predicted state of each part, as if they could freely proceed to the next position in the sequence. Then it verifies that no part will conflict (i.e., more than one part will occupy the same node in the next step). In case of conflicting parts the algorithm will choose which part has the highest priority. The state of the part with higher priority is confirmed. To the other parts it assumed to assign their previous state, except for the time counter that is always updated. Such greedy strategy refers to the priority order exposed in the block diagram 2.3. Assuming that there is just one loading node, the GPFS algorithm guarantees that the part with the highest priority is always one, thus there is always one unique choice. This operation is cyclically repeated until

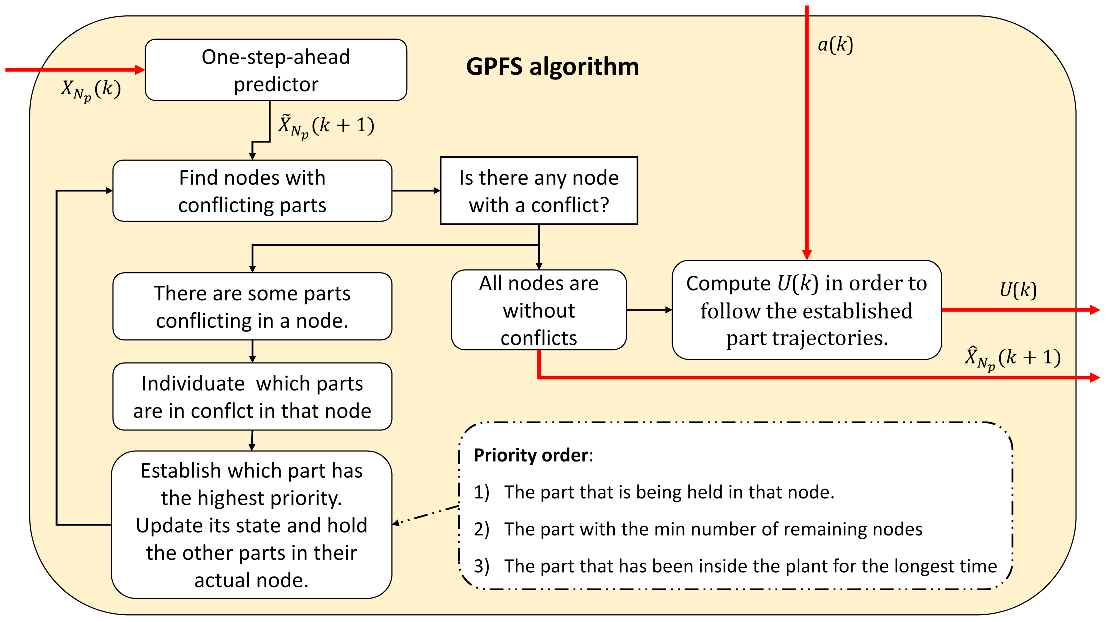

GPFS algorithm
$X_{N_p}(k)$
One-step-ahead predictor
$\tilde{X}_{N_p}(k+1)$
Find nodes with conflicting parts
Is there any node with a conflict?
There are some parts conflicting in a node.
Individuate which parts are in conflct in that node
Establish which part has the highest priority. Update its state and hold the other parts in their actual node.
All nodes are without conflicts
$a(k)$
Compute $U(k)$ in order to follow the established part trajectories.
$U(k)$
$\hat{X}_{N_p}(k+1)$
Priority order:
1) The part that is being held in that node.
2) The part with the min number of remaining nodes
3) The part that has been inside the plant for the longest time

Figure 2.3: *Greedy path following strategy* algorithm scheme.

there are no more conflicting parts, therefore the correct one-step-ahead state is provided and, following a ruled-base procedure, the movements are translated in control signals (i.e., the active transitions). In conclusion, if possible and if the exogenous signal $a(k)$ is equal to 1, the loading control action $u_{0,h_l}(k)$ is activated in order to put a new part in the plant.

The Greedy Path Following Strategy ensures the compliance with the constraints (2.2), while the temporal logic constraints (2.2)-(2.3) are satisfied a priori by a correct sequence generation. Assuming that the machine nodes are repeated consecutively along the path for at least L_m positions, the part is held inside the machine for the same number of steps. When the HRC approach has been adopted for the real plant located in the CNR laboratory, the path following algorithm has been improved, because the generalized formulation does not includes enough constraints. In particular there is a list of transitions that can not occur at the same time. To correctly model the plant behaviour additional conflict solver subroutines has been developed and included in the GPFS algorithm.

They use the same priority order introduced above to compute which transition has the priority. Most of the conflicts generated by these constraints have been solved defining a particular node property, that allows to immediately detect the parts that must be held in position. These topic will be discuss properly in the next chapter.

Finally to be able to correctly update the predicted Lagrangian plant the Greedy Path Following Strategy algorithm has been included in another one (for simplicity has been called *Algorithm 2*), that performs the operation of creating the state for a part that will enter in the plant in the next step and the operation of removing from the overall state vector the state of a part that has been unloaded. The control action $u_{0,h_l}(k)$ and the control action $u_{h_u,0}(k)$, both obtained from GPFS, report to the Algorithm 2 that a part is going to be loaded or is going to be unloaded respectively (the node index 0 corresponds to the outside).

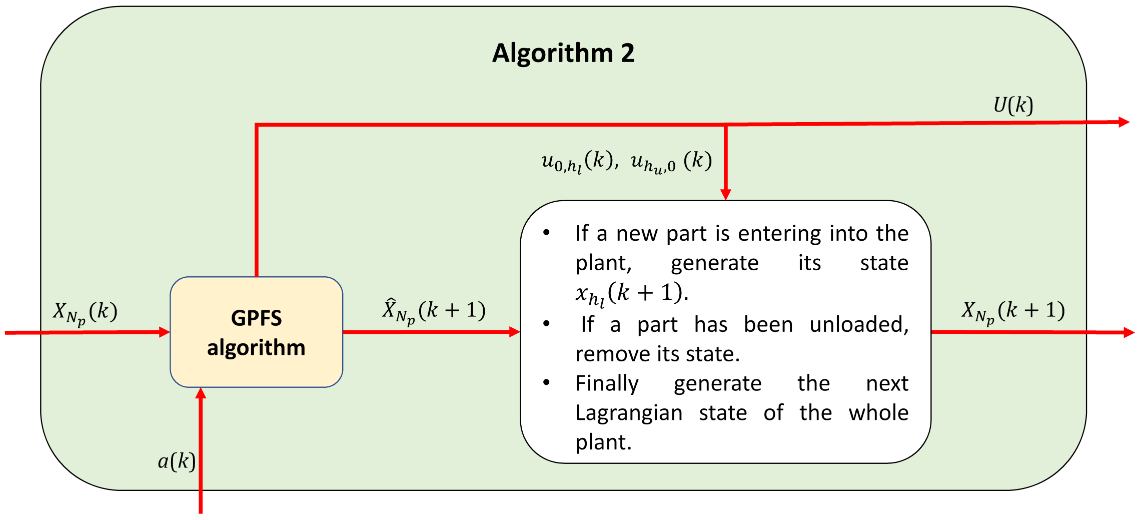

Algorithm 2
$U(k)$
$u_{0,h_l}(k)$, $u_{h_u,0}(k)$
$X_{N_p}(k)$
GPFS algorithm
$\hat{X}_{N_p}(k+1)$
• If a new part is entering into the plant, generate its state $x_{h_l}(k+1)$.
• If a part has been unloaded, remove its state.
• Finally generate the next Lagrangian state of the whole plant.
$X_{N_p}(k+1)$
$a(k)$

Figure 2.4: *Algorithm 2* scheme.

In conclusion the control policy can be assumed as the description of the plant behaviour (which is forced to act according to the low-logic rules). Indeed Algorithm 2 receives as input the actual overall state and the exogenous signal $a(k)$, returning as output the correct one-step-ahead prediction of the next state. Moreover it provides as an additional information the feasible control action that allows each part to move forward along the assigned sequence, in order to perform the predicted state evolution. This plant model can be expressed by the following set of functions $f_{(N_p(k+1),N_p(k))} : 3\mathbb{N}^{N_p(k)} \to 3\mathbb{N}^{N_p(k+1)}$, that describe the evolution of the state:

$$X_{N_p(k+1)}(k+1) = f_{(N_p(k+1),N_p(k))}(X_{N_p(k)}(k), a(k)). \tag{2.9}$$

Thanks to this plant model representation, the high-level control algorithm can predict the system evolution to solve the FHOCP without explicitly including the challenging constraints (2.2)-(2.3) in the problem formulation. In this way it can ignore how the lower controller operates and it is able to impose a desired motion to the parts just changing the actual state with a compatible one, after having predicted how the state trajectory will evolve in free motion.

The vector of the control signals $\boldsymbol{U}(k)$ is an output of the system model described by Algorithm 2 and it constitutes the information needed to impose the same behaviour to the real system. The plant is considered as a black box model from the high-level controller point of view: assigning a state as input it is possible to compute its evolution and the corresponding control action, as represented in the figure 2.5.

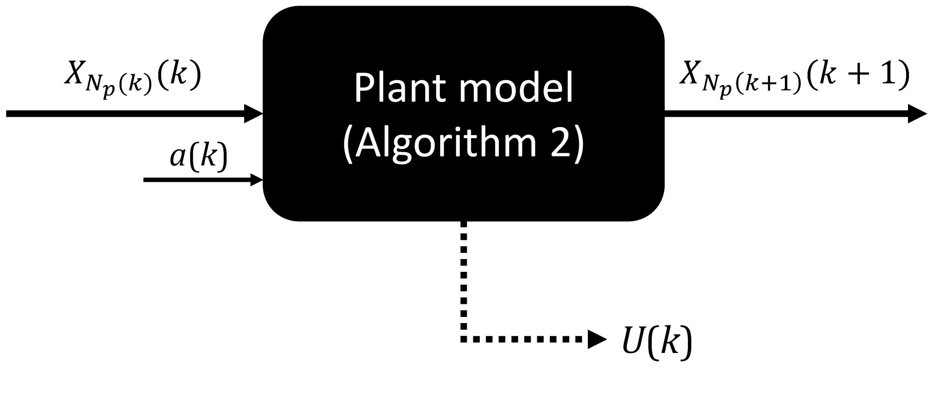


Figure 2.5: *Algorithm 2* as a black box model, useful for a simulation-based optimization method

2.3.2 Model Predictive Path Allocation

The high-level logic represents the core of the MPC. At each time step the task of the controller is to allocate a sequence $s \in S$ (and a position p along that sequence) to each part, in order to optimize the performances of the plant. For this purpose a FHOCP is solved predicting the system behaviour by means of the equations (2.9). Below the features of the *Model Predictive Path Allocation* (MPPA or Algorithm 3) control policy will be introduced as proposed by Fagiano et al. in their paper.

Firstly the actual state is processed for generating the sets of the compatible states $\mathcal{X}_i(k),\ i = 1, \dots, N_p(k)$, whose elements are defines as a couple of values (s, p), indicating a sequence index s and a position p inside that sequence. Each set $\mathcal{X}_i(k)$ corresponds to the i-th part inside the plant, and represents all the compatible states with respect to the i-th Lagrangian state $x_i(k)$, considering the two first elements of the state. In two compatible states, the positions along the related sequences must correspond to the same node and must indicate the same target. In addition when the part is held by a machine node, the state is defined compatible taking into account also the previous positions, back to the ones in which the machine has begun the job. In the paper [7] the sets have been formulated as reported below:

$$\text{if } \mathcal{S}(s_i(k))^{(1,p_i(k))} \notin \mathcal{M}:$$

$$\mathcal{X}_i(k) = \left\{ \begin{array}{l} (s,p) \in S \times \mathbb{N}: \\ \mathcal{S}(s)^{(1,p)} = \mathcal{S}(s_i(k))^{(1,p_i(k))} \\ \wedge\, \mathcal{S}(s)^{(2,p)} = \mathcal{S}(s_i(k))^{(2,p_i(k))} \end{array} \right\} \tag{2.10a}$$

$$\text{else if } \mathcal{S}(s_i(k))^{(1,p_i(k))} \in \mathcal{M}:$$

$$\mathcal{X}_i(k) = \left\{ \begin{array}{l} (s,p) \in S \times \mathbb{N}: \\ \mathcal{S}(s)^{(1,p-j)} = \mathcal{S}(s_i(k))^{(1,p_i(k)-j)}, \\ j = 0, \ldots, k - k_{\mathcal{S}(s_i(k))^{(1,p_i(k))}} \\ \wedge\, \mathcal{S}(s)^{(2,p)} = \mathcal{S}(s_i(k))^{(2,p_i(k))} \end{array} \right\} \tag{2.10b}$$

where the operator $S(s)$ returns for each index s the corresponding sequence, the two operators $\mathcal{S}(s_i(k))^{(1,p_i(k))}$ and $\mathcal{S}(s_i(k))^{(2,p_i(k))}$ define the first and the second value, respectively, of the p_i-th element belonging to the sequence s_i. The time step when the i-th part has started the operation in machine $m = \mathcal{S}(s_i(k))^{(1,p_i(k))}$ has been defined by $k_{\mathcal{S}(s_i(k))^{(1,p_i(k))}}$. The sequences redundancy problem, already introduced during the sequence formulation, has been solved adding a new condition in (2.10) (in both cases), that excludes a possible compatible state if the related third element f_p is equal to zero. That concept will be better explained in the Chapter 4, that deals with the sequence features. It is important to underline that the sets $\mathcal{X}_i(k)$ contain at least one pair, the one corresponding to the actual Lagrangian state of the part. The elements of the compatible state sets represent the optimization variables of the finite horizon optimal control problem, because indicate the pull of possible path to be assigned to a part.

After building the sets (2.10), the Algorithm 3 computes the optimal state (i.e., assigns the optimal path to each part), solving the following FHOCP, directly reported from [7]:

$$\min_{(\sigma_i,\pi_i),\, i=1,\dots,N_p(k)} \sum_{o=0}^{N} \ell_{N_p(o|k)} \left(X_{N_p(o|k)}(o|k) \right) \tag{2.11a}$$

subject to

$$\boldsymbol{x}_i(0|k) = \left[\sigma_i, \pi_i, t_i(k)\right]^T,\ i = 1, \dots, N_p(k) \tag{2.11b}$$

$$\begin{gathered} X_{N_p(0|k)}(0|k) = \\ \left[\boldsymbol{x}_1(0|k)^T, \dots, \boldsymbol{x}_{N_p(k)}(0|k)^T\right]^T \end{gathered} \tag{2.11c}$$

$$\begin{gathered} X_{N_p(o+1|k)}(o+1|k) = \\ f_{(N_p(o+1|k),N_p(o|k))}(X_{N_p(o|k)}(k), a(o|k)), \\ o = 0, \dots, N-1 \end{gathered} \tag{2.11d}$$

$$(\sigma_i, \pi_i) \in \mathcal{X}_i(k),\ i = 1, \dots, N_p(k) \tag{2.11e}$$

Where $N \in \mathbb{N}$ denotes the prediction horizon and $l_{Np}(X_{N_p})$ represents the stage cost function chosen by the designer, $X_{N_p(o|k)}(o|k)$, $\boldsymbol{x}(o|k)$ indicates the predictions of overall plant state and part Lagrangian states, respectively, computed at time k and pertaining to time $k+o$. The prediction of the exogenous signal, indicating the necessity to load a new part, is denoted by $a(o|k) \in \{0,1\}$, $o = 0, \dots, N-1$. During the tests, reported in the final section of this thesis, it has always been considered as known. The constraints (2.11b) and (2.11c) indicate that the optimization variables, the pairs (σ_i, π_i), $i = 1, \dots, N_p(k)$, are the sequences indexes and the corresponding positions in those sequences to be assigned to the Lagrangian states. Constraints (2.11d) correspond to the system model defined by Algorithm 2, introduced above. The last set of constraints (2.11e) indicates that the optimization variables are chosen from the sets of the sequences and positions compatible with the actual part states. At the end (σ_i^*, π_i^*), $i = 1, \ldots, N_p(k)$ are the solution to (2.11), which is used to compute the optimal overall state of the system $X^*_{N_p(k)}(k)$. Finally the optimal state is applied to the GPFS algorithm in order to compute the control inputs.

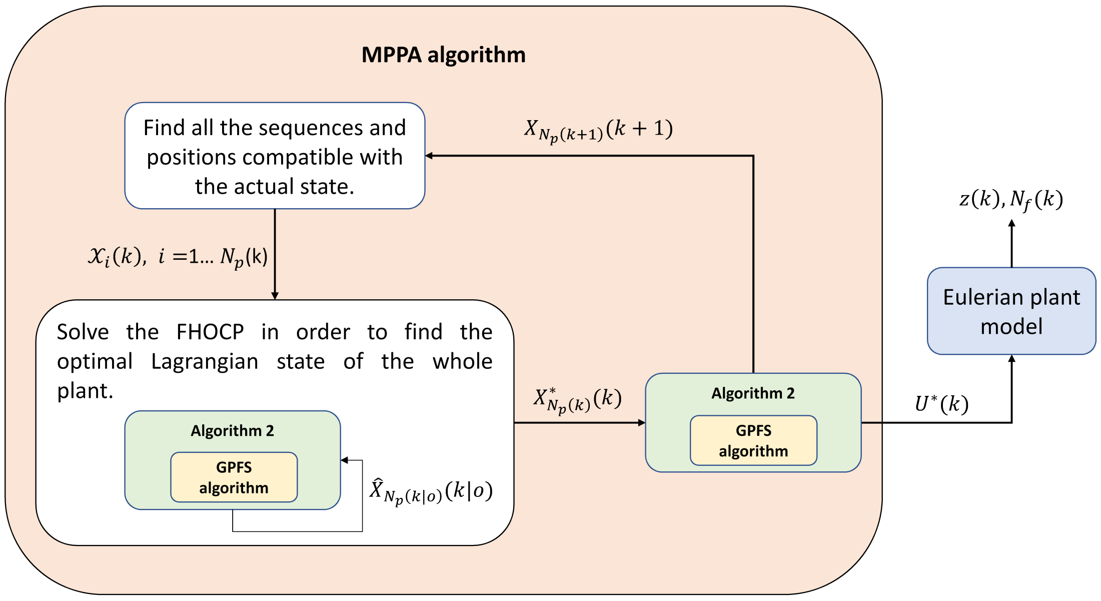

MPPA algorithm
Find all the sequences and positions compatible with the actual state.
$X_{N_p(k+1)}(k+1)$
$\mathcal{X}_i(k),\ i = 1\ldots N_p(k)$
Solve the FHOCP in order to find the optimal Lagrangian state of the whole plant.
Algorithm 2
GPFS algorithm
$\hat{X}_{N_p(k|o)}(k|o)$
$X^*_{N_p(k)}(k)$
Algorithm 2
GPFS algorithm
$U^*(k)$
Eulerian plant model
$z(k), N_f(k)$

Figure 2.6: *Model predictive path allocation (Algorithm 3)* scheme.

In the real case the feedback loop can not be closed directly on the Lagrangian state, because the sensors of the handling system just check the position of the pallets inside the nodes (i.e, the Eulerian state) and take track of the target to be reached by each pallet. Two control strategies can be adopted: a verification Eulerian state feedback or an estimated Lagrangian state feedback.

In the first option, as showed in the block diagram in fig. 2.7, the control action is applied to the Eulerian model of the plant in order to obtain a state comparable with the one measured by the sensors. In that case the feedback loop becomes only a validation of the prediction, considering the Lagrangian states as

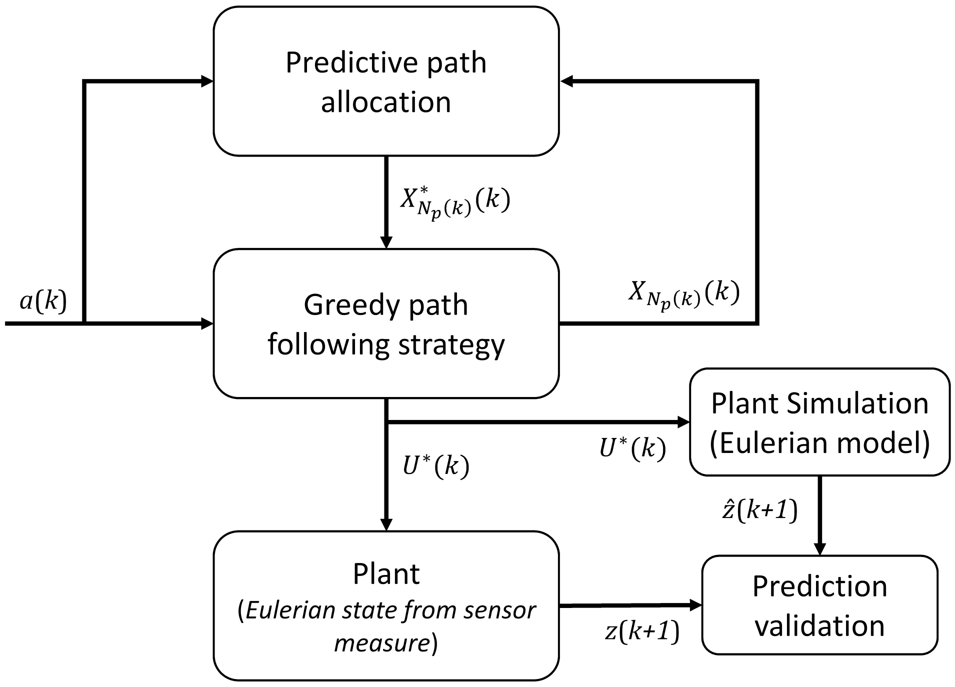

Predictive path allocation
$X^*_{N_p(k)}(k)$
$a(k)$
Greedy path following strategy
$X_{N_p(k)}(k)$
Plant Simulation (Eulerian model)
$U^*(k)$
$U^*(k)$
$\hat{z}(k+1)$
Plant (*Eulerian state from sensor measure*)
Prediction validation
$z(k+1)$

Figure 2.7: Feedback loop with Eulerian state validation

an image of the process, known and updated only by the controller.

Otherwise a state observer could be implemented to estimate the next overall Lagrangian state, by means of the actual and past inputs $U(k), U(k-1), ...$ and the measured Eulerian states of the next and previous steps $z(k+1), z(k), z(k-1), ...$ This control feedback is shown below in figure 2.8.

Only the first control strategy has been tested (figure 2.7), because in this thesis the plant is assumed to operate in nominal condition. Therefore no movement faults occur and it is sufficient that at each step the feedback loop verifies the consistency between the plant model and the real system.

In case movement faults would be considered, two types of error may arise with respect to the actual state measured by the sensors:

- One or more parts may not move to the next node. In this case it would be sufficient to reassign to those parts their previous Lagrangian state (except for the time counter).

- One or more parts may move in a wrong node. In this case, being able to isolate the parts that have moved incorrectly, it would be sufficient to assign them a new sequence, compatible with the node they are in and the target they must reach.

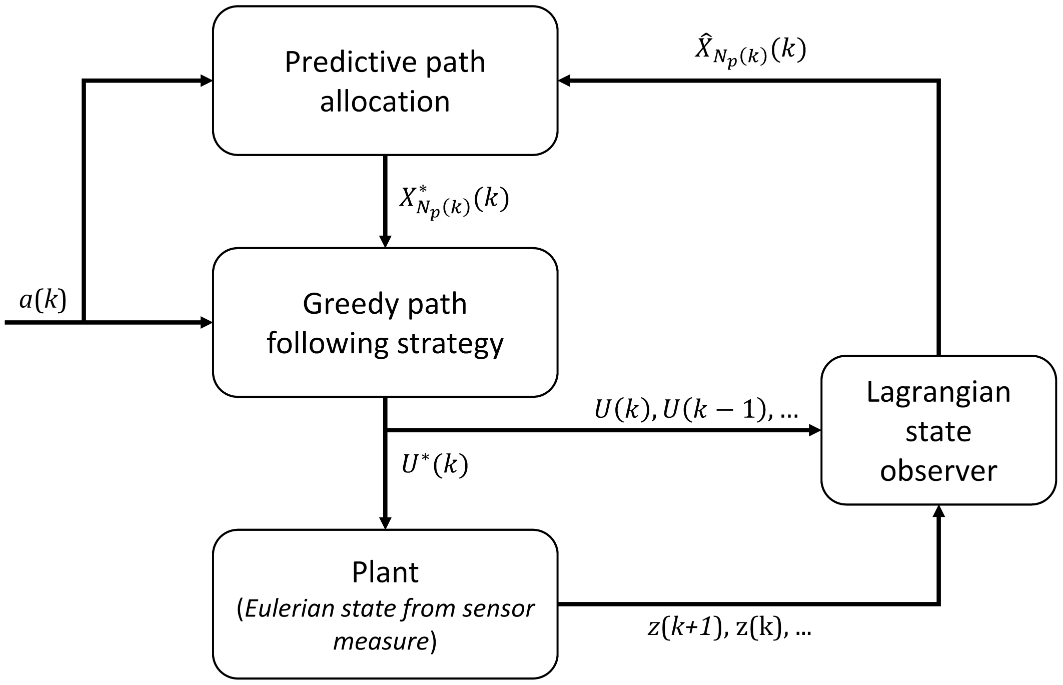

Predictive path allocation
$\hat{X}_{N_p(k)}(k)$
$X^*_{N_p(k)}(k)$
$a(k)$
Greedy path following strategy
$U(k), U(k-1), \dots$
$U^*(k)$
Lagrangian state observer
Plant (*Eulerian state from sensor measure*)
$z(k+1)$, z(k), …

Figure 2.8: Lagrangian Feedback loop with state observer.

The actual FHOCP (2.11) follows a Move Blocking approach [14], in which the Lagrangian states can be changed only at the first step of the prediction horizon (i.e. the pair (σ_i^*, π_i^*) consists in the two controllable elements in the state vector of the i-th part, indeed the time $t_i(k)$ is just a counter). This is a heavy limitation, meaning that a very high number of sequences must be precomputed in order to cover all the possible paths for every possible configurations of the pars. If the pool of sequences does not cover all the possibilities, in some cases, especially with many parts in the plant, the controller may not be able to avoid the lockout during the prediction, because is running in open loop and needs a specific sequence to avoid the lockout from the beginning. Also because a non optimized sequence, for example a default path, is assigned to a new part entering in the plant during the prediction.

For these reasons an improvement of the optimization problem will be adopted, considering the possibility to change the Lagrangian state of each part also during the prediction. In this way the sequence generation problem becomes feasible, because all the possible part trajectories can be described by a finite number of sequences. Moreover the model simulation is able to reproduce exactly the movements of the parts along the horizon. These aspects will be described in more detail in Chapter 4 regarding the sequence generation and Chapter 5 regarding the optimization problem.

Chapter 3

Real case implementation

In this thesis the HRC approach is implemented and tested on a real plant for the fist time. It is adopted in an automated demanufacturing plant for testing and repairing electronic boards. In the following section the plant functioning will be presented and the operations performed by the different machines will be explained. Then the handling system will be described by means of a directed graph in order to develop the Eulerian system model. Furthermore new constraints will be identified and added to those formulated in 2.2. In particular these constraints will be collected in two classes:

- constraints on transitions related to specific nodes
- constraints on pairs of transitions with no node in common

Finally the two corresponding subroutines will be included in the GPFS to correctly represent the plant behaviour from a Lagrangian point of view.

3.1 Plant description

The approach of Hierarchical Routing Control, introduced in the previous chapter, has been implemented to a real case: a demanufacturing plant designed to manage the testing, the repair or dismantling of electronic boards. The plant is located in laboratory of the Institute of Industrial Technologies and Automation (STIIMA), National Research Council (CNR), Italy. Its structure and components are shown in the scheme 3.1.

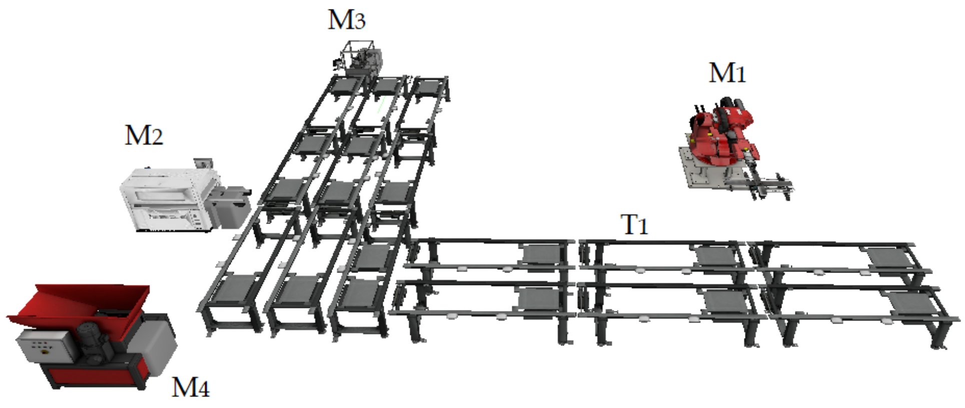


Figure 3.1: Demanufacturing plant structure

The components of the plant consist in the following machines:

- Machine M_1 is the Load/Unload Robot Cell: its purpose is to unload the pallet occupying the transport module adjacent to the manipulator, replace the electronic board with a new one, then reload the pallet in the plant. Or i could directly insert or remove a pallet from the plant, if there is an external pallet buffer.

- Machine M_2 is the Testing Machine: is the machine in which the boards are tested to determine whether they need to be repaired, dismantled or if they work correctly. For each of these cases the machine determines which new target the board should reach.

- Machine M_3 is the Reworking Machine: in case the board has been evaluated as not working, but still repairable, this machine has the task to rework the board for repairing the malfunction that has been detected by M_2.

- Machine M_4 is the Discharge Machine: all boards that can not be repaired by M_3 are unloaded from the pallet and destroyed. Then the empty pallet is positioned on the transport line.

- The handling system is composed of fifteen transport modules T_n, $n = 1, \dots, 15$: they form a flexible modular transport line, that connects the machines to each other and is able to move a part individually along its specific path. The particular scheme of the modules allows to reach one machine from another by making different routes, and this determines the degrees of freedom that enables to optimize the overall handling of the boards in the plant. The components of these module are translated into the nodes of the control volumes graph, adopted in the Eulerian system modelling. Their features will be reported below.

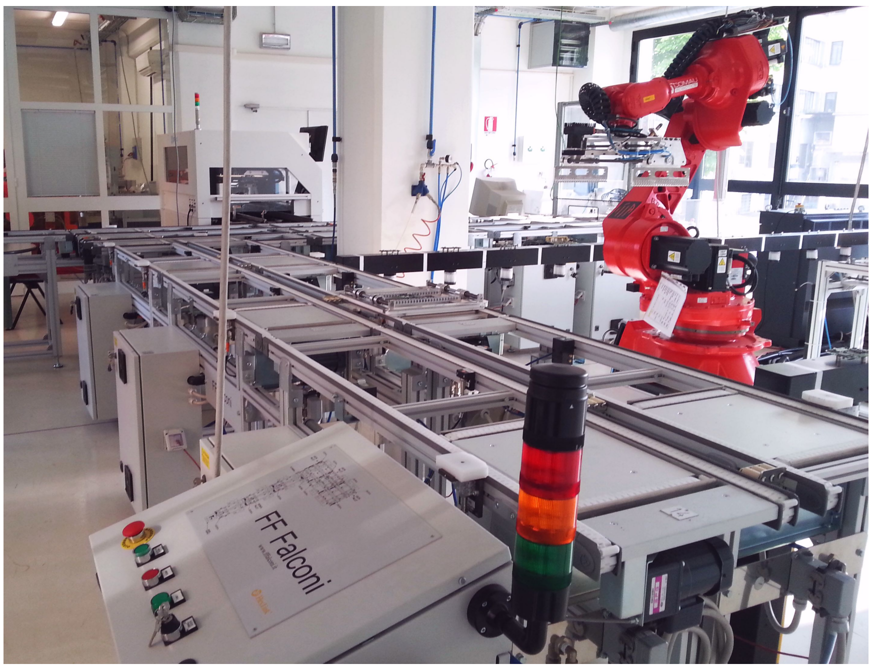


Figure 3.2: Demanufacturing plant in STIIMA-CNR laboratory

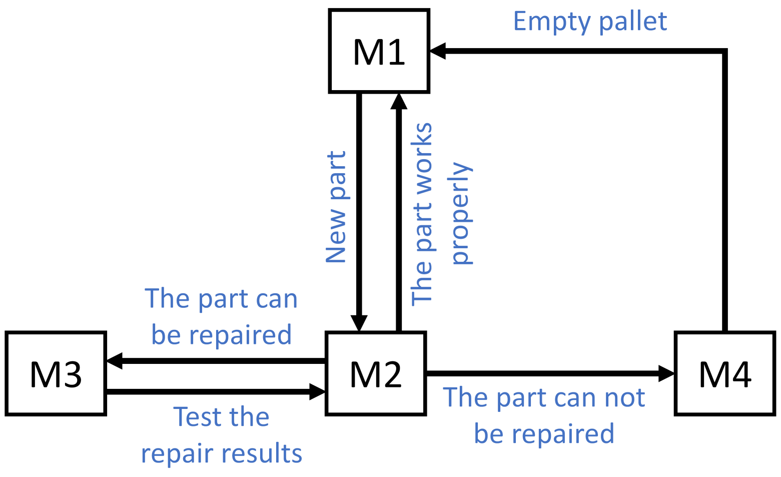


Figure 3.3: Targets order scheme, M_1 is the Load/Unload Robot Cell, M_2 Testing Machine, M_3 Reworking Machine, M_4 Discharge Machine

The operation of the system is described by a routine that may vary depending on the characteristics of the processed electronic board. Firstly the board is loaded on a pallet and enters in the plant, its initial target is the machine M_1 in order to analyze the board conditions and to identify its next target. As previously explained, the result of the Testing Machine can correspond to three different paths: the first one directed to the Reworking Machine, in case the board can still be repaired, the second one directed to the Discharge Machine, if the board must be destroyed, or finally to the Loading/Unloading Cell, if the board works correctly. When the board is sent to the Reworking Machine, after the restoration process, it is sent back to the Testing Machine, to check the new state of the board. When it is sent to the Discharge Machine, the board is removed from the pallet and is destroyed. In this case the pallet is sent back to the Loading/Unloading Cell to assign it a new board.

The test performed by machine 1 is what introduces uncertainty into the model. In fact not knowing exactly what routine the part will have to perform is a hard problem for a predictive control model, which in order to be robust to

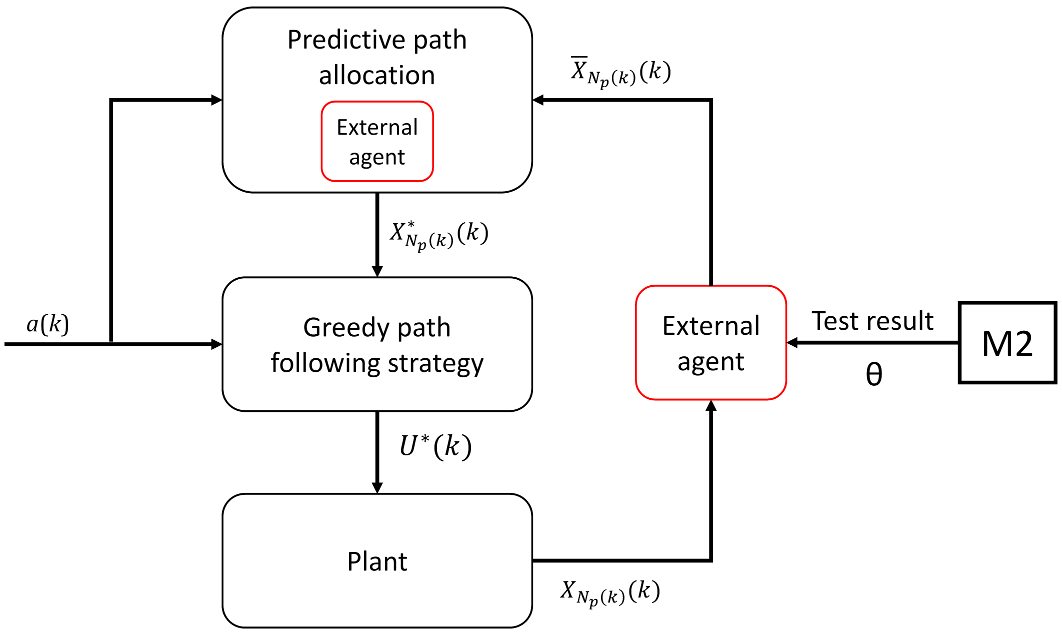


Figure 3.4: HRC with external agent scheme

lockout should take into account all possible test results and simulate all possible scenarios. It also becomes necessary to modify the route of a part according to the machine that will be assigned as target after the test. A sequence allocated to a generic part describes the entire path it will follow through the plant, always indicating as the last node the one representing the outside. So, in order to change the target to a part it is necessary to introduce an external agent that replaces his Lagrangian state just before it leaves the testing machine. In this way a new sequence will be assigned to the part according to the routine it will have to perform. This replacement must also be performed during the simulation to predict correctly the behavior of the plant. For this reason a Move Blocking approach, which allows the controller to change the trajectory of a part only at the beginning of the prediction horizon, becomes even more disadvantageous. Within the prediction horizon, after the external agent has replaced the state of a part, the control can not decide the sequence for reaching the new target and the control on that part is completely lost. And even if the test results were known a priori, the control logic would have to define an even larger number of sequences to cover all the path combinations that can occur.

At each step, before the control reads the actual system state, the external agent checks if a part is present in the node indicating the M_2 machine and checks if it will leave the machine at the next step. At this point, depending on the test results it assigns a sequence starting from the same node but with a different target to move the part towards the next machine. The new target is provided to the external agent by a parameter denoted as θ and generated by the M2. It can assume 4 values to indicate respectively: no test performed, next target M_3, next target M_4 or next target M_1. Obviously the model of the external agent must be included in the MPPA algorithm to reproduce its behavior during the prediction of the system behaviour, see figure 3.4. Considering θ as known it is possible to predict exactly the different routine of each part, otherwise it will have to be estimated according to the probability of the outcome of the tests.

3.2 Eulerian plant modelling

The control method proposed in this thesis deals with managing the movement of parts through the plant, while the operations that take place inside the machines are assumed to be controlled locally. Therefore, the machines (Load/Unload Cell and the Testing, Reworking and Discharge Machines) are simply seen as nodes where the part must be held for a certain number of steps, which corresponds to the time needed to complete the specific operation (described by L_m in the temporal-logic constraints 2.3). Those machine nodes are connected to the adjacent transport module node by a pair of transitions that indicate the possibility of accessing and exiting the machine. The remaining parts of the plant consists of the 15 transport modules, which in previous study, see [13] [1], have been modelled as in the following lines.

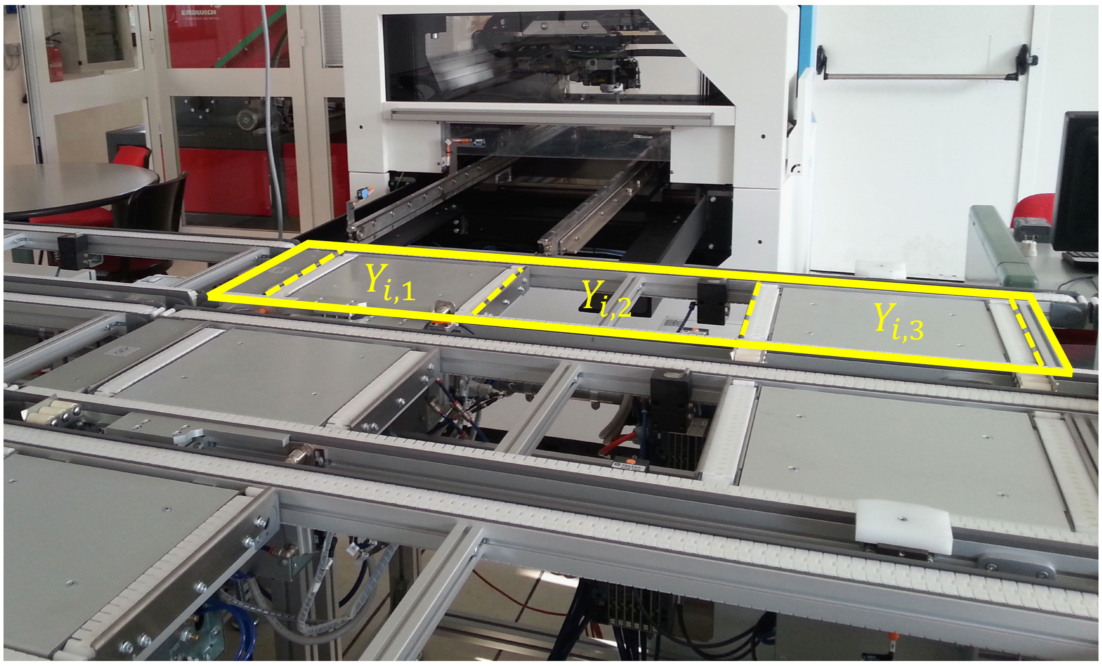


Figure 3.5: Transport module T_i photo

A transport module T_i is composed of three Buffer Zones $Y_{i,j}$, with $j = 1, 2, 3$ indicating the three possible positions that one pallet can occupy inside it. From a Buffer Zone (BZ) a pallet can move in an adjacent BZ in the same module or, if the transport line configuration allows it, in a BZ belonging to another module. Referring to [13], on a transport module the movements of a pallet from one of its three BZs have been defined as S_n, $n = 1, \dots, 36$ and called *Control Sequences*, in the fig. 3.6 they are indicated by the red arrows (they must not be confused with the concept of sequence introduced in Chapter 2). Moreover, a low-level control logic has been realized, which in this work is considered as already implemented. It allows to translate the control inputs, representing the transitions between the nodes, in the Control Sequences and finally in the signals for the actuators of the transport modules.

Then the entire plant can be represented by aggregating the model of the individual transport module and identifying the possible transitions between the buffer zones between one module and another. In the diagram in figure 3.7 the

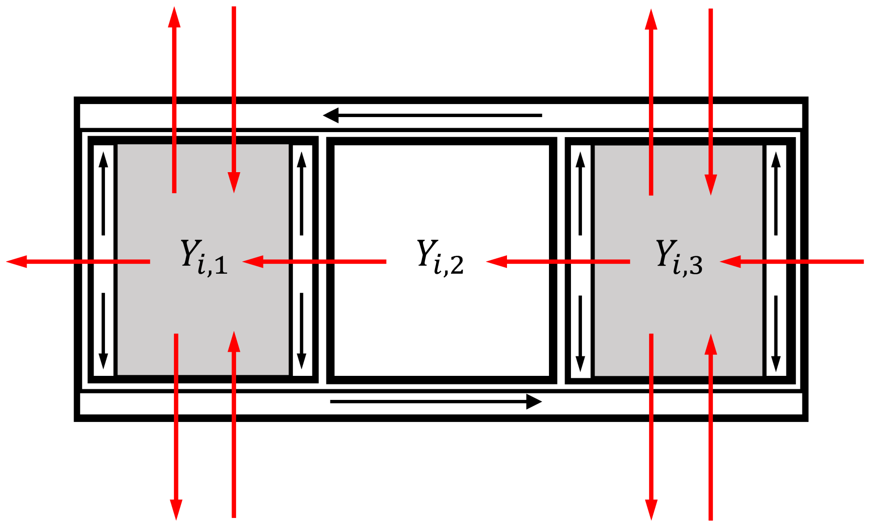


Figure 3.6: Transport module T_i

entire line has been modeled according to the representation with Buffer Zones and Control Sequences.

The representation as direct graph is obtained simply by replacing the Buffer Zones with nodes and connecting the nodes with the transitions defined by the related Control Sequences. Each of these transitions corresponds to a binary control action that determines the displacement of the part from one node to another. ($u_i, j = 1$ indicates that the command to move a part from the node h_i to the node h_j is active).

Unlike the model presented in paper [13], a further transition previously ignored is added $u_{21,23}$ (from node 21 to node 23) and a new node has been introduced: the node 36, which represents the outside of the plant and is connected with the Load/Unload Cell, node 32. This last one represents both the load and the unload node, respectively h_l and h_u. The outside node was not considered because it was not needed by the previous approach, which did not consider the possibility to load and unload a pallet from the plant, but only to be able to replace or load a new electronic board inside of it (these additions are reported in red in the di-graph in fig. 3.8). In conclusion the dynamical model can be described by the equations already introduced in the chapter defining the Eulerian system model formulation.

$$\begin{aligned} \boldsymbol{z}(k+1) &= \boldsymbol{z}(k) + \boldsymbol{v}(k) \\ N_f(k+1) &= N_f(k) + u_{32,36}(k) \end{aligned} \tag{3.1}$$

Where:

- $\boldsymbol{z(k)}$ is the array collecting the state at time k of all the nodes in the plant, $\boldsymbol{z} = [z_1, \dots, z_{36}]^T$ ($N_n = 36$);
- $\boldsymbol{v}(k)$ is the Eulerian state update vector, derived as function of the 53 Boolean control actions ($U(k) \in \{0,1\}^{53}$, $N_u = 53$);
- the output of the system $N_f(k+1)$ indicates the number of finished part at the step $k+1$, and it is function of the unloading control action $u_{32,36}(k)$, that corresponds to a part moving from M_1 to outside the plant, (node 36).

Moreover N_m=4 is number of machine nodes, in fact the nodes 32, 33, 34, 35, denote the machines M_1, M_2, M_3, M_4, respectively.

The constraints that characterize the real plant have been discussed in the previous chapter and have been considered in the control algorithms to directly limit the system inputs and provide a feasible system behaviour. As anticipated, the transport modules must satisfy a further set of constraints, determining which transitions can not occur at the same time. Thus it will be necessary to modify the GPFS algorithm to take into account also these additional limitations due to the configuration of the transport line.

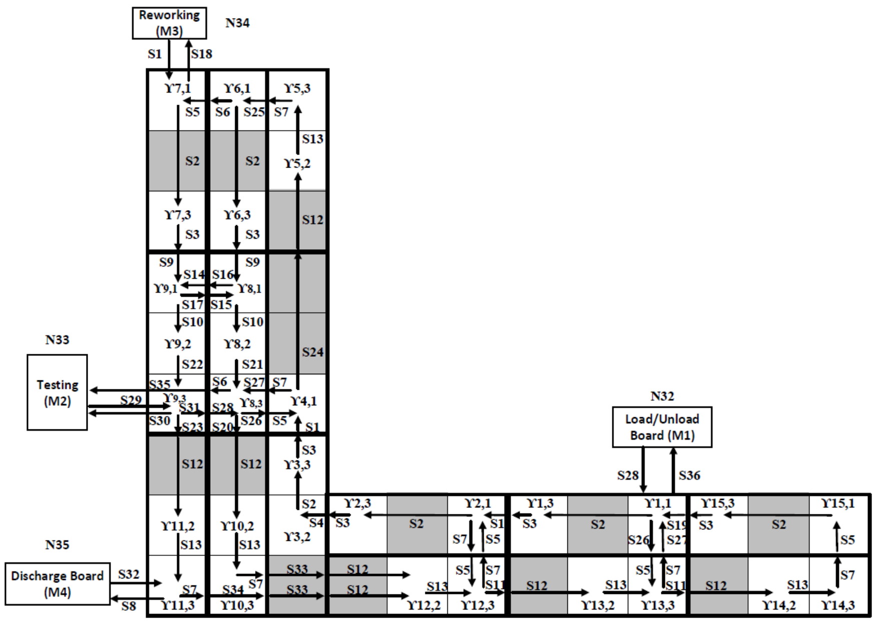


Figure 3.7: Buffer zones and Control Sequences model [1]

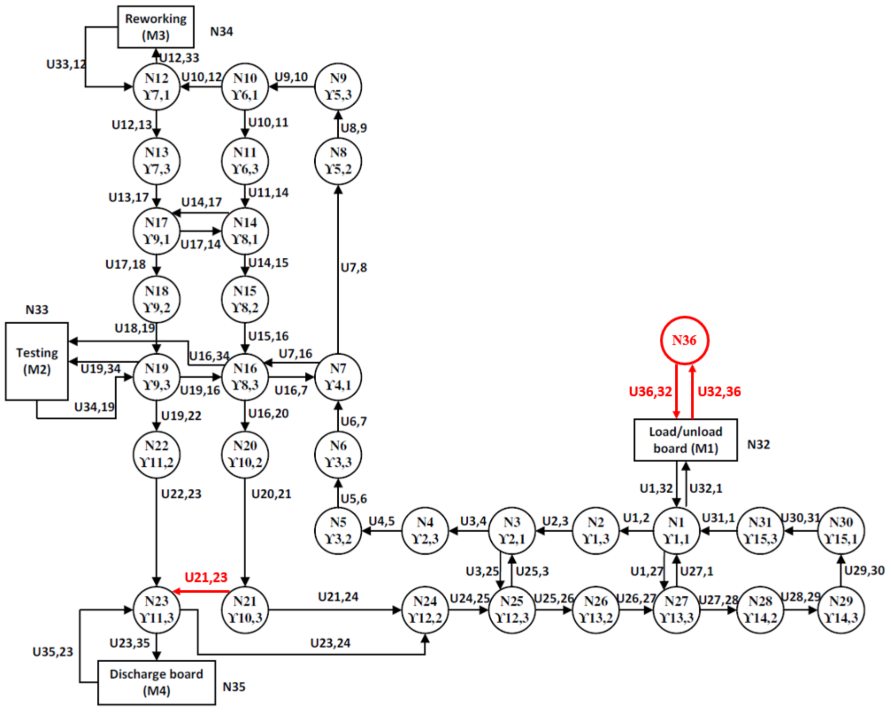


Figure 3.8: Directed Graph model [1]

3.3 Additional constraints

The constraint that must be considered in order to have a faithful representation of the plant behaviour are denoted by the following formulation:

$$\sum_{j \in \mathcal{C}_i} u_j(k) \leq 1,\ i = 1, \dots, N_c \tag{3.2}$$

Where N_c is the number of additional constraints. Each one of them is defined by a set $\mathcal{C}_i \subset \{1, \dots, N_u\}$ of transitions that can not occur at the same time, $N_u = 53$ corresponds to the number of control inputs (i.e. the number of possible transitions in the directed graph representing the plant). In particular the i-th constraint indicates that just one of the transitions, related to the control actions belonging to the set $\mathcal{C}_i$, can occur at the step k. These transitions may not refer to the same node, because the additional constraints take into account the structural limitations of the transport modules and their configuration. Furthermore two different subclasses of this type of constraints have been defined depending on the characteristics of the corresponding set of control inputs $\mathcal{C}_i$. In the first subclass the set indicates transitions in or out of a specific node, whereas in the second one the set indicates pairs of transitions that have no nodes in common. The latter depends on the particular layout of the transport modules and on the limited number of pallets they can move at the same time. Two different subroutines have been implemented in the path following algorithm in order to comply with both categories of constraints.

3.3.1 Constraints on transitions related to specific nodes

It is possible to collect all the constraints related to a node in a set that characterizes the properties on the transitions of that node. In particular it can be observed that for some nodes a common property $\mathcal{P}$ can be defined. For this reason, they have been distinguished from the others and have been defined as *Constrained Nodes* (CN), to indicate that transitions involving that specific node must follow certain limitations. The constraints on the transitions of these nodes can be summarized in few rules:

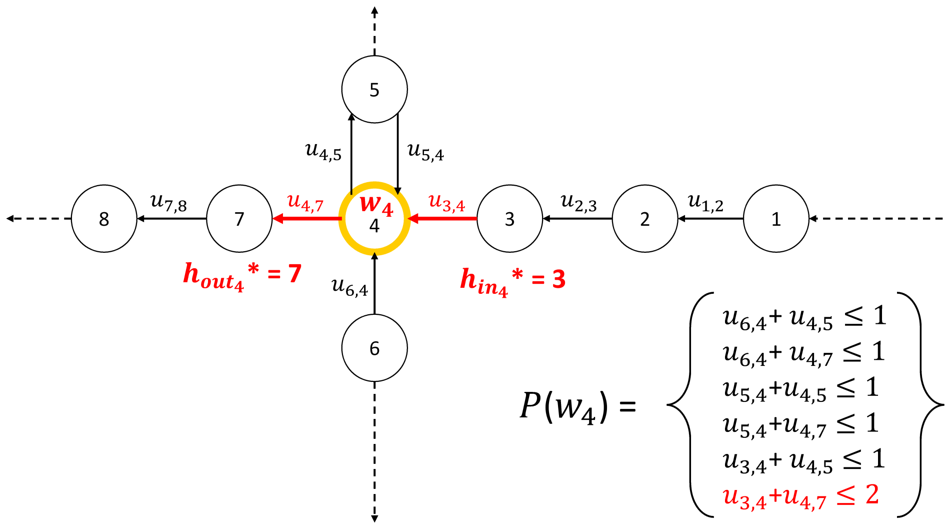
- If there is a part in a CN and that part is moving out, it is forbidden for another part, that is leaving an adjacent node, to enter in the CN at the same time step.
- A couple of nodes (h^*_{out}, h^*_{in}) can be assigned to some adjacent CNs. The first one h^*_{out} corresponds to the destination for the part that is leaving the CN and the second one h^*_{in} corresponds to the actual node for the part that is entering in the CN. That couple determines the only pair of transitions that can occurs simultaneously (not respecting the first rule).

The set of constrained nodes is denoted as $\mathcal{W}$, including the indexes of all nodes in the plant $h = 1\ldots, N_n$ with the property $\mathcal{P}$, that is formulated below. Thus the CNs are indicated as $w_j,\ j \in \{1, \ldots N_n\}$, where N_n is the total number of nodes and $N_w = card(\mathcal{W})$ is the number of constrained nodes, with $N_w < N_n$. The property $\mathcal{P}$ of those type of nodes is described by the two rules previously introduced and can mathematically be defined as:

$$\mathcal{P}(w) = \{u_{w,h_{out}}(k) + u_{h_{in},w}(k) \leq 1, \forall h_{out} \in \mathcal{O}_w \wedge \forall h_{in} \in \mathcal{I}_w, \text{except for a}$$
$$\text{specific pair } (h^*_{out}, h^*_{in}) \text{ such that } u_{w,h^*_{out}}(k) + u_{h^*_{in},w}(k) \leq 2\}$$

An example is proposed in figure 3.9, where the CN is the fourth node w_4, underlined with the colour yellow, and the two transitions that can occur at the same time are $u_{4,7}$ and $u_{3,4}$, represented with the red arrows. These transitions are determined by the couple of nodes $(h^*_{out} = 7, h^*_{in} = 3)$, and the specific property $\mathcal{P}(w_4)$ related to the CN w_4 is defined by the set of constraints shown next to the figure. In particular the constraint $u_{3,4} + u_{4,7} \leq 2$ is implicit and indicates which transitions are free to occur at the same time, in fact it is always satisfied.

It is possible to notice that if a part is going to move in a occupied CN, the only case in which the part can actually move is the following: the part is in the



5
$u_{4,5}$
$u_{5,4}$
8
$u_{7,8}$
7
$u_{4,7}$
w_4
4
$u_{3,4}$
3
$u_{2,3}$
2
$u_{1,2}$
1
$h_{out_4}* = 7$
$u_{6,4}$
$h_{in_4}* = 3$
6
$P(w_4) = \left\{ \begin{array}{l} u_{6,4}+u_{4,5} \leq 1 \\ u_{6,4}+u_{4,7} \leq 1 \\ u_{5,4}+u_{4,5} \leq 1 \\ u_{5,4}+u_{4,7} \leq 1 \\ u_{3,4}+u_{4,5} \leq 1 \\ u_{3,4}+u_{4,7} \leq 2 \end{array} \right\}$

Figure 3.9: Example of a constraint node w_j (in this case j=4)

node $h_{in}*$ and the part occupying the constrained node will move simultaneously in the node $h_{out}*$.
In all other cases the transition of this specific part can not occur and the corresponding control input must be set to zero (for some constrained nodes no pair (h^*_{out}, h^*_{in}) is assigned and therefore no pair of transitions that can take place at the same time). This is the main reason why it was decided to treat these constraints separately from the others. There is no need to resolve a conflict between different parts, because a priori it is known whether a part is in a position to move or not. The idea proposed is to filter the output of the one-step-ahead predictor in order to reset the Lagrangian states of the parts that are moving in a node w_i, but are not allowed due to $\mathcal{P}(w_i)$. Conflicts caused by this property are always won by the part that is held in node w_i, so it is not necessary to make a choice, for example with a greedy policy as for the type of conflicts that are solved by the path following strategy.

The algorithm that deals with the correction of the result of the one-step-ahead prediction has been included in the Greedy Path Following algorithm. It

has been implemented as a subroutine, immediately after the next Lagrangian states are computed. Thus it has been defined as *Forbidden Transitions Filtering Strategy* (FTFS) and in this thesis we will refer to this subroutine simply as Algorithm 4.
It is assumed that to each node of the plant recognised as constrained node w_j, $j = 1, \dots \mathbb{N}_w$ the related pair of nodes $(h^*_{out_j}, h^*_{in_j})$ has already been assigned. If no transitions can occur at the same time for a specific node, the corresponding pair of nodes is denoted by two null values. The algorithm receives as input the one-step-ahead prediction $\hat{\boldsymbol{x}}_i(k+1)$ and gives as output its correction $\hat{\hat{\boldsymbol{x}}}_i(k+1)$.

Algorithm 4 *Forbidden Transitions Filtering Strategy.* After each time the one-step-ahead prediction of the state is computed in GPFS:

1. For each part $i = 1, \dots, N_p(k)$ in the plant compute from its Lagrangian state $\boldsymbol{x_i}(k)$ the node where it is located at the current time k and the node that it will reach in the prediction $\hat{\boldsymbol{x}}_i(k+1)$, respectively as $h_{ai} = \mathcal{S}(s_i(k))^{(1,p_i(k))}$ and as $h_{ni} = \mathcal{S}(s_i(k))^{(1,\hat{p}_i(k+1))}$.

2. For each part i:

2.a If the next node is not a constrained node $h_{ni} \notin \mathcal{W}$, or it will be the same of the previous one $h_{ni} = h_{ai}$ (i.e, the part is held in the same position) update normally the state of that part:

$$\hat{\hat{\boldsymbol{x}}}_i(k+1) = \hat{\boldsymbol{x}}_i(k+1)$$

Thus if one of the two assumptions is not verified then proceed to the next step, otherwise verify another part state prediction.

2.b The part will move in a constrained node $w_j = h_{ni}$ at the next step, thus check if another parts is actually occupying that node. In case no part is held by w_j update normally the state of that part and verify another part state prediction, otherwise proceed to the next step.

2.c Verify if the l-th part, occupying the node w_j, will move to $h_{nl} = h^*_{out_j}$ and if the i-th part is actually in the node $h_{ai} = h^*_{in_j}$. In that case update normally the state of that part and verify another part state prediction otherwise proceed to last step.

2.d The i-th part is not allowed to move and its state prediction must be corrected:

$$\hat{\hat{\boldsymbol{x}}}_i(k+1) = \begin{bmatrix} s_i(k) \\ p_i(k) \\ t_i(k+1) \end{bmatrix}$$

As can be noticed, no conflicts are solved by this algorithm. It only checks if each part is in the condition to advance along its path, otherwise its position in the sequence will be moved back to the current step. At the end of the prediction correction some conflicts may remain unsolved, furthermore other conflicts may also have been generated keeping some parts in position (because a part can not move into a node that remains occupied). Those conflicts will be solved by the next steps of the Greedy Path Following Strategy, in which the **Algorithm 4** has been included.

One problem arises from this translation of the additional constraints, because the pair of transitions that can occur simultaneously in a constrained node w_j always has priority over any other. This means that if a continuous flow of parts triggers this pair of transitions, all other transitions are always inhibited. In this way the control algorithm can not choose to leave the constrained node empty for an instant so that another part, coming from a node different from $h^*_{in_j}$, can enter in w_j. The priority of conflicting parts in a crossroads can be forced by modifying their sequences appropriately. Repetitions have been introduced in order to allow the controller to evaluate the option of keeping in position a part that would otherwise have precedence. This will be discussed in detail in the Chapter 4 on the creation of sequences.

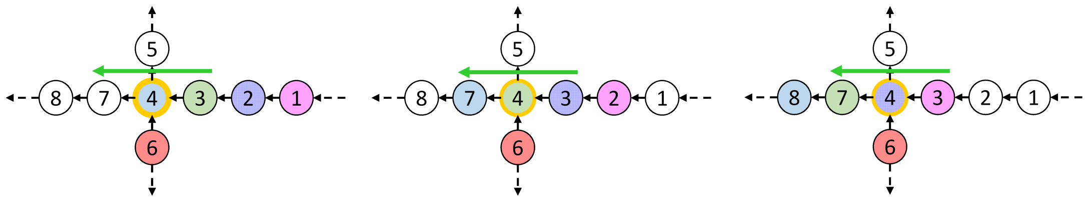


Figure 3.10: Priority problem due to Algorithm 4

In the figure 3.10 is shown the problem previously described. Each color indicates a different part. In this specific case the red one, located in node six, can not pass in w_4 because its transition is always inhibited by the parts that trigger the pair of transitions indicated by the green arrow.

Finally in figure 3.11 the scheme of the plant is reported to show which nodes have been considered as constrained nodes, coloured yellow, and the related couple of unconstrained transitions has been sketch as a green arrow (only for nodes that allow a couple of possible synchronous transitions).

Also the node 36, directly connected to the loading/unloading machine and representing the outside, could be considered a constrained node. In fact a part can not be loaded in the same step when another one is leaving the plant. But to take this into account, it was enough to modify the condition for which a new part is introduced in the system.

$$\begin{gathered} u_{36,32}(k) = 1, \text{ if } a(k) = 1 \ \wedge \nexists i : [\mathcal{S}(s_i(k))^{(1,\hat{p}_i(k+1))} = h_l \vee \mathcal{S}(s_i(k))^{(1,\hat{p}_i(k))} = h_l] \\ u_{36,32}(k) = 0, \text{ else.} \end{gathered} \tag{3.3}$$

The modification in the entry condition, with respect to the formulation introduced in [7] has been colored in red, and it adds that a part can be introduced into the system only if the load node is empty at the next step $k+1$ and even at the current step k.

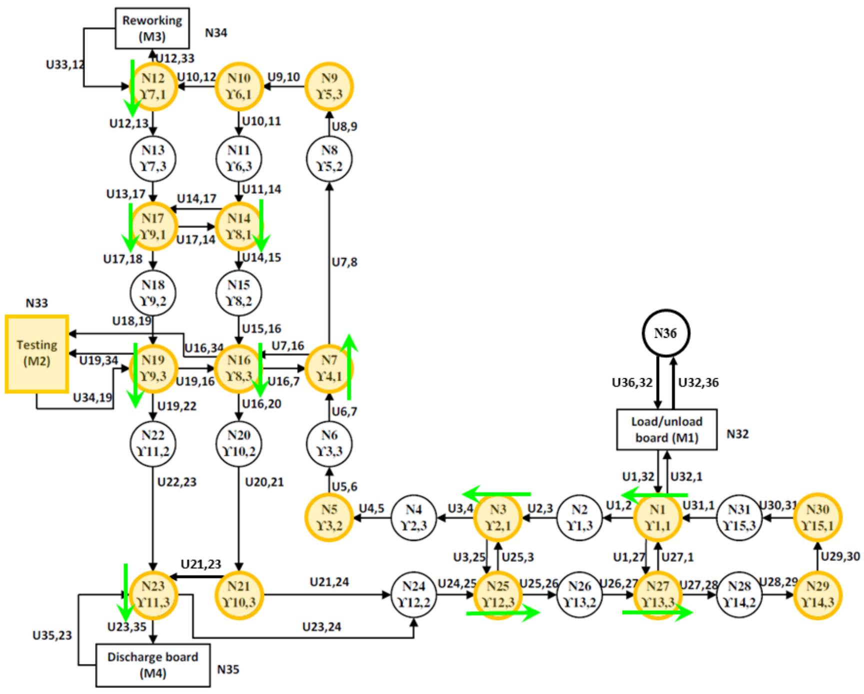


Figure 3.11: Constraints nodes in the CNR plant

3.3.2 Constraints on pairs of transitions with no node in common

The second category of additional constraints includes all the pairs of transitions that can not occur at the same time and that have no nodes in common. Therefore these constraints can not be grouped in a set that defines the property of a node. Usually the control inputs included in this type of constraints activate cross transitions, which if were triggered at the same time they would cause two pallets to collide with each other. Another case characterized by these constraints is related to a technical limitation of specific transport modules that can not perform two certain movements at the same time, even if they concern different nodes.

In particular for the real CNR pilot plant the set of this type of constraints is composed by six elements, reported below. It is possible to notice that for each pair of transitions the control actions are defined by four different nodes, and

$$\boxed{\begin{array}{l} u_{13,17}(k) + u_{12,33}(k) \leq 1 \\ u_{18,19}(k) + u_{16,34}(k) \leq 1 \\ u_{19,22}(k) + u_{16,34}(k) \leq 1 \\ u_{20,21}(k) + u_{23,24}(k) \leq 1 \\ u_{19,22}(k) + u_{35,23}(k) \leq 1 \\ u_{19,22}(k) + u_{21,23}(k) \leq 1 \end{array}}$$

Figure 3.12: Set of the pairs of constrained transitions

thus the general formulation of this kind of constraints can be defined as follows:

$$u_{a,b} + u_{c,d} \leq 1,\ a \neq b \neq c \neq d\ \wedge\ a, b, c, d \in \{1, \ldots, Nn\}$$

Unlike the previous case, these constraints generate a new type of conflicts between the parts. They are caused by the fact that only one of the two transitions can be triggered in a given instant k. Actually the path following algorithm only solves conflicts between two or more parts that would occupy the same node. The new constraints are translated into conflicts that can be resolved using the same greedy policy introduced previously. In order to do that the path following strategy must be improved by introducing a subroutine that has been denoted as Transitions Conflicts Solver algorithm, and in this thesis will be named Algorithm 5. To each j-th constraint of this type is assigned pair of couple of nodes $(a_j, b_j), (c_j, d_j)$ that indicates the couple of transitions τ_{1j}, τ_{2j} that are not allowed to occur at the same time, referring to the constraint j.

Algorithm 5 *Transitions Conflicts Solver.* For each j-th transition constraint:

1. Detect if there is a conflict between two parts i, l due to the constraints j.

1.a Check if there is a part i actually occupying the first node of transition τ_{1j} and if the next node of its path will be the second node of that transition:

$$a_j = \mathcal{S}(s_i(k))^{(1,p_i(k))} \ \wedge \ b_j = \mathcal{S}(s_i(k))^{(1,p_i(k+1))} \tag{3.4}$$

If the condition is verified, go to the next step. Otherwise the j-th constraint is fulfilled.

1.b Check if there is a different part j actually occupying the first node of transition τ_{2j} and if the next node along its path will be the second node of that transition:

$$c_j = \mathcal{S}(s_l(k))^{(1,p_l(k))} \ \wedge \ d_j = \mathcal{S}(s_l(k))^{(1,p_l(k+1))} \tag{3.5}$$

If the condition is verified the conflicting pair of parts (i, l) is identified, go to the next step. Otherwise the j-th constraint is fulfilled.

2. Compute the index i^* of the part with the highest priority:
`If` $r_i(k) < r_l(k)$ `then` $i^* = i$
`Elseif` $r_i(k) > r_l(k)$ `then` $i^* = l$
`Elseif` $r_i(k) = r_l(k)$ `then` $i^* = \arg\min_{y \in \{i,l\}} t_y(k)$.
where $r_i(k)$ is the number of remaining nodes for the generic part i to complete its sequence.

3. The one-step-ahead prediction state of the part i^* is kept the same while the state of the part $\underline{i}$ that has lost the conflict is corrected:

$$\hat{\boldsymbol{x}}_{\underline{i}}(k+1) = \begin{bmatrix} s_{\underline{i}}(k) \\ p_{\underline{i}}(k) \\ t_{\underline{i}}(k+1) \end{bmatrix}$$

The Algorithm 5 solves just conflicts on couple of transitions, it does not solve all the type of conflict, and at the end of its execution it can leave some conflicts unresolved or even generate others (holding a part in position could cause a conflict). Moreover the Algorithm 5 does not detect the conflicts on transitions checking directly the control actions because the vector on the control inputs will be computed later, at the last step of the path following algorithm in which it has been included.

3.3.3 New subroutines implemented in the Greedy Path Following Strategy

The two algorithms previously described (Algorithm 4, Algorithm 5) are both included in the Path following strategy in order to take into account also the additional constraints for generating the control action. Is important to notice that the Hierarchical Routing Control allows to solve the optimal control problem related to the path allocation without the issue of complying with rigid constraints. All the plant constraints have been converted into specific subroutines inside the low-level logic, which ensures that they are fulfilled when generating the control action. Following this approach every logical constraint can be translated into a specific algorithm and every temporal constraint can be described with a correct formulation of the sequences.

In the figure 3.13 has been schematized how the two subroutines have been implemented in the Greedy Path Following Strategy. The Forbidden Transition Filter (Algorithm 4) corrects the output of the One-step-ahead predictor. It ensures that in the prediction the properties of all constrained nodes are fulfilled holding in position the parts that can not move. The block named "Conflicts solving" groups all the steps of the GPFS that detect and solve the conflicts caused by two or more parts competing for a single node. When all the conflicts have been solved the predicted state is also modified by the Transitions Conflicts Solver (Algorithm 5), in order to solve conflicts on pairs of transitions due to the additional constraints described above. Then the GPFS algorithm searches if new nodes with conflicts have been generated by the Transitions Conflicts Solver subroutine, and if so, it solves them. Finally the one-step-ahead state

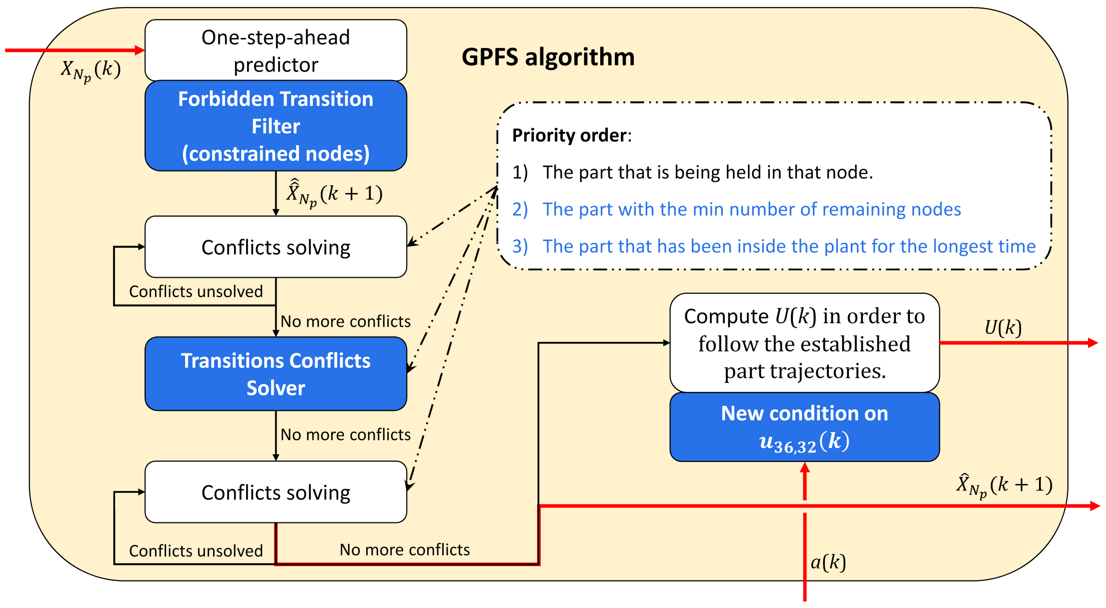

$X_{N_p}(k)$
One-step-ahead predictor
Forbidden Transition Filter (constrained nodes)
$\hat{\hat{X}}_{N_p}(k+1)$
Conflicts solving
Conflicts unsolved
No more conflicts
Transitions Conflicts Solver
No more conflicts
Conflicts solving
Conflicts unsolved
No more conflicts
GPFS algorithm
Priority order:
1) The part that is being held in that node.
2) The part with the min number of remaining nodes
3) The part that has been inside the plant for the longest time
Compute $U(k)$ in order to follow the established part trajectories.
New condition on $u_{36,32}(k)$
$U(k)$
$\hat{X}_{N_p}(k+1)$
$a(k)$

Figure 3.13: Scheme of the implementation of Algorithm 4 and Algorithm 5 in the Greedy Path Following Strategy

prediction is provided and the control inputs needed to reach it are computed (following the new entry condition to establish $u_{36,32}$). It is possible to notice that the Transitions Conflicts Solver subroutine shares the same greedy policy to compute the part with the highest priority, in order to be consistent with the choices made by the GPFS conflict solving.

At this point the model of the real plant is described almost correctly by the new algorithm. In fact as announced above, further constraints, such as temporal constraints due to the processing times of different machines, are directly described by the sequences that will be assigned to the parts. This problem is discussed in the next chapter together with the more general problem of sequence generation.

Chapter 4

Path allocation redundancy and sequence generation problem

In the previous chapter the approach implementation has been discussed referring to the low-level logic and the complete model of the plant has been obtained including the constraints on transitions in the Greedy Path Following Strategy. After that it is necessary to analyze the two main features of high-level logic (i.e., the Model Predictive Path Allocation): the generation of the sequences to be assigned and the methods applied to solve the FHOCP. The first one of these issues is analyzed in the following lines, while the latter will be exposed in the next chapter.

Recalling how the sequences have been formulated in the paper [7], the general expression of a sequence $\mathcal{S}(s)$ is:

$$\mathcal{S}(s) = \left\{ \begin{bmatrix} h_1 \\ g_1 \end{bmatrix}, \dots, \begin{bmatrix} h_p \\ g_p \end{bmatrix}, \dots, \begin{bmatrix} h_{N_s} \\ g_{N_s} \end{bmatrix} \right\} \tag{4.1}$$

where $s \in S$ is the index of the sequence. The length of a sequence corresponds to the number of nodes that constitute its trajectory, in other words the N_s number of positions along the sequence. The elements h_p and g_p, with $p = 1 \dots N_s$, are the node occupied by the part in the p-th sequence position and the related target to reach, respectively. In order to allocate a sequence to a part at the time step k, the sequence index $s(k)$ and a position along that sequence $p(k)$ have to be assigned to the Lagrangian state $\boldsymbol{x}(k)$ of that part:

$$\boldsymbol{x}(k) = \begin{bmatrix} s(k) \\ p(k) \\ t(k) \end{bmatrix}$$

The first problem that will be discussed has already been introduced in Chapter 2, dealing with the paths redundancy in evaluating which paths can be assigned. The sequence generation problem will be covered secondarily.

4.1 Paths redundancy in the set of compatible sequences

At each step the Model Predictive Path Allocation assigns a sequence to each part in order to avoid lockouts and in order to reach the best overall pallet handling, maximizing the throughput of the plant. In the ideal case the controller would have available all the possible sequences that each part could perform from the node in which it is located. In this way the high-level logic could evaluate all the trajectories combinations starting from the actual state of the system. The approach introduced presents a Move Blocking FOHCP formulation [14]. According to this strategy it is allowed to assign a sequence to a part only at the first step of the prediction horizon. This means that the sequences can not be combined during the simulation and must indicate the entire trajectory of the part from the current node to the final target. As an example the scheme of a generic plant graph is reported below, in which a part must reach node 7 from node 1. The sequences that indicates the possible evolution of the part from each node through the graph have been highlighted in three different colours and their indexes are denoted as s_1, s_2, s_3.

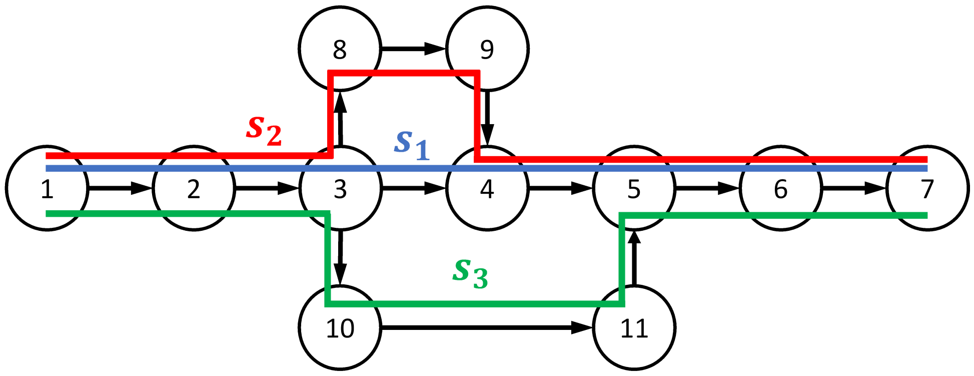


Figure 4.1: Paths redundancy example

The mathematical formulation of those sequences $\mathcal{S}(s_1)$, $\mathcal{S}(s_2)$, $\mathcal{S}(s_3)$ is reported below. It is possible to notice that the target is always equal to 7 for each position in each sequence, while the other elements describe the trajectory of the part as an ordinate list of nodes (a sequences could have many targets, the corresponding index changes each time one of them has been reached).

$$\mathcal{S}(s_1) = \left\{ \begin{bmatrix} 1 \\ 7 \end{bmatrix}, \begin{bmatrix} 2 \\ 7 \end{bmatrix}, \begin{bmatrix} 3 \\ 7 \end{bmatrix}, \begin{bmatrix} 4 \\ 7 \end{bmatrix}, \begin{bmatrix} 5 \\ 7 \end{bmatrix}, \begin{bmatrix} 6 \\ 7 \end{bmatrix}, \begin{bmatrix} 7 \\ 7 \end{bmatrix} \right\}$$

$$\mathcal{S}(s_2) = \left\{ \begin{bmatrix} 1 \\ 7 \end{bmatrix}, \begin{bmatrix} 2 \\ 7 \end{bmatrix}, \begin{bmatrix} 3 \\ 7 \end{bmatrix}, \begin{bmatrix} 8 \\ 7 \end{bmatrix}, \begin{bmatrix} 9 \\ 7 \end{bmatrix}, \begin{bmatrix} 4 \\ 7 \end{bmatrix}, \begin{bmatrix} 5 \\ 7 \end{bmatrix}, \begin{bmatrix} 6 \\ 7 \end{bmatrix}, \begin{bmatrix} 7 \\ 7 \end{bmatrix} \right\}$$

$$\mathcal{S}(s_3) = \left\{ \begin{bmatrix} 1 \\ 7 \end{bmatrix}, \begin{bmatrix} 2 \\ 7 \end{bmatrix}, \begin{bmatrix} 3 \\ 7 \end{bmatrix}, \begin{bmatrix} 10 \\ 7 \end{bmatrix}, \begin{bmatrix} 11 \\ 7 \end{bmatrix}, \begin{bmatrix} 5 \\ 7 \end{bmatrix}, \begin{bmatrix} 6 \\ 7 \end{bmatrix}, \begin{bmatrix} 7 \\ 7 \end{bmatrix} \right\}$$

The problem that derives from this formulation concerns the creation of the set $\mathcal{X}_i(k)$, that is the set of all compatible pairs (s, p) with the Lagrangian state $\boldsymbol{x_i}(k)$ of the i-th part at the step k. In fact, if some sequences coincide in the final section of their path, they would still be considered as different options to be evaluated, even if the choice is absolutely indifferent in that part of the trajectory. For example, in the case just reported the last three steps of each sequences are identical and the FHOCP solver will uselessly perform three identical simulations when the part will be in the node 5 or 6. This redundancy increases significantly the complexity of the optimization problem, in particular when there are many different parts in the plant and when the path to be taken through the plant are long and overlying.

The solution adopted is to add a third Boolean value $f_p \in \{1, 0\}$ to the vectors that constitute the elements of the sequence. Its function is to worn when it is needed to evaluate the sequence as a different option or when it can be ignored, because it is identical to another one. The new formulation adopted is the following:

$$\mathcal{S}(s) = \left\{ \begin{bmatrix} h_1 \\ g_1 \\ f_1 \end{bmatrix}, \dots, \begin{bmatrix} h_p \\ g_p \\ f_p \end{bmatrix}, \dots, \begin{bmatrix} h_{N_s} \\ g_{N_s} \\ f_{N_s} \end{bmatrix} \right\} \tag{4.2}$$

For the previous example it is possible to choose that only s_1 will be evaluated from the node 5 on, and the sequences will assume this form:

$$\mathcal{S}(s_1) = \left\{ \begin{bmatrix} 1 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 4 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 5 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 6 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 7 \\ 7 \\ 1 \end{bmatrix} \right\}$$

$$\mathcal{S}(s_2) = \left\{ \begin{bmatrix} 1 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 8 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 9 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 4 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 5 \\ 7 \\ 0 \end{bmatrix}, \begin{bmatrix} 6 \\ 7 \\ 0 \end{bmatrix}, \begin{bmatrix} 7 \\ 7 \\ 0 \end{bmatrix} \right\}$$

$$\mathcal{S}(s_3) = \left\{ \begin{bmatrix} 1 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 10 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 11 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 5 \\ 7 \\ 0 \end{bmatrix}, \begin{bmatrix} 6 \\ 7 \\ 0 \end{bmatrix}, \begin{bmatrix} 7 \\ 7 \\ 0 \end{bmatrix} \right\}$$

The conditions that define the set of compatible sequences $\mathcal{X}_i(k)$ (each compatible sequences is indicated by the pair (s, p) including the sequence index and the position along that sequence) must be modified in order to implement this improvement. Thus a new condition has been added, that allows to consider a compatible position p in a sequence only if $f_p = 1$. The formulation of $\mathcal{X}_i(k)$ introduced above in (2.10), is reported with the additional conditions colored in red:

$$\text{if } \mathcal{S}(s_i(k))^{(1,p_i(k))} \notin \mathcal{M}:$$

$$\mathcal{X}_i(k) = \left\{ \begin{array}{l} (s,p) \in S \times \mathbb{N}: \\ \mathcal{S}(s)^{(1,p)} = \mathcal{S}(s_i(k))^{(1,p_i(k))} \\ \wedge\, \mathcal{S}(s)^{(2,p)} = \mathcal{S}(s_i(k))^{(2,p_i(k))} \\ \wedge\, \mathcal{S}(s)^{(3,p)} = 1 \end{array} \right\} \tag{4.3a}$$

$$\text{else if } \mathcal{S}(s_i(k))^{(1,p_i(k))} \in \mathcal{M}:$$

$$\mathcal{X}_i(k) = \left\{ \begin{array}{l} (s,p) \in S \times \mathbb{N}: \\ \mathcal{S}(s)^{(1,p-j)} = \mathcal{S}(s_i(k))^{(1,p_i(k)-j)}, \\ j = 0, \dots, k - k_{\mathcal{S}(s_i(k))^{(1,p_i(k))}} \\ \wedge\, \mathcal{S}(s)^{(2,p)} = \mathcal{S}(s_i(k))^{(2,p_i(k))} \\ \wedge\, \mathcal{S}(s)^{(3,p)} = 1 \end{array} \right\} \tag{4.3b}$$

Where $\mathcal{S}(s_i)^{(3,p_i)}$ is the third entry in of the vector in position $p_i(k)$ of sequence $\mathcal{S}(s_i(k))$, corresponding to the Boolean value f_{pi} that has been introduced in the new sequence formulation. This improvement ensures to obtain the same plant behaviour, drastically reducing the computational cost needed to solve the FHOCP, because the redundant optimization variables $(\sigma_i, \pi_i) \in \mathcal{X}_i(k)$, $i = 1, \dots, N_p(k)$ have been removed (referring to the optimal control problem set in (2.11)).

A practical example is proposed with respect to the demonstration graph showed in the figure 4.1. It is considered the case with two parts in the graph, one in node 5 and one in node 6, both with node 7 as target. The controller should test three sequences for each part. Then a total of nine combinations is obtained. While, after introducing the new condition, the optimizer is not even activated, because the choice for each part is only one and therefore obligatory. The number of redundant combinations grows exponentially if we considered the graph of the real plant, with more parts inside and much longer trajectories. In conclusion this improvement has been essential to implement the Model Predictive Path Allocation with reasonable computational time.

4.2 Sequence generation problem

The previous contributions in the literature do not provide any guidelines on how the sequences should be created, they are simply assumed as already given, leaving this problem open. Considering the Move Blocking formulation of the optimization problem, the ideal case would be to provide the optimizer with all the possible paths that any part placed in any node could perform to reach the next target. This hypothesis is obviously impossible, because if loops are present in the graph an infinite number of sequences becomes necessary to represent all the combinations. Actually it is sufficient to define a loop only once within a sequence, so that when the part returns to the position at the beginning of the loop it can be reassigned the same sequence and repeat the loop, but unfortunately this happens only in the real behavior of the plant. In the prediction, when the part returns to the initial position of the loop has no possibility to go back in the sequence and repeat the path; it will be forced to perform the rest of the nodes indicated in the sequence assigned at the fist step of the horizon. Therefore the simulation of the model would not be able to reflect the true behavior of the system.

For these reasons, the Move Blocking approach that defines the control problem has been replaced by an extensive search in order to explores all branches of the search tree. In this way the ability of changing the sequences at every

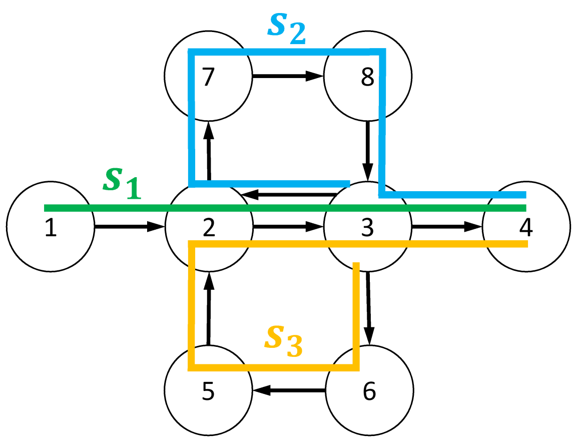


Figure 4.2: Example of sequence generation for a generic graph

step of the prediction horizon is correctly simulated, becoming consistent with the real plant behavior. In order to do that, the sets of compatible sequences are computed at each step, $\mathcal{X}_i(k+o)$, $i = 1, \ldots, Np(k+o)$, $o = 0, \ldots, N-1$, where $N \in \mathbb{N}$ is the length of the prediction horizon. Obviously the number of optimization variables increases significantly, because several optimization variables are generated at each step: $(\sigma_i(k+o), \pi_i(k+o)) \in \mathcal{X}_i(k+o)$, $o = 0, \ldots, N-1$. The complexity of the optimization problem grows exponentially as function of the length of the horizon and of the number of compatible states, that determines the number of branches for the corresponding time step. However the number of sequences to be assigned can be greatly reduced, and an infinite number of paths can be covered simply by combining a few sequences. In this way it is possible to evaluate all the possible parts trajectories during the simulation of the plant behavior.

A simple example is given to indicate the advantages of the search tree exploration in defining sequences. In figure 4.3 the parts must reach the node 4 from the node 1. With the Move Blocking approach the number of sequences needed to describes all the possible trajectories of a part starting from the node 1 would be infinite, in fact in the graph there are three loops which give the possibility to reach the node 4 with an infinite number of different trajectories. Thanks to the

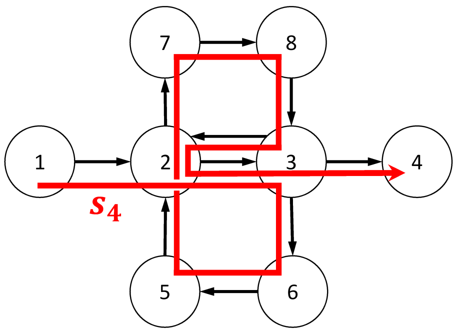


Figure 4.3: Example of sequences combinations

search tree exploration approach all the possible paths can be described by just three sequences:

$$\mathcal{S}(s_1) = \left\{ \begin{bmatrix} 1 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 4 \\ 4 \\ 1 \end{bmatrix} \right\}$$

$$\mathcal{S}(s_2) = \left\{ \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 7 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 8 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}, \begin{bmatrix} 4 \\ 4 \\ 0 \end{bmatrix} \right\}$$

$$\mathcal{S}(s_3) = \left\{ \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 6 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 5 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix}, \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}, \begin{bmatrix} 4 \\ 4 \\ 0 \end{bmatrix} \right\}$$

An example of a more complex sequences s_4 is proposed, obtained along the prediction horizon as a combination of the precomputed sequences s_1, s_2, s_3:

$$\mathcal{S}(s_4) = \left\{ \begin{bmatrix} 1 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 4 \\ 1 \end{bmatrix} \right\}^{\subset \mathcal{S}(s_1)} \cup \left\{ \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 6 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 5 \\ 4 \\ 1 \end{bmatrix} \right\}^{\subset \mathcal{S}(s_3)} \cup$$
$$\left\{ \begin{bmatrix} 2 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 7 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 8 \\ 4 \\ 1 \end{bmatrix} \right\}^{\subset \mathcal{S}(s_2)} \cup \left\{ \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 4 \\ 1 \end{bmatrix} \right\}^{\subset \mathcal{S}(s_2)} \cup \left\{ \begin{bmatrix} 2 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 4 \\ 4 \\ 1 \end{bmatrix} \right\}^{\subset \mathcal{S}(s_1)}$$

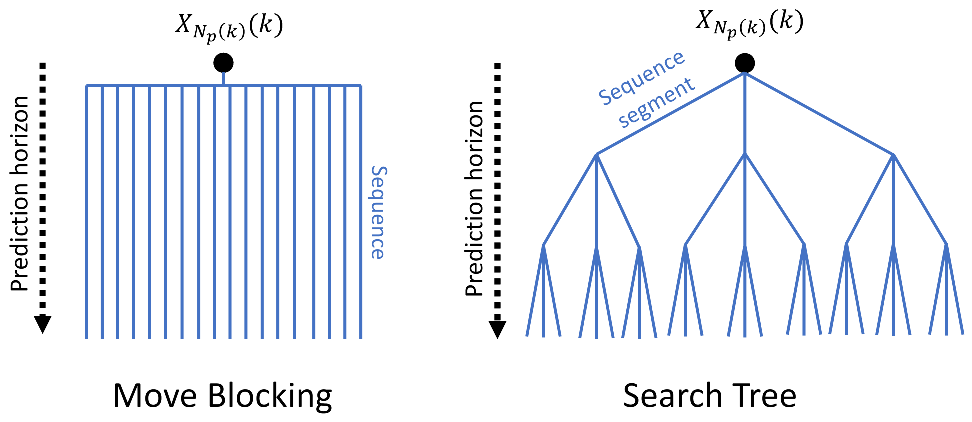


Figure 4.4: Move blocking and search tree schemes comparison

In order to define the minimum number of sequences it is necessary that they cover all the transitions of the graph. In this way it is possible to move the part freely through the plant performing any possible trajectory, combining segments of sequences. Following the Moving Blocking approach the sequences s_4, reported in the previous example, should have been precomputed and stored in the set S. The combination of the sequences, resulting from the exploration of the search tree, partially delegates the problem of sequence generation to the control algorithm. But the problem is not yet solved, because you have to take into account that along a sequence a part can have different targets and it is not possible to combine two sequences with different targets.

Along the prediction horizon a part motion does not necessarily coincide with the sequences assigned to it, because the motion of each part depends on the control actions computed by the Path Following Strategy algorithm, that takes into account the constraint and the limitation of the plant. But the algorithm can not change the trajectory of a part, it can only hold the part in position when its motion is not allowed.

Another important fact to be considered is the necessity of holding a part in a node. The temporal constraints due to the machine working time are represented in the sequence by repeating the specific machine node as many time as the steps needed to finish that machine process. Considering $L_m \geq 1$ as the time steps needed by the machine m to complete the task, the node m must be repeated at least L_m times along the sequences each time it is indicated. The second expression in the definition (4.3) determines how the set $\mathcal{X}_m(k)$ is generated for the machine node m. It guarantees that the compatible couple (s, p) take into account the steps already spent by the part inside the machine node, in this way the time constraint is ensured to be respected also when the part changes sequence during the machine process.

Having the possibility to hold a part in a node can be useful for several reasons. In the Move Blocking approach it is necessary to repeat a node along the sequence as many time as we want it to remains in that node during the prediction. In the real plant behaviour the part can be held in position for an infinite number of steps, it is sufficient that at each instant the previous position in the sequence is reassigned. At the first step of the simulation the part can decide how many steps to wait in its node, choosing a number of steps equal or less to how many times the node has been repeated along the sequences. While for the rest of the horizon it can wait in the nodes exactly how many times they have been repeated consequently (not considering when the part is held in position due to the path following strategy). For these reasons it is important to know in advance how long a part will have to wait in a specific node and that is not so easy to be estimated. In the search tree exploration approach this problem is solved; in fact it is sufficient to repeat a node exactly twice within a sequence to explore the possibility for a part of leaving the node or remaining there at each step of the horizon. The nodes repeated twice within the sequences have been defined as waiting nodes, and are chosen according to their strategic position. The functions of a waiting node are the following.

- **To avoid the lockout**: in many points in the plant graph two or more different trajectories intersect. Those nodes could cause a lockout if two parts block each other, such as at the entrance of machine nodes. In these cases it is necessary to choose a node adjacent to the crossing as waiting node. In this way we are able to anticipate the evaluation of the priority between part trajectories.

- **To force the part priority**: sometimes could be useful not to apply the greedy policy to solve conflicts between parts, perhaps to avoid plant congestion. To bypass the path following strategy it is enough to provide to the high-level logic the opportunity to evaluate the case where the part with the highest priority is waiting in position.

- **To inhibit the couple of synchronous transitions in a constrained node**: as anticipated, to a constrained node w_j could be associated a pair of transition that can simultaneously occur. A continuous flow of parts could trigger this pair of transitions, inhibiting all other transitions entering in w_j. To solve this problem it is sufficient to make the node $h^*_{in_j}$ a waiting node.

Because the search tree exploration is computationally costly, the number of sequences must be reduced as much as possible, and the presence of many waiting nodes along the sequences must be avoided. In the tests will be shown that just a long optimized sequence can be adopted to reach excellent performances, except when the part in the plant are to many. In conclusion the guidelines to generate the set of sequences are provided by these instructions:

1. Generate a main sequences s^* connecting the shortest path from the loading node to the first target node, from the first target node to the second one and so on. The last target must be the node representing the outside. In this way has been defined the ideal trajectory of a part through the plant, thus the fastest route possible that cover all the targets. To compute the shortest path between two targets is possible to use the Dijkstra's algorithm

[15], considering all the transition edges identically weighted.

2. Generate an additional number of sequences in order to represent the assignment of different targets when a part exits from a machine. In the case of the CNR plant, a part can be sent from the testing machine to the reworking machine, to the discharging machine or to the unloading cell. (one of this options is already included in the main sequence) The additional sequences are assigned by the external agent, determining the routine of a specific part. To generate those path is used the same method as for the main sequence.

3. A machine node m inside each sequences must be repeated as many times as the corresponding working time L_m in order to fulfilled the temporal constraints.

4. Other sequences can be generated to support a specific sequence already defined, in particular the main sequences, covering nodes that have not been included in that sequence and transition that could be useful to decongest the handling system.

5. Establish what nodes must be considered as waiting nodes, taking into account the three functions to be performed by these kind of nodes, previously indicated. The waiting nodes must be repeated exactly twice along the sequences.

6. Check the supporting sequences (generated at step **4.**) and the additional sequences(generated at step **2.**), if there are redundant segments with respect to other paths, set to zero the Boolean variable in all the third value of the elements of those segments.

The tests performed have underlined that the definition of the main sequence is fundamental to obtain good performances. In fact, in order to speed up the solution of the control problem, it has been decided not to introduce other supporting sequences and to manage the pallet traffic only by means of the waiting nodes. It is hoped that when the number of parts in the plant is not too high, the parts can follow the same path defined by the main sequence, calculated as the best one (except for the additional sequences that can change the trajectory of the part for sending it to a different target). Moreover it has been noticed that the number of waiting nodes establishes the number of branches in the search tree, i.e. the number of paths combinations to be simulated. Thus, in case no supporting sequence is adopted, the presence of waiting node determines the computational cost. The generation of the main sequence s^* can be formulated as a operations research problem. A Shortest Path Problem (SPP) with crossing avoidance [16] can be formulated: *Find the shortest path that connects the loading node to the unloading node, connecting in the right order all the targets that characterize the plant routine. At the same time minimize the number of waiting nodes needed by the paths (i.e, minimize the intersections between two different sections of the same path or between the additional sequences, which represent the routine flexibility, in particular those occurring in a constrained node.)*

The solution to this problem has not been discussed in this thesis, since it is outside of its purpose and part of future work. In particular, an interesting problem is to automate the sequence generation process, obtaining the right balance between computational cost and sequences variety. An on-line sequences generator might allow to develop an adaptive controller able to deal with line faults, for example the unexpected inaccessibility of some nodes.

4.2.1 Sequence generation in the real plant case

The following schemes show the result of the sequence generation guidelines. First, the main sequence is generated, connecting the paths between the different machines 4.5. The paths are chosen empirically as a compromise between sequence shortness and the reduction of the number of waiting nodes. In the second figure 4.6 the additional sequence is defined, representing the testing machine output that establishes the discharge machine as next target. The two other outputs of the testing machine (reworking machine or unload cell) are obtained reallocating the main sequences sections that present that specific target. Finally two different supporting sequences are proposed 4.7, 4.8, but their implementation can be useful only when the pallet traffic is congested and the controller need more nodes to avoid the lockout (e.g., by exploiting the creation of loops some nodes can be used just as buffer).

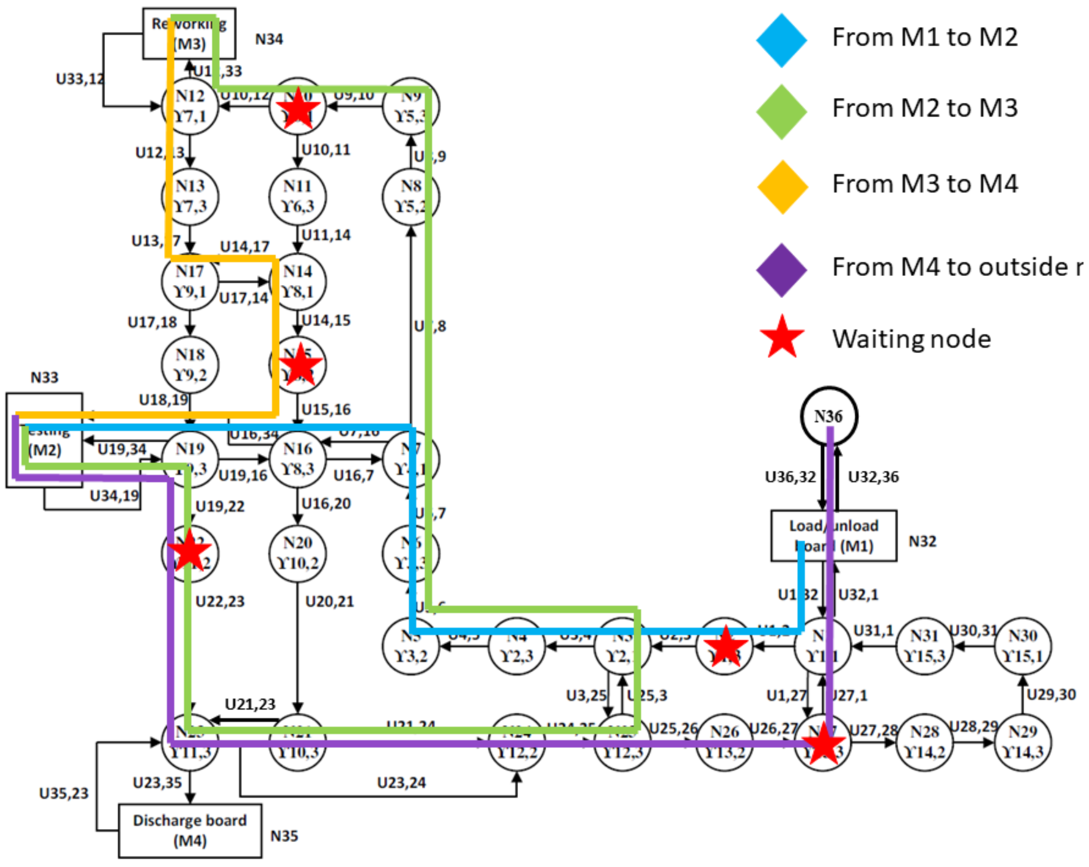


Figure 4.5: Main sequence generation example

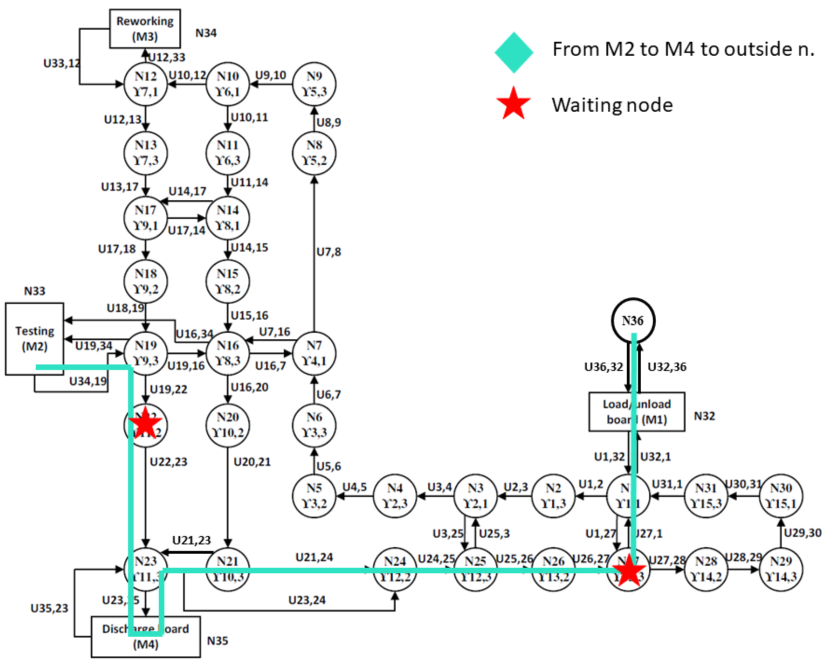


Figure 4.6: Additional sequence generation example

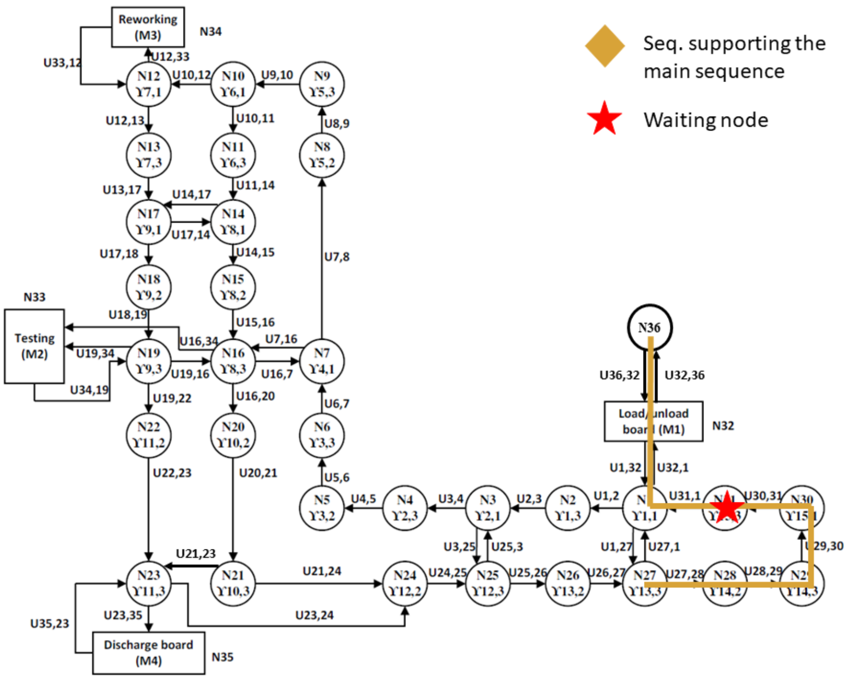


Figure 4.7: Supporting sequence generation example 1

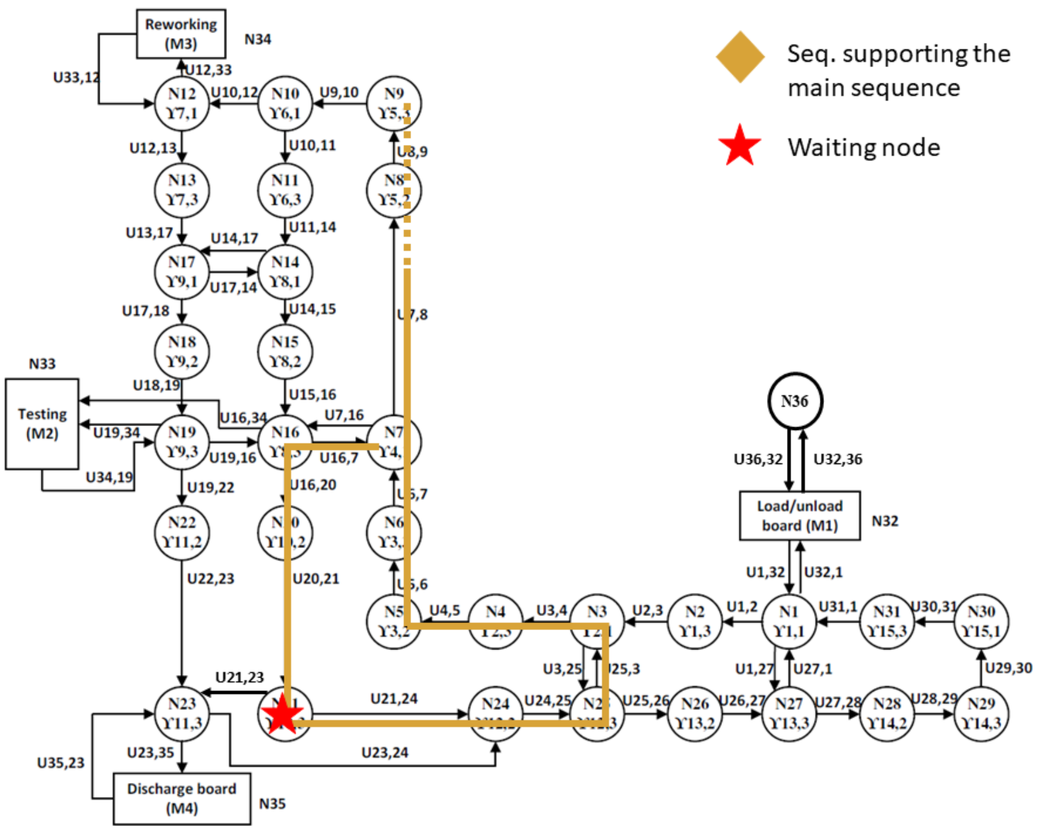


Figure 4.8: Supporting sequence generation example 2

Chapter 5

Solution methods for FHOCP in MPPA

The Finite Horizon Optimal Control Problem (FHOCP) is the core of the control strategy developed in this thesis. The purpose of all the algorithms previously proposed is to provide the optimizer with a model that allows at any time step to assign the best path combination by a *simulation based optimization.* The performance of the plant depends on the design of the stage cost, thus it is possible to adopt a multi-agent criterion that determines the plant behaviour. For example it can balance a trade off between conflicting goals, such as the throughput maximization and the energy saving. In the following sections the FHOCP is reformulated in order to implement a strategy based on the search tree exploration. In addition different optimal and sub-optimal solution methods are proposed. Finally a Move Blocking strategy is presented, assuming that is possible to choose when to bring a new part into the plant (i.e., assuming the exogenous signal $a(k)$ as controllable).

A key task of the controller is to avoid the lockout of the plant; to do this we must be able to identify when a choice will cause the lockout in the next steps and prevent it from happening. In the first section of this chapter a tool is provided to identify the formation of lockouts and a definition of the different types of lockouts is proposed.

5.1 Lockout detection

The ability to detect a lockout is very important from the computational cost point of view. The stage cost function defined in the FHOCP allows to avoid the choice of blocking paths, maximising the throughput or minimizing the time spent by the parts inside the plant. A non-blocking solution is always more convenient with respect to a blocking one. But if a lockout detector is implemented it is possible to interrupt and reject a sequences combination even during the simulation, or to interrupt the search without analyzing other scenarios when a non blocking paths combination has been reached. In the priority tree search it is fundamental to interrupt the search in a branch when a lockout occurs, otherwise the computation time would increase unnecessarily. The global lockout of the system (all the parts can no longer move in any direction) is caused by a local lockout, in which just two or more parts are blocking each other (unless they are not blocking fundamental nodes and other paths are still free to reach their targets). For this reason the movements of the parts must be analysed individually and it must be prevented that even a local lockout occurs.

In the present thesis two different types of lockouts are defined: the *local lockout* and the *theoretical lockout*. It is sufficient that even just one of the two happens to stop the simulation and discard the corresponding state causing the lockout.

5.1.1 Local lockout

A local lockout occurs when two different parts are reciprocally blocking each other. When the motion of a part along its sequences is inhibited, there is always another part with an higher priority blocking it. The problem arises when a part is indirectly inhibiting its own motion. Note that this case always occurs simultaneously for almost two different parts. A chain of conflicts could be created, starting and ending with the same part. In this case the conflicts force all the parts to be held in position and nobody will ever win. An example is proposed in figure 5.1, where the green part in node 6 can not move because it is blocked by the part in node 3, that is indirectly blocked by the green one by means of a chain of conflicts. This means that the green part is indirectly blocking itself, and will be no longer able to move. The same reasoning applies to part in node 3 with respect to the part in node 6.

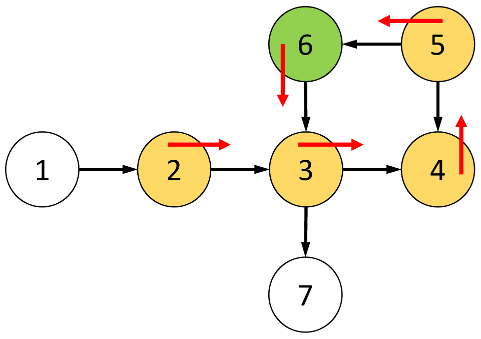


Figure 5.1: Local lockout example.

It is enough to find a single part that is indirectly blocking itself to detect a local lockout. And in order to do that a vector $\boldsymbol{Q}$ is defined at each step, where the elements q_i, $i = 1 \ldots, N_p(k)$ indicates the index of the part that is blocking the i-th part. If the part is free to move the corresponding element is set to zero. These values are computed each time a part is held in position by the path following algorithm (and by its subroutines, Algorithm 4 and Algorithm 5). It assigns the index of the part with the highest priority, that wins the conflict, to the value q_i related to a part i that loses the conflict. The condition that determines a lockout occurring is:

$$\exists(i,j) : \mathcal{Q}(i)^j = i \text{ and } i,j = 1 \ldots, N_p(k) \tag{5.1}$$

Where the operator $\mathcal{Q}(i)^1$ returns the index value $q_i \in \boldsymbol{Q}$, that denote the index of the part that is blocking i. The exponent j indicates how many times the operator is applied consequently (e.g, $\mathcal{Q}(i)^3 = \mathcal{Q}(\mathcal{Q}(\mathcal{Q}(i)))$). The condition (5.1) indicates that the local lockout occurs when the same part index is detected going backwards up the chain of conflicts.

In a particular case a lockout can be hidden to this condition: if a part is in a waiting node. Even if it can not move, because the adjacent node is occupied, the conflict is not reported and the part wait in position to avoid the unavoidable lockout. It can only progresses one position along the sequence and at the next step the previous one will be reassigned (remaining in the same node). To overcome this problem, it is sufficient to modify the vector of the blocking indexes $\boldsymbol{Q}$ in this way:

$$\text{If } \ \mathcal{S}(s_i)^{(1,p_i)} = \mathcal{S}(s_i)^{(1,p_i+1)} \ \wedge \ \exists \text{ a part } j \ : \ \mathcal{S}(s_j)^{(1,p_j)} = \mathcal{S}(s_i)^{(1,p_i+2)}$$
$$\text{then } \ q_i = j$$

The blocking index q_i assigned to a part i in a waiting node, corresponds to the index of the part occupying the node indicated in the next position along the sequence related to i.

5.1.2 Theoretical lockout

Theoretical lockout denotes a specific evolution of the state of the whole system characterized by the fact that state is equivalent to the previous one. Equivalent means that all the parts remain in the same node of the previous state and if there are parts in a machine node, those parts are not moving along their sequence (the node is the same and the position inside the sequence is not progressing, the machine has already completed the operation). The theoretical lockout does not necessarily correspond to a global lockout, because the part are not blocked, but all those parts that could move are waiting in a position. This type of lockout does not cause the irreversible blockage of the parts but indicates the fact that the chosen state has evolved in an equivalent one, thus the control action has been completely useless (it could repeat this condition indefinitely). The parts are not locked but are kept locked by the controller itself.

The detection of theoretical lockouts removes useless branches from the search tree and especially avoids that the controller keeps the parts in position to avoid incurring a unavoidable local lockout. In order to detect a theoretical lockout is needed to compute the one-step-ahead prediction of the state and compare it with the previous one. The three conditions required to identify a theoretical lockout are the following:

$$N_p(k) = N_p(k+1) = N_p$$

$$\mathcal{S}(s_i(k))^{(1,p_i(k))} = \mathcal{S}(s_i(k+1))^{(1,p_i(k+1))} \wedge \ \mathcal{S}(s_i(k))^{(1,p_i(k))} \notin \mathcal{M}, \ \forall i = 1 \ldots, N_p$$

$$\mathcal{K}(s_j(k), p_j(k)) < \mathcal{K}(s_j(k+1), p_j(k+1)), \forall j : \mathcal{S}(s_j(k))^{(1,p_j(k))} \in \mathcal{M}, j = 1 \ldots, N_p$$

Where the operator $\mathcal{K}(s_j, p_j)$ returns the number of positions, preceding the position p_j in the sequence s_j, with a node index identical to the actual node $\mathcal{S}(s_j)^{(1,p_j)} = m$ (i.e., it indicates for how many steps the part j has been inside the machine m up to the position p_j in the sequence s_j).

5.2 Brute force or extensive search

In order to easily generate the sequences set and to obtain a simulation consistent with the real plant behaviour, it has been decided to adopt a search tree exploration approach over the Move Blocking approach. The optimal solution can be found by means of a extensive exploration with a Brute Force method. The FHOCP proposed in (2.11) is thus reformulated:

$$\min_{(\sigma_i(k+o),\pi_i(k+o)),\ i=1,\ldots,N_p(k+o),\ o=0,\ldots,N-1} \sum_{o=0}^{N} \ell_{N_p(o|k)}\left(X_{N_p(o|k)}(o|k)\right) \tag{5.2a}$$

subject to

$$\boldsymbol{x}_i(o|k) = \left[\sigma_i(k+o), \pi_i(k+o), t_i(k+o)\right]^T,\ i = 1, \ldots, N_p(k+o), \tag{5.2b}$$

$$o = 0, \ldots, N \tag{5.2c}$$

$$\begin{gathered} X_{N_p(o|k)}(o|k) = \\ \left[\boldsymbol{x}_1(o|k)^T, \ldots, \boldsymbol{x}_{N_p(k)}(o|k)^T\right]^T, \end{gathered} \tag{5.2d}$$

$$o = 0, \ldots, N \tag{5.2e}$$

$$\begin{gathered} X_{N_p(o+1|k)}(o+1|k) = \\ f_{(N_p(o+1|k),N_p(o|k))}(X_{N_p(o|k)}(k), a(o|k)), \\ o = 0, \ldots, N-1 \end{gathered} \tag{5.2f}$$

$$(\sigma_i(k+o), \pi_i(k+o)) \in \mathcal{X}_i(k+o),\ i = 1, \ldots, N_p(k+o), \tag{5.2g}$$

$$o = 0, \ldots, N-1 \tag{5.2h}$$

Where the optimization variables $\sigma_i(k+o), \pi_i(k+o)$ are the sequence index and the position inside those sequence, respectively, to be assigned to each part at each step $k+o$ of the predicting horizon. Their number change at each step depending on the evolution of the number of parts in the plant $N_p(k+o)$ and

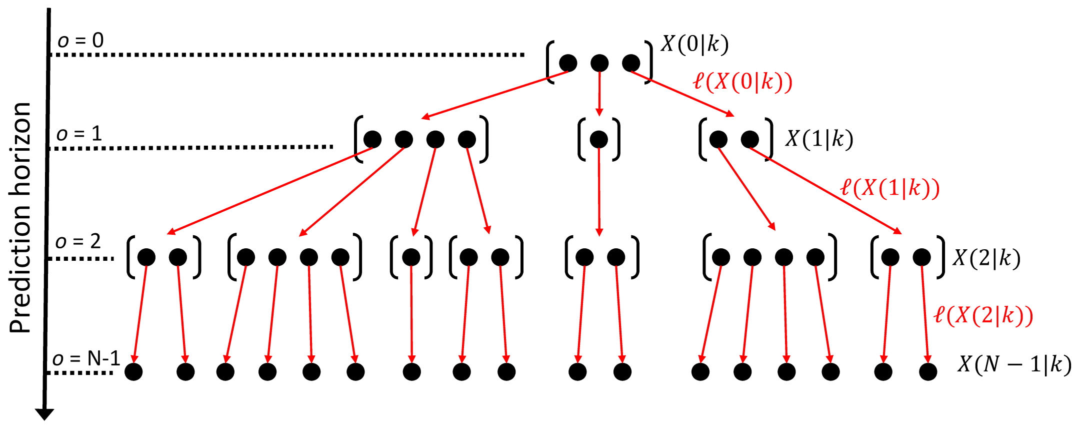


Figure 5.2: Search tree

they belong to the set of compatible states $\mathcal{X}_i(k+o)$ which can be assigned to that specific parts i at the prediction step $k+o$.

Because that problem is a non-linear integer optimization problem, a brute force approach has been adopted in order to test all the possible sequences combination. The optimal solution is find exploring extensively the search tree and choosing the branch associated to the minimum sum of edges weights. The stage cost function $\ell_{N_p(o|k)}\left(X_{N_p(o|k)}(o|k)\right)$ computes the weight related to the edge that connect the state $k+o$ to the next state. The search tree is generated simply simulating the plant model one-step behavior for each part state derived by the related compatible set $\mathcal{X}_i(k+o)$, represented in the figure 5.2 in brackets. The branches falling into a state that causes a local lockout or that repeat cyclically the same state evolution, because the simulation incurs an equivalent state (theoretical lockout), are immediately rejected to speed up the tree exploration. This is possible thanks to the ability to detect a lockout previously introduced.

The brute force approach can be implemented adopting the *Deep-First Search* [17]. Thus the main optimization problem can be divided into sub-problems as progressing along the horizon. Then the solution of the main problem can be easily obtained solving step by step each sub-problem, starting from the bottom of the horizon and coming backwards. The branches obtained from the underlying

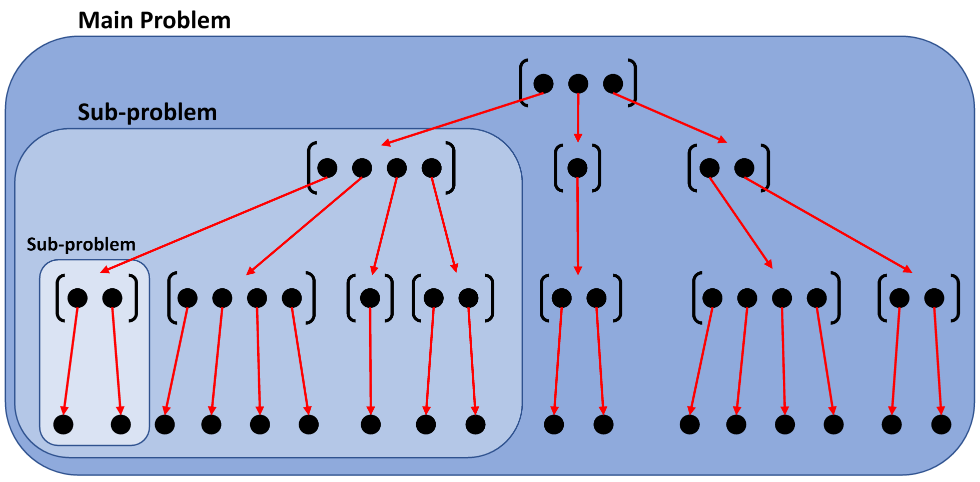


Figure 5.3: Search tree recursive solution via sub-problems

solutions are recursively connected to the upper edge related to the next sub-problem. In this way the problem to be solved at each step is trivial and consists only in the choice of the branch with the minimum cost in a finite range of possibilities. A graphical example of the recursive search tree exploration is proposed in following figures: in fig. 5.4 is represented the overall search tree scheme, in figure 5.5 the lowest sub-problems are solved, those at the end of the horizon, in fig. 5.6 the middle sub-problems are solved including the solutions of the previous sub-problems; finally in fig. 5.7 the branch with the minimum cost is found comparing the solutions obtained from the lower sub-problems.

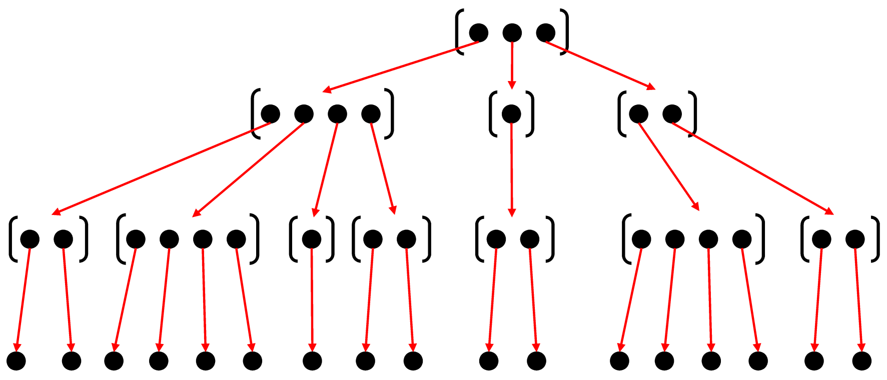


Figure 5.4: Search tree scheme

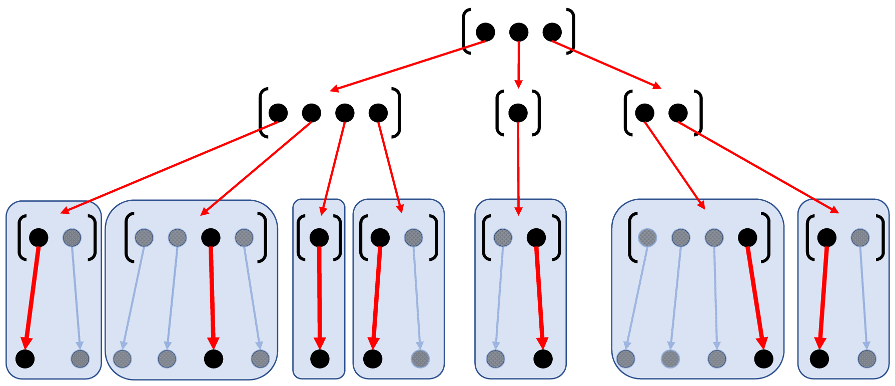


Figure 5.5: Solution of the sub-problems at the end of the horizon

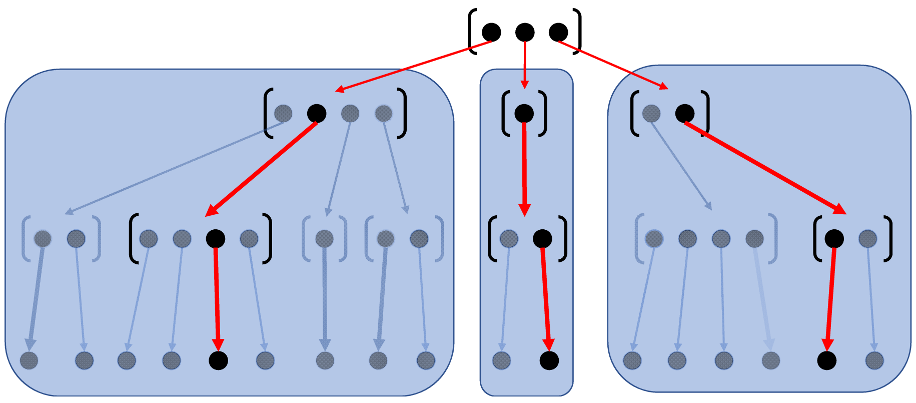


Figure 5.6: Backwards solution of the middle sub-problems

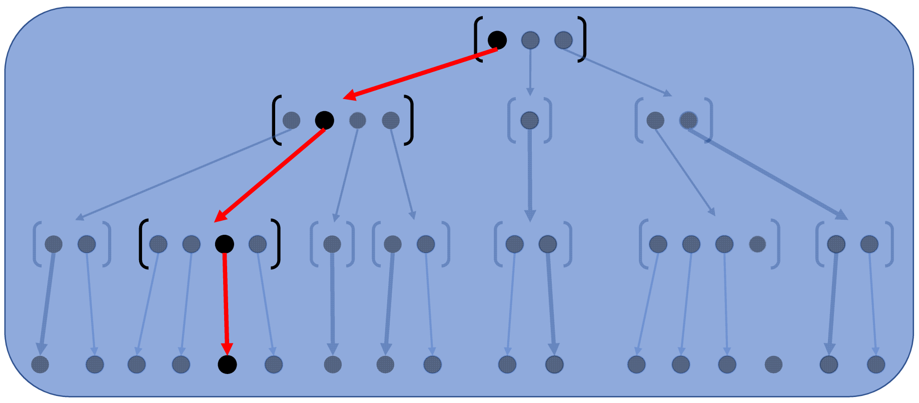


Figure 5.7: Solution of the main optimization problem

It is possible to notice that the solution of sub problems, that refers to the same step of the prediction horizon, can be parallelized and computed independently. The solution of the FHOCP via extensive search ensure to find the optimal solution but becomes unfeasible when there are too many pallets in the plant or too many sequences to evaluate. An approximate calculation of the number of one-step simulations can be performed considering a constant average value for the number of elements of the set of complementary states. Assuming a single sequence to be assigned, the size of the set of equivalent states depends on the number of waiting nodes actually occupied at that step. For example, considering approximately two occupied waiting nodes per step, there are four different state combinations to be tested (i.e, 2^n with n denoting the number of waiting nodes). The average number of simulation to be performed can be computed using this equation:

$$N_{sim} = \sum_{o=1}^{N} (2^{\bar{N}_w})^o$$

where $\bar{N}_w$ is the average number of waiting node occupied by a part at each time step $k+o$ of the prediction horizon. Considering always two constrained nodes to be occupied and a relatively short prediction horizon $N = 20$, the number of one-step simulation would be $1,466 \cdot 10^{12}$. Referring to the application developed in this thesis, the average computational time needed to simulate the system for one-step is $0,004s$. Thus the whole computational time needed to an extensive exploration of the search tree would approximately be $5.86 \cdot 10^9 s$. Therefore the brute force approach is not implementable and it has been replaced by sub-optimal tree exploration strategies in order to obtain a feasible computational time.

5.3 Intelligent Move Blocking

A sub-optimal alternative to extensive tree search is proposed. This method is developed calculating a feasible branch for each of the compatible states at the first step of the horizon. Thus the optimization variable are the pairs $(\sigma_i(k), \pi_i(k)$, $i = 1, \ldots, N_p(k)$, as for the move blocking approach, but with an improvement. Each time the simulation incurs a state causing a lockout, the simulation is returned in the previous state (one step backwards) and a new compatible state is chosen, in order to find a non blocking solution, if it exists. In conclusion the sequence assigned to each part can change only at the current time step or every time a lockout is detected along the prediction horizon. The Backtracking is used in the artificial intelligence field, because is an effective method to solve constraint satisfaction problems [18]. A graphical explanation of the lockout-avoidance backtracking is showed in figure 5.8 below, where the yellow cross indicates the detection of a lockout.

The Intelligent Move Blocking provides a sub-optimal solution to the problem, because the optimization depends exclusively on the choice between compatible states with respect to the actual state. The additional degrees of freedom help to reach the end of the horizon without incurring a lockout, making no contribution to the optimization.

The computational time is greatly reduced, from thousands of hours, see fig 5.9, to a few seconds, see fig. 5.10, but the problem is not completely solved. In fact a peak may occur when the plant is in a condition where it is difficult to avoid the lockout and the search tree must be explored almost completely before a feasible branch is found. For this reason other exploration methods have been developed and will be exposed in the next sections, obtaining better computational performances but at the cost of a worse sub-optimal solution.

The choice to adopt only one main sequence and to control the parts flow thanks to the waiting nodes (repeated twice along the sequence) adds a degree of

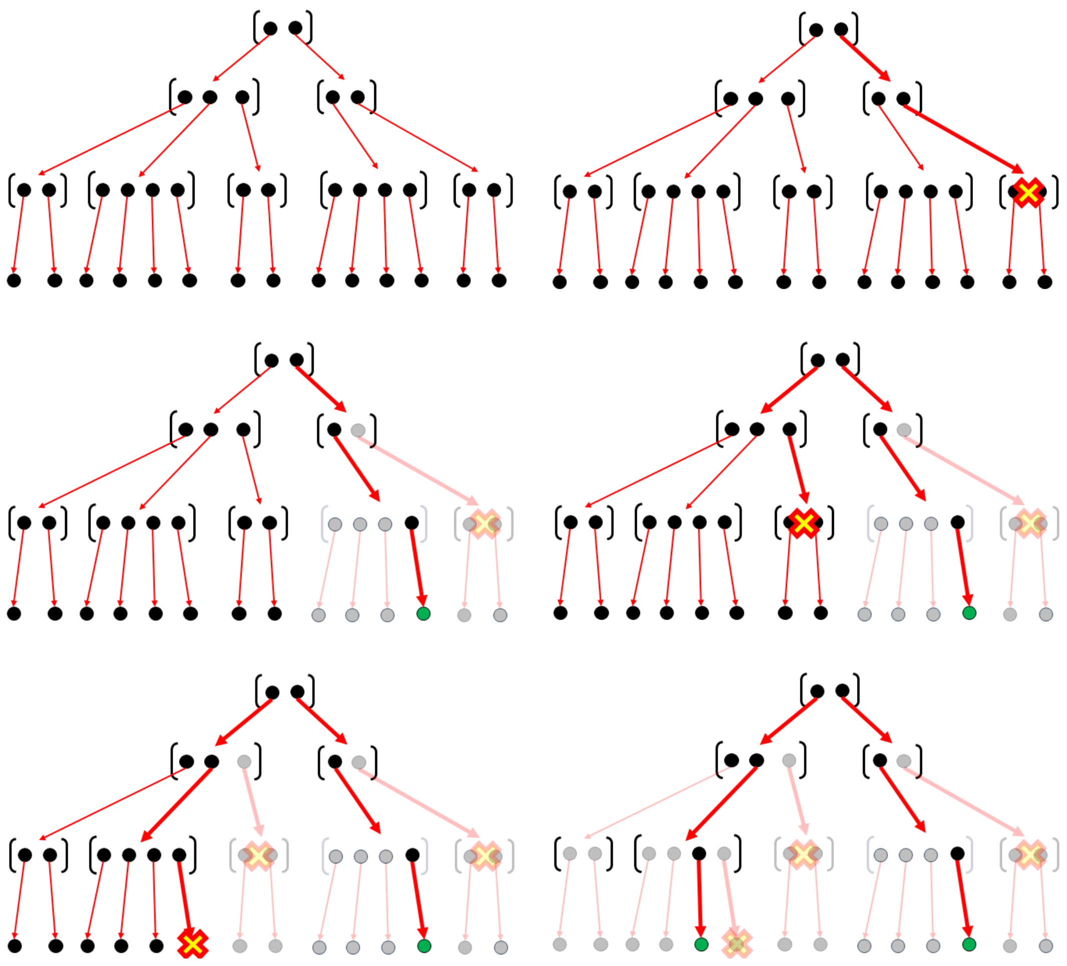


Figure 5.8: Intelligent Move Blocking strategy

freedom to the design of the Intelligent Move Blocking strategy. In fact, for that specific case the set of compatible states consists in the combination of choices regarding the motion or the waiting of the parts that are occupying a waiting node. Furthermore at each step of the prediction horizon it is possible to direct the evolution of the system, instead of letting the parts advance in open loop along the sequences assigned at the first step (as in the classic Move Blocking approach). In order to do that the set of all compatible states is computed at each step and the state chosen to continue the simulation depends on the order of choice (*Approaching Direction*) of the elements of the set. The Approaching Direction is an additional parameter that can be tuned by the designer in order to impose an aggressive or a soft control policy (i.e., it determines the ratio of how many part in a waiting node are push forward and how many parts are held in position at each step of the prediction horizon). The concept of approaching direction will be better explained in the following section.

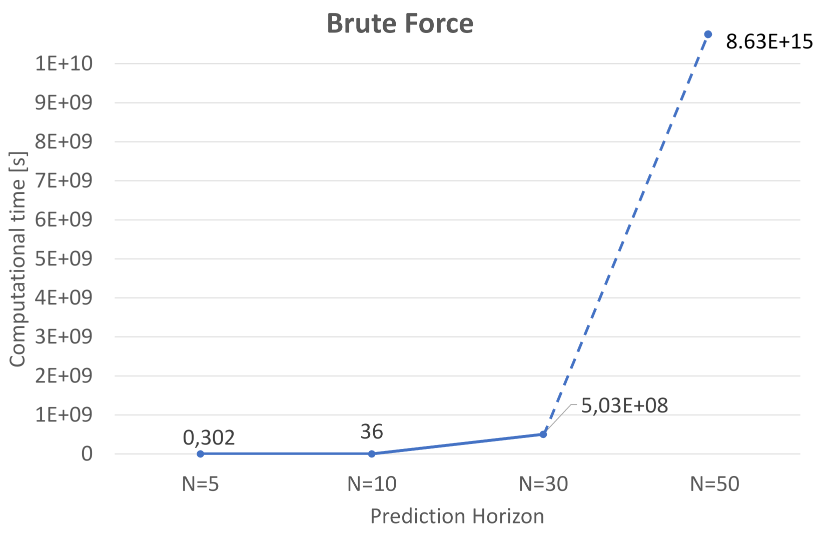


Figure 5.9: Brute force computational time

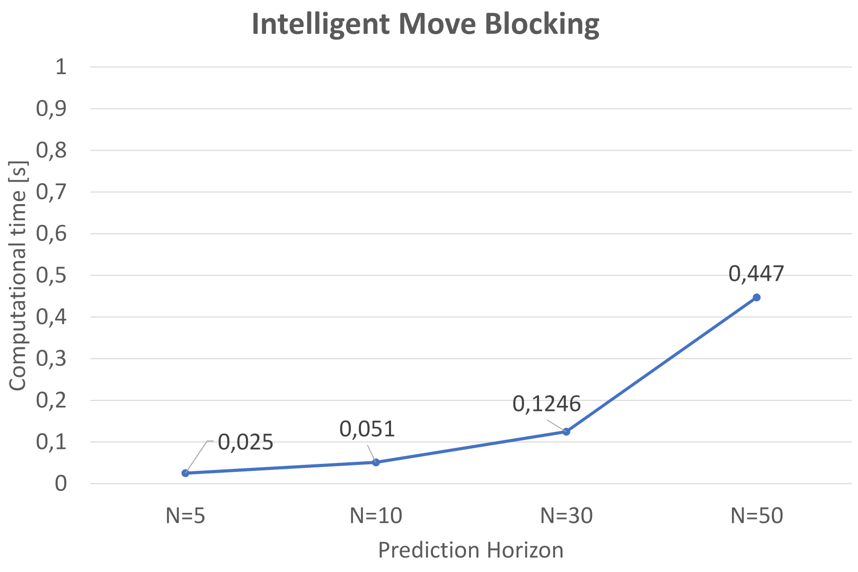
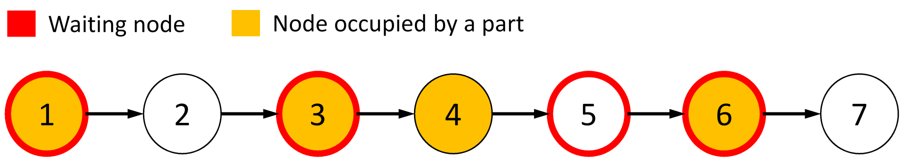


Figure 5.10: Intelligent Move Blocking computational time

5.3.1 Approaching direction

In the case a single main sequence with waiting nodes has been defined, the set of compatible states can be defined in an orderly manner. In that case it is possible to choose an *Approaching Direction* (AD) to test the elements of the set. In particular when the search tree will not be completely explored, it is crucial to determine what Lagrangian states combination must be tested first, because this will determine the quality of the sub-optimal solution. In the set of compatible states there are sequences combinations that try to move forward most of the part and others that hold in position most of the parts occupying a waiting node. This determines the *aggressiveness* of the control action and affects the result of the optimization. Thus the approaching direction consists in the testing order of the next edges in the search tree.

An example is proposed in order to define some approaching directions that will be implemented in the control algorithm in the next chapter. In the following figure 5.11 a simple graph is proposed with the precomputed main sequence $S(s_1)$ that defines four different waiting nodes $1, 3, 5, 6$ (underlined in red). At the actual time step in the below graph there are four parts and the nodes $1, 3, 4, 6$ are those occupied by those parts, coloured in orange.



$$\boldsymbol{S}(s_1)=\left\{\begin{bmatrix}1\\7\\1\end{bmatrix},\begin{bmatrix}1\\7\\1\end{bmatrix},\begin{bmatrix}2\\7\\1\end{bmatrix},\begin{bmatrix}3\\7\\1\end{bmatrix},\begin{bmatrix}3\\7\\1\end{bmatrix},\begin{bmatrix}4\\7\\1\end{bmatrix},\begin{bmatrix}5\\7\\1\end{bmatrix},\begin{bmatrix}5\\7\\1\end{bmatrix},\begin{bmatrix}6\\7\\1\end{bmatrix},\begin{bmatrix}6\\7\\1\end{bmatrix},\begin{bmatrix}7\\7\\1\end{bmatrix}\right\}$$

Figure 5.11: Example of a sequence defining the waiting nodes in a graph

The current four set of compatible sequences are: $\mathcal{X}_1(k) = \{(s_1, 1), (s_1, 2)\}$, $\mathcal{X}_2(k) = \{(s_1, 4), (s_1, 5)\}$, $\mathcal{X}_3(k) = \{(s_1, 6)\}$, $\mathcal{X}_4(k) = \{(s_1, 9), (s_1, 10)\}$ with part 1 in node 1, part 2 in node 3, part 3 in node 4, part 4 in node 6. It is possible to notice that all parts in a waiting node have two possibilities: to stay in position or to move to the next node, with respect to the first and the second pair respectively. The Lagrangian state $\boldsymbol{x}_i(k)$ related to the part i and to the first pair in $\boldsymbol{\mathcal{X}}_i(k)$ has been denote as $\boldsymbol{x}_i(k)[1]$, and that one related to the second pair has been denoted as $\boldsymbol{x}_i(k)[2]$, e.g. $\boldsymbol{x}_2(k)[1] = [s_1\ 4\ t_3]^T$ and $\boldsymbol{x}_2(k)[2] = [s_1\ 5\ t_3]^T$.

Therefore the set of compatible states related to the case just introduced is $X_{com}(k) = \{\boldsymbol{X}_1(k),\ \boldsymbol{X}_2(k),\ \boldsymbol{X}_3(k),\ \boldsymbol{X}_4(k),\ \boldsymbol{X}_5(k),\ \boldsymbol{X}_6(k),\ \boldsymbol{X}_7(k),\ \boldsymbol{X}_8(k)\}$. the number of elements is obtained as $2^{N_w(k)}$, where $N_w(k)$ is the number of waiting nodes actually occupied. The elements of the set $X_{com}(k)$ can be ordered from the more aggressive to the softer, following the combinatorial method reported below (referring to the specific example).

$$\begin{aligned}
X_1(k) &= [x_1(k)[1]^T, x_2(k)[1]^T, x_3(k)[1]^T, x_4(k)[1]^T]^T \\
X_2(k) &= [x_1(k)[2]^T, x_2(k)[1]^T, x_3(k)[1]^T, x_4(k)[1]^T]^T \\
X_3(k) &= [x_1(k)[1]^T, x_2(k)[2]^T, x_3(k)[1]^T, x_4(k)[1]^T]^T \\
X_4(k) &= [x_1(k)[2]^T, x_2(k)[2]^T, x_3(k)[1]^T, x_4(k)[1]^T]^T \\
X_5(k) &= [x_1(k)[1]^T, x_2(k)[1]^T, x_3(k)[1]^T, x_4(k)[2]^T]^T \\
X_6(k) &= [x_1(k)[2]^T, x_2(k)[1]^T, x_3(k)[1]^T, x_4(k)[2]^T]^T \\
X_7(k) &= [x_1(k)[1]^T, x_2(k)[2]^T, x_3(k)[1]^T, x_4(k)[2]^T]^T \\
X_8(k) &= [x_1(k)[2]^T, x_2(k)[2]^T, x_3(k)[1]^T, x_4(k)[2]^T]^T
\end{aligned}$$

It is possible to observe that $\boldsymbol{X}_1(k)$ is the softer state, where all the parts are held in position (the part 3 can not be held, because is not in a waiting node), while $\boldsymbol{X}_8(k)$ is the more aggressive state, where all the parts are pushed forward. The states $\boldsymbol{X}_7(k)$, $\boldsymbol{X}_6(k)$, $\boldsymbol{X}_4(k)$ are moderately aggressive, while the states $\boldsymbol{X}_1(k)$, $\boldsymbol{X}_2(k)$, $\boldsymbol{X}_5(k)$ are weakly aggressive. Not always the most aggressive states correspond to the best performances, in fact they can more easily cause a congestion of parts or even a lockout. In this thesis the following Approaching Directions (ADs) have been tested:

1. AD1: From the most aggressive to the softest
   $X_{eq}(k) = \{X_8(k), X_7(k), X_6(k), X_5(k), X_4(k), X_3(k), X_2(k), X_1(k)\}$.

2. AD2: From the softest to the most aggressive
   $X_{eq}(k) = \{X_1(k), X_2(k), X_3(k), X_4(k), X_5(k), X_6(k), X_7(k), X_8(k)\}$.

3. AD3: Inward alteration
   $X_{eq}(k) = \{X_1(k), X_8(k), X_2(k), X_7(k), X_3(k), X_6(k), X_4(k), X_5(k)\}$.

4. AD4: Outward alternation
   $X_{eq}(k) = \{X_5(k), X_4(k), X_6(k), X_3(k), X_7(k), X_2(k), X_8(k), X_1(k)\}$.

5. AD5: Alternation directed to the most aggressive
   $X_{eq}(k) = \{X_5(k), X_1(k), X_6(k), X_2(k), X_7(k), X_3(k), X_8(k), X_4(k)\}$.

The choice of the correct approaching direction can be very useful to avoid computational peak. In fact when the handling system is congested by many pallets, a too aggressive AD facilitates a lockout generation and it makes it very difficult to find a non-blocking solution able to reach the end of the horizon. In those situation the computational cost could rise tremendously, because the most prudent solutions, that would facilitate to solve the congestion, are tested last. Especially when the algorithm need to go back several steps to discard the choice that will cause the lockout. Another solution could be to interrupt the exploration of a branch that runs into a large number of lockouts during the prediction horizon, to avoid too much useless backtracking.

The approaching directions previously described have been implemented at the beginning of the tests as a constant control parameter, but they could be automatically changed at the beginning of the tree exploration, depending on the number of parts inside the plant, or they could even be changed at each step of the predicting horizon as function of the cardinality of the current compatible state set $X_{com}(k + o)$.

5.4 Non-optimal solution

If a feasible solution to the control problem has to be found as soon as possible, the optimization can be abandoned directly choosing the first solution computed during the exploration the search tree. The only condition to be satisfied in order to choose a feasible solution is that the plant lockout must be avoided for all the prediction horizon. The problem is identical to the one proposed in the Intelligent Move Blocking, but it is not necessary to choose the best solution among those found for each compatible state at the beginning of the horizon: it is chosen the first one.

Obviously, this method loses all the advantages of implementing optimal control, in fact the only task of the controller becomes to avoid the lockout, it can no longer affect the throughput or improve other performance of the system. Nevertheless the progression of the parts along the assigned routine is guaranteed. This is due to the fact that the sequences assigned to the parts contains the correct routine to be performed and if the sequences have been correctly generated, they partially solves the optimization problem, representing the most convenient paths. The control guarantees the advancement of the parts along the sequences, because the theoretical lockout has been included in the lockout detection, according to which a control action that keeps the system in a state equivalent to the previous one can not be chosen.

As for the Intelligent Move Blocking, the solution computation along the search tree can be directed by choosing properly the Approaching direction. That allows to test first the solutions that theoretically should improve the performances (a soft approaching direction could uselessly keep waiting parts in position). For the above reasons, even if the system performances are no longer controllable by the non-optimal controller, the quality of the solution is partially preserved; in particular when there are only a few pallets in the plant the solution is identical to the optimize one. Using this control method is very helpful in case the available computing power is extremely limited or the complexity of the search tree is really high, because the parts are in an unfortunate configuration where many different compatible sequences are detected.

The second case has been taken into account to develop a switching control strategy, able to adapt the solution search method to different scenarios in order to avoid computational peaks. A Intelligent Move Blocking strategy has been implemented with the ability to interrupt the optimization and take the first solution available. A switching criterion has been adopted to activate or deactivate the optimization with respect to the complexity of the search tree. The computational time in an Intelligent Move Blocking approach depends on the number of elements included in the compatible states set at the first step of horizon. In fact for each of those states it is required to find (if possible) a non blocking solution to be compared by the optimization process (figure 5.12). Therefore a maximum value of the cardinality of $X_{com}(0|k)$ has been established above which it is sufficient to compute a single non-optimal solution to the control problem.

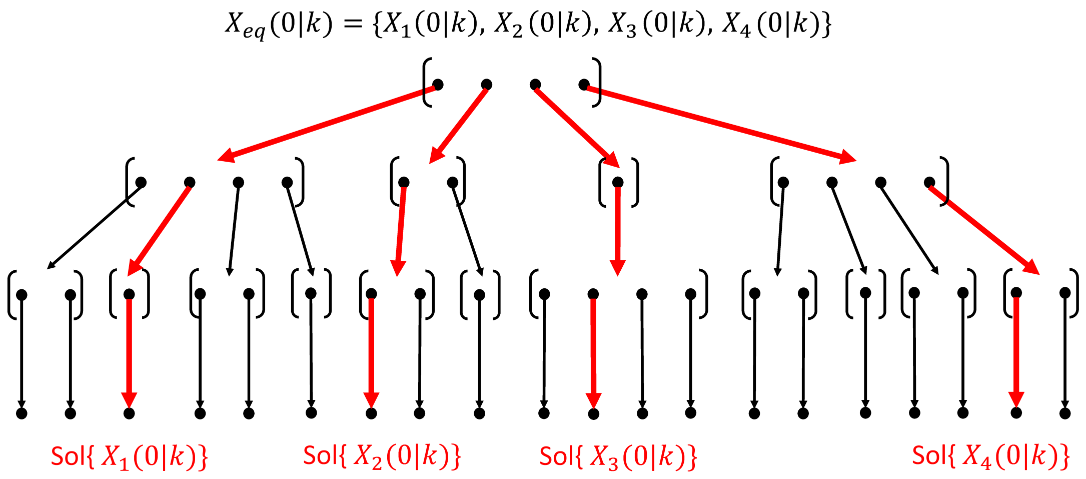


Figure 5.12: Number of solutions compared in a IMB problem

The parameter triggering the non-optimal solution strategy is denoted with ξ and is compared with $card(X_{com}(0|k))$ at the first step of the Model Predictive Path allocation algorithm (after computing of the sets $\mathcal{X}_i(0|k), i = 1\ldots, N_p(0|k))$. In conclusion the switching criterion between the two control strategy can be defined as follows:

$$\text{Control strategy} = \begin{cases} card(X_{com}(0|k)) \leq \xi & \rightarrow \text{ Intelligent Move Blocking} \\ card(X_{com}(0|k)) > \xi & \rightarrow \text{ Non-optimal solution} \end{cases} \tag{5.4}$$

Another switching criterion can be the based on the number of one-step simulations performed. After a certain number of simulations the control algorithm can interrupt the search tree exploration and choose between the feasible solutions already found or, in case no solution has been found, it can switch to a non-optimal solution approach. This strategy has not been developed in this thesis, because less interesting for the controller design.

The partial optimizing control strategy formulated in 5.4 has been defined as IMB(ξ): Intelligent Move Blocking with switching criterion based on ξ. Obviously the case IMB(0) corresponds to the non-optimal solution approach. Many tests have been performed; the results, that will be reported in the next chapter, have shown how throughput and computational cost depend on the value of ξ. In fact the more ξ decreases the more the throughput and the computational time decrease. But, in case an aggressive approaching direction has been adopted, with few parts in the plant the performances remains almost constant, while the computational cost decreases. It can be established that if the plant is not heavily congested, the optimisation is no longer relevant because the controller has only to push forward the parts along the main sequence and their trajectories do not conflict with each other.

5.5 Intelligent Move Blocking with sub-horizons

In order to improve the optimization result, the search tree has to be explored more extensively. The Move Blocking approach ignores most of the possible solutions, even with an lockout-avoidance backtracking. Moreover, when a soft approaching direction is adopted it would be very useful to compare a greater number of feasible scenarios to compensate the weakness of the solutions found. The strategy that has been developed for this purpose consists in a recursive nesting of Intelligent Move Blocking problems, in which the prediction horizon is divided in many sub-horizons. At the end of each sub-horizon a new IMB problem is set, starting from the current state. In this way the number of optimization variables increases at each sub-problem. It can be interpreted as the prediction of the plant behaviour branches out at the limit at each sub-horizon, thus developing all current alternative scenarios. The sub-horizons length is chosen depending on the desired level of detail adopted for the tree exploration. Of course, the higher the number of intervals in which the horizon is divided the more computational cost will increase. A graphical explanation is reported in the following figures, where the horizon is divided in two sub-horizons. Firstly the sub-problems are solved and their results is connected to the previous branches in order to solve the main problem. (This strategy can be implemented also with IMB(ξ) problems nesting)

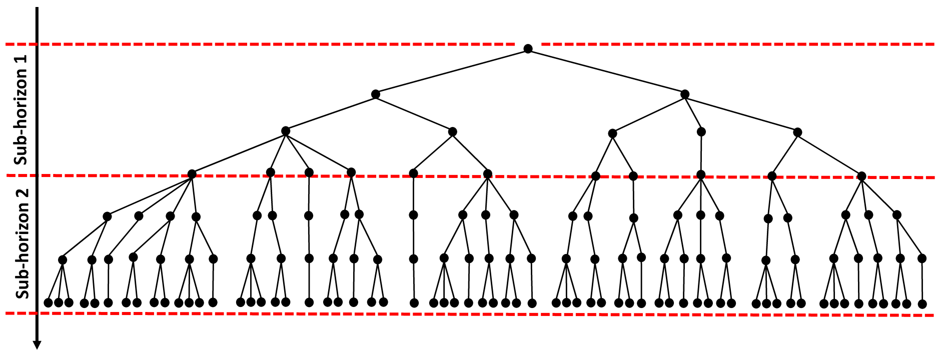


Figure 5.13: Search tree sub-horizons division

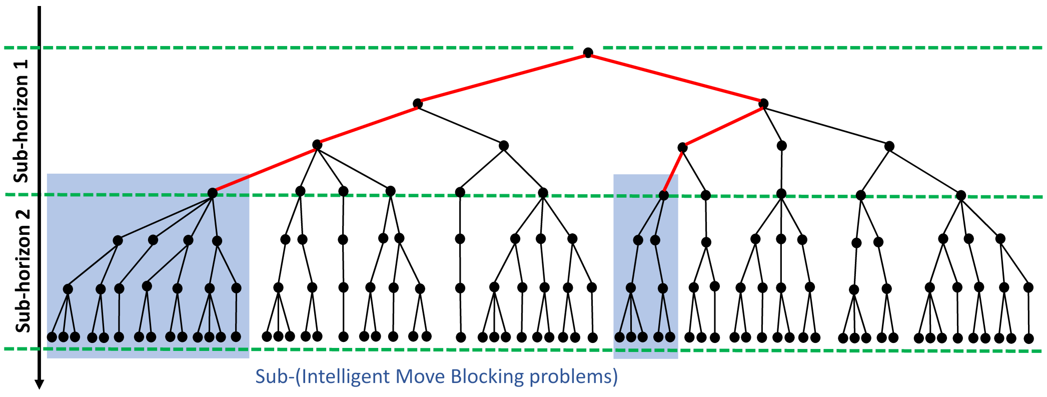


Figure 5.14: Lower sub-horizon IMB problems

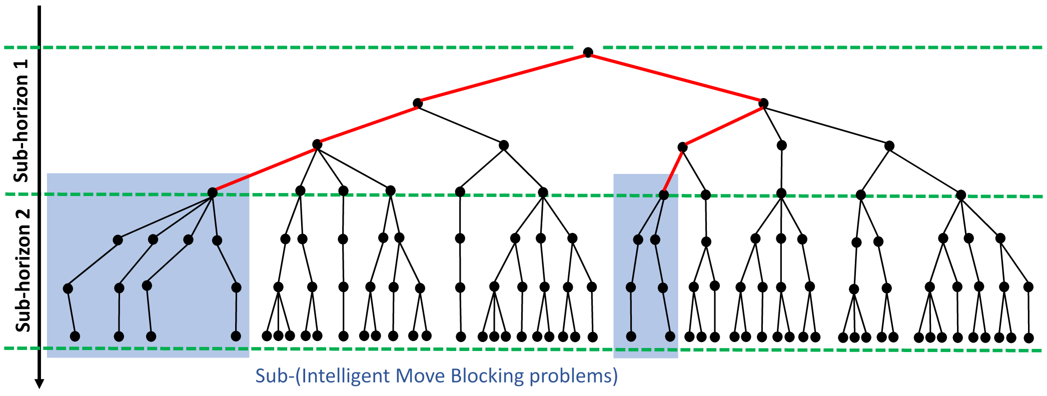


Figure 5.15: Solutions to be compared in the Sub-IMB problems

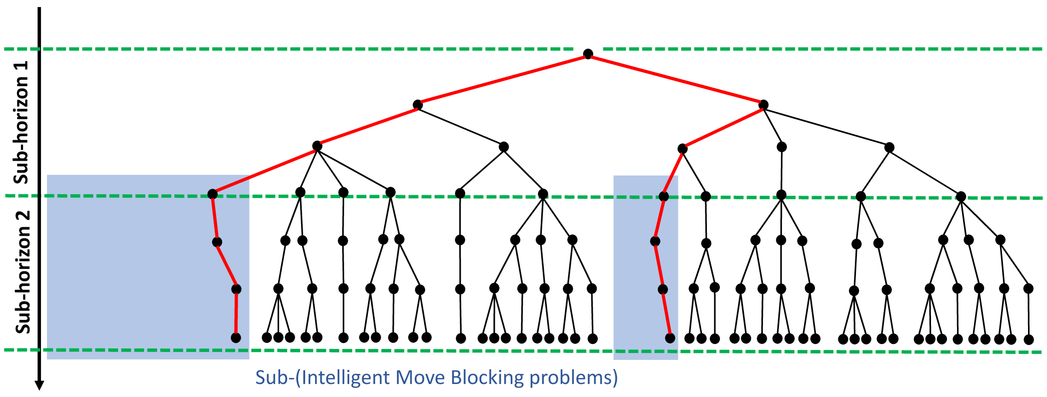


Figure 5.16: Main IMB: solutions to be compared

5.6 Move blocking with controllable exogenous input and emptying condition

The last control strategy that has been developed consists in a classical Move Blocking optimal control problem but with some improvements in order to make it effective. As already mentioned, the main problem of the MB strategy is due to the sequence generation. In fact it would be necessary an infinite number of path to represent each possible part trajectory from each node in the plant (i.e for each plant parts distribution). Moreover to a new part entering in the plant during the prediction horizon can not be assigned a different path, thus can not be controlled. While in the Intelligent Move Blocking variant, the system evolution is directed at each time step thanks to the approaching direction of the compatible state sets. Another disadvantages is that if the simulation from an initial compatible state incurs in a lockout, the only thing that can be done is to interrupt that simulation, because the state evolution can not be corrected to avoid the lockout. Some assumptions can be introduced to simplify the control problem and to make the Move Blocking strategy implementable:

- The exogenous signal $a(k)$, indicating that a new part must be loaded into the plant, is considered controllable. It is not included in the control problem as an additional optimization variable, but its value is chosen in order to avoid the lockout. In case a new part would enter the plant and the Model Predictive Path Allocation could not find a feasible solution to the next control problem, the signal $a(k)$ is switch off and the part is not loaded. That occurs because the set of precomputed sequences stores just few possible assignable path or because that maximum manageable number of parts has been reached

- The prediction horizon is variable, depending on the emptying plant condition. The system behaviour is simulated from the actual state until all the parts have been unloaded, guaranteeing that the plant is able to process all parts without incurring a lockout. To this purpose no new parts are loaded during the next steps of the simulation. Moreover the optimal control problem cost function can be substitute by the length of the prediction horizon. In fact the simulation in which the plant is emptied in the shortest time, corresponds to the optimal initial state.

The model predictive path allocation is modified in order to take into account these assumptions and the strategy adopted to solve the control problem can be summarized in few steps:

1. Set $a(k) = 1$ and compute the one-step-ahead prediction of the state $\hat{\boldsymbol{X}}(k+1)$ from the actual state $\boldsymbol{X}(k)$ by means of the plant model that include the Greedy Path Allocation Strategy.
Then check the provided vector of control actions if a new part would be loaded into the plant at $k+1$ (i.e., $u(k)_{36,32} = 1$).

2. If no part will be loaded, compute the optimal state $\boldsymbol{X}^*(k)$ compatible with the actual state $\boldsymbol{X}(k)$ applying the Model Predictive Path Allocation algorithm with a move blocking approach.
Then compute the control action by means of the plant model simulation and update the plant state.
If no part will be loaded go to the next step.

3. In case a new part is loaded compute the optimal state $\boldsymbol{X}^*(k+1)$ compatible with the state obtained from the one-step-ahead prediction $\hat{\boldsymbol{X}}(k+1)$ applying a move blocking approach. If no solution is possible set $a(k) = 0$ and go to step 2. otherwise apply the control input computed at the step 1. and update the plant state with the optimal state $\boldsymbol{X}^*(k+1)$ (in this way the parts will be routed along the optimized paths when the algorithm is restarted from step 1.).

This approach has reached excellent performances in terms of computational cost and of throughput. However compared to previous approaches it has less restrictive conditions. In fact the other approaches can not control the loading signal and they have to manage a new part even when it becomes very complicated to avoid the lockout. Besides the computational cost of the previous approaches is not necessarily higher, even if the number of parts in the plant remains almost constant along the prediction horizon and the number of optimization variables must be higher to avoid the lockout. In fact this last strategy has much longer prediction horizon length due to the emptying condition, increasing significantly the computational cost.

In conclusion the Move Blocking strategy can not be compared with the previous approaches, because operates in special conditions. It do not include the lockout-avoidance backtracking and the possibility to choose the approaching direction to affect the system evolution along the search tree. Thus when all the solutions incur a lockout, the control logicn can just renounces to manage an extra part.

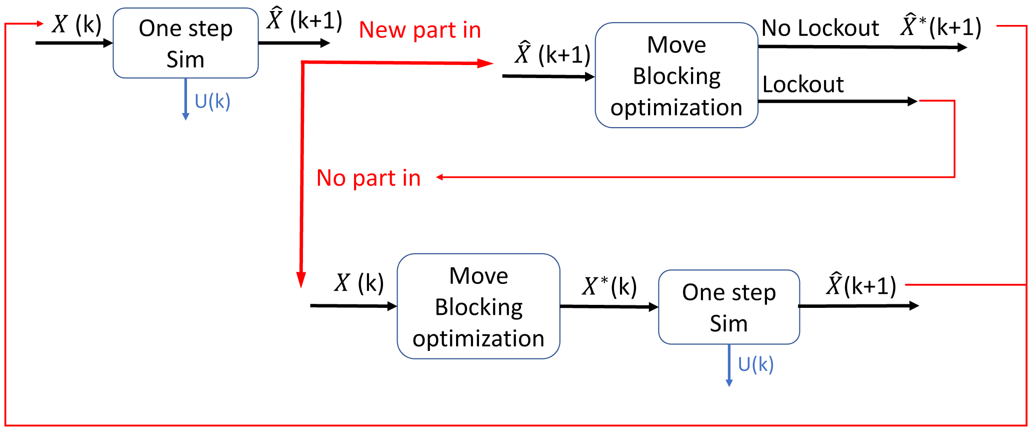


Figure 5.17: Move blocking with controllable exogenous input and emptying condition, control logic scheme

Chapter 6

Test results and methods comparison

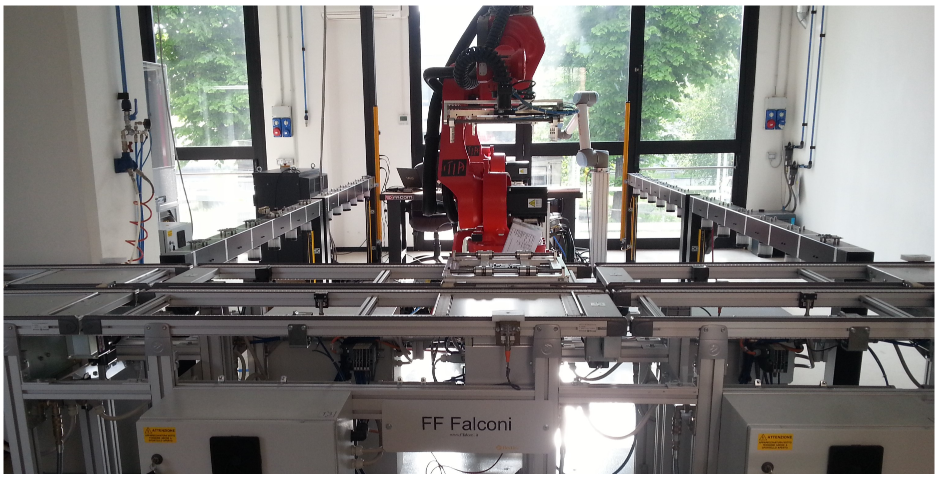


Figure 6.1: Photo of the CNR pilot plant

In this chapter are presented the results of the tests obtained from all the simulations of the control algorithm, to verify the validity of the proposed HRC approach and to compare all the possible methods for solving the optimization problem. A laptop with 16GB RAM and an Intel Core i7-7700HQ at 2.8 GHz has been used and the simulations have been implemented via MATLAB.

The routine characterizing the target of a specific part has been defined as shown in figure 6.2 by setting the probability of the possible testing machine results (M2). The first time a board is processed by the testing machine the probability associated to the new target are:

- $P_{M4} = 0.4$ the board can not be repaired and is discharged;
- $P_{M1} = 0.2$ the board works properly and is unloaded from the plant;
- $P_{M3} = 0.4$ the board can be repaired and it is sent to the reworking machine.

If a part is sent to the reworking machine, then it will be processed by the testing machine a second time. In that case the board can no longer be sent to the reworking machine, thus the outputs probabilities of the two remaining options are:

- $P_{M4} = 0.4$ (the board is definitely broken and is discharged);
- $P_{M1} = 0.6$ (the part works properly and is unloaded from the plant).

For each machine a fixed working time L_m as been chosen. This is a simplification, because the specific operations performed by the testing machine or by the reworking machine may vary depending on the status of the processed board. Specifically, the working times (denoted by number of steps) has been set as following:

L_{m1}	L_{m2}	L_{m3}	L_{m4}
1	5	4	3

Table 6.1: Table of machines working time

The main purpose of the tests is to verify in different scenarios which method is more efficient with respect to performance and computational time. The two parameters that will most affect the results are the *maximum number of parts* in the plant and the *length of the prediction horizon*. It has been considered that

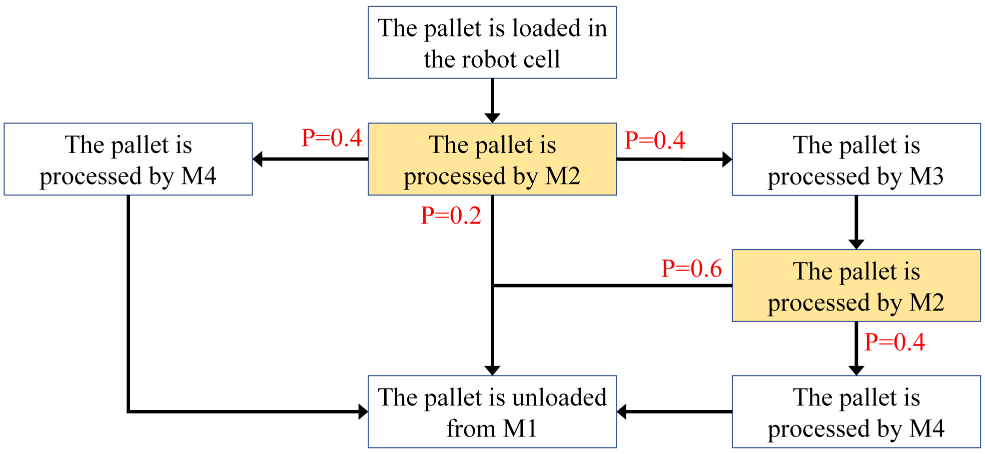

The pallet is loaded in the robot cell
The pallet is processed by M4
P=0.4
The pallet is processed by M2
P=0.4
The pallet is processed by M3
P=0.2
P=0.6
The pallet is processed by M2
P=0.4
The pallet is unloaded from M1
The pallet is processed by M4

Figure 6.2: Testing machine results probability

the task of the controller is just to maximize the throughput, ignoring all the energy issues. Firstly a deterministic model prediction has been implemented, i.e the controller perfectly knows the results of the testing machine along the prediction horizon. Then the robustness of the controller has been tested, referring to the uncertain plant model, where the previous probability distributions are considered known.

For all the following test only two sequences has been adopted: a main sequence s_1 and an additional sequence s_2 representing a different testing machine output. The graphical representation of the main sequences can be found in figure 4.5 and defines the routine of a non working board that is fixed by the reworking machine and is unloaded after the second test in M2. The mathematical formulation is reported below. In order to distinguish the first time the board is processed from the second one, the testing machine target node is denoted by two indexes 34 and 34.2 . Moreover it is possible to observe that the waiting nodes are $\{2, 10, 15, 22, 27\}$, because they are repeated twice along the sequence. The machine nodes has been repeated as many times as corresponding number of the working time steps.

The additional sequence s_2 indicates the trajectory of a board that has been processed by M2 and is going to be discharged before returning to the Load/Unload robot Cell. Its graphical representation is shown in figure 4.6. Along the sequence s_2 the Boolean variable f_p has been set to zero from the node 23 to the node 36, to avoid redundancy in path evaluation. In fact the final sections of the two sequences are identical.

The mathematical formulation of the two sequences is the following:

$$S(s1) = \left[\begin{bmatrix}32\\34\\1\end{bmatrix}, \begin{bmatrix}1\\34\\1\end{bmatrix}, \begin{bmatrix}2\\34\\1\end{bmatrix}, \begin{bmatrix}2\\34\\1\end{bmatrix}, \begin{bmatrix}3\\34\\1\end{bmatrix}, \begin{bmatrix}4\\34\\1\end{bmatrix}, \begin{bmatrix}5\\34\\1\end{bmatrix}, \begin{bmatrix}6\\34\\1\end{bmatrix}, \begin{bmatrix}7\\34\\1\end{bmatrix}, \begin{bmatrix}16\\34\\1\end{bmatrix}, \begin{bmatrix}34\\34\\1\end{bmatrix}, \begin{bmatrix}34\\33\\1\end{bmatrix}, \begin{bmatrix}34\\33\\1\end{bmatrix}, \begin{bmatrix}34\\33\\1\end{bmatrix}, \begin{bmatrix}34\\33\\1\end{bmatrix}, \right.$$
$$\begin{bmatrix}19\\33\\1\end{bmatrix}, \begin{bmatrix}22\\33\\1\end{bmatrix}, \begin{bmatrix}22\\33\\1\end{bmatrix}, \begin{bmatrix}23\\33\\1\end{bmatrix}, \begin{bmatrix}24\\33\\1\end{bmatrix}, \begin{bmatrix}25\\33\\1\end{bmatrix}, \begin{bmatrix}3\\33\\1\end{bmatrix}, \begin{bmatrix}4\\33\\1\end{bmatrix}, \begin{bmatrix}5\\33\\1\end{bmatrix}, \begin{bmatrix}6\\33\\1\end{bmatrix}, \begin{bmatrix}7\\33\\1\end{bmatrix}, \begin{bmatrix}8\\33\\1\end{bmatrix}, \begin{bmatrix}9\\33\\1\end{bmatrix}, \begin{bmatrix}10\\33\\1\end{bmatrix}, \begin{bmatrix}10\\33\\1\end{bmatrix}, \begin{bmatrix}12\\33\\1\end{bmatrix},$$
$$\begin{bmatrix}33\\33\\1\end{bmatrix}, \begin{bmatrix}33\\34.2\\1\end{bmatrix}, \begin{bmatrix}33\\34.2\\1\end{bmatrix}, \begin{bmatrix}33\\34.2\\1\end{bmatrix}, \begin{bmatrix}12\\34.2\\1\end{bmatrix}, \begin{bmatrix}13\\34.2\\1\end{bmatrix}, \begin{bmatrix}17\\34.2\\1\end{bmatrix}, \begin{bmatrix}14\\34.2\\1\end{bmatrix}, \begin{bmatrix}15\\34.2\\1\end{bmatrix}, \begin{bmatrix}15\\34.2\\1\end{bmatrix}, \begin{bmatrix}16\\34.2\\1\end{bmatrix}, \begin{bmatrix}34\\34.2\\1\end{bmatrix},$$
$$\left. \begin{bmatrix}34\\36\\1\end{bmatrix}, \begin{bmatrix}34\\36\\1\end{bmatrix}, \begin{bmatrix}34\\36\\1\end{bmatrix}, \begin{bmatrix}34\\36\\1\end{bmatrix}, \begin{bmatrix}19\\36\\1\end{bmatrix}, \begin{bmatrix}22\\36\\1\end{bmatrix}, \begin{bmatrix}22\\36\\1\end{bmatrix}, \begin{bmatrix}23\\36\\1\end{bmatrix}, \begin{bmatrix}24\\36\\1\end{bmatrix}, \begin{bmatrix}25\\36\\1\end{bmatrix}, \begin{bmatrix}26\\36\\1\end{bmatrix}, \begin{bmatrix}27\\36\\1\end{bmatrix}, \begin{bmatrix}27\\36\\1\end{bmatrix}, \begin{bmatrix}1\\36\\1\end{bmatrix}, \begin{bmatrix}32\\36\\1\end{bmatrix}, \begin{bmatrix}36\\36\\1\end{bmatrix} \right]$$

$$S(s2) = \left[\begin{bmatrix}34\\35\\1\end{bmatrix}, \begin{bmatrix}34\\35\\1\end{bmatrix}, \begin{bmatrix}19\\35\\1\end{bmatrix}, \begin{bmatrix}22\\35\\1\end{bmatrix}, \begin{bmatrix}22\\35\\1\end{bmatrix}, \begin{bmatrix}23\\35\\1\end{bmatrix}, \begin{bmatrix}35\\35\\1\end{bmatrix}, \begin{bmatrix}35\\36\\1\end{bmatrix}, \begin{bmatrix}35\\36\\1\end{bmatrix}, \begin{bmatrix}23\\36\\0\end{bmatrix}, \begin{bmatrix}24\\36\\0\end{bmatrix}, \begin{bmatrix}25\\36\\0\end{bmatrix}, \begin{bmatrix}26\\36\\0\end{bmatrix}, \begin{bmatrix}27\\36\\0\end{bmatrix}, \begin{bmatrix}27\\36\\0\end{bmatrix}, \begin{bmatrix}1\\36\\0\end{bmatrix}, \begin{bmatrix}32\\36\\0\end{bmatrix}, \begin{bmatrix}36\\36\\0\end{bmatrix} \right]$$

By properly combining these two sequences is possible to obtain all board routines previously defined in fig. 6.2 and the default sequence s_1 is assigned to a new board. When it reach the penultimate step in the testing machine (node 34) there are three option: if the board has to be repaired the sequence does not change, if the board works properly the sequences does not change, but the position skips to the point when the board finishes the second test, thus it is sent to the robot cell, finally if the board has to be discharged the additional sequences s_2 is assigned in place of s_1. If the board has been sent for repair, when it returns to the testing machine for the second time, at the penultimate working step the sequence s_1 will remain or s_2 will be assigned, depending on the result of the machine; in any case it will never be able to return to M3. The exogenous signal $a(k)$ is always considered as known and it is constantly set equal to one. In this way, when the max number of pallet is reached, a new pallet will be loaded only after another one will be unloaded. To add the new constraint dealing with the maximum number of pallet in the plant, the entering condition 3.3 has been modified:

$$\begin{aligned} u_{36,32}(k) = 1, \text{ if } \ a(k) = 1 \ \wedge \nexists i : [\mathcal{S}(s_i(k))^{(1,\hat{p}_i(k+1))} = h_l \ \vee \ \mathcal{S}(s_i(k))^{(1,\hat{p}_i(k))} = h_l] \\ \wedge \ N_p(k) < N_{p,max} \\ u_{36,32}(k) = 0, \text{ else.} \end{aligned} \tag{6.1}$$

The new condition is coloured in red and $N_{p,max}$ denotes the maximum number of parts. In this way no part can be loaded if the limit has already been reached in the current step.

In the following sections the results of the test are exposed. The different approaches that have been tested are:

- BF: Brute force or extensive search;
- IMB: Intelligent move blocking;
- IMB(ξ): Intelligent Move Blocking switching to a non-optimal solution;
- IMBSH: Intelligent Move Blocking with sub-horizons.

In all the test the IMB and its variants will be implemented considering the most aggressive approaching direction (AD1: from the most aggressive to the softest compatible state) and a comparison between different APs will be proposed at the end.

Because the purpose of the controller is only to maximize the throughput, the stage cost, computed at each step of the prediction horizon, is defined by the following function:

$$\ell_{N_p(o|k)} = \lambda_1 \sum_{i=1}^{N_p(o|k)} r_i(o|k) + \lambda_2 \sum_{i=1}^{N_p(o|k)} t_i - \lambda_3 u_{32,36}(o|k) \tag{6.2}$$

where λ_1, λ_2, λ_3 are three strictly positive weighting factor. The first term of the stage cost consists in the sum of the remaining nodes for each part and ensures

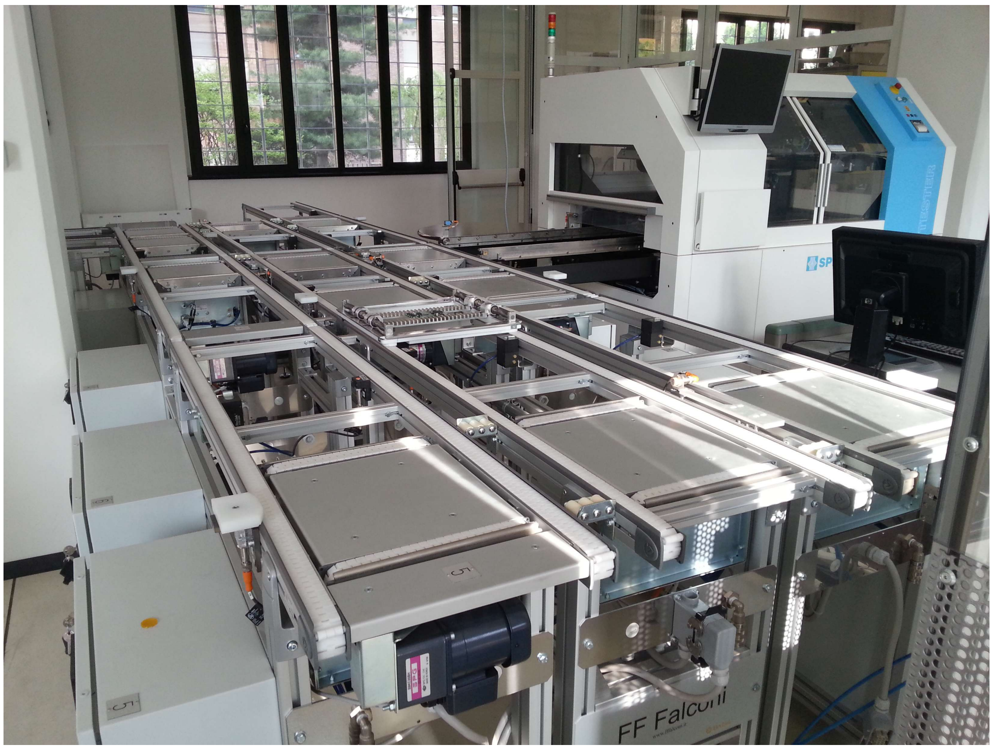


Figure 6.3: Testing machine in the real plant, STIIMA-CNR laboratory, Milano

that the parts are pushed forward along their sequences. The second is the sum of the time counter values t_i and avoids that a part is held in position for too long, even if that choice would allows to move a greater number of parts. The last negative term is a *reword* that is triggered when a pallet is unloaded from the robot cell and reduces significantly the value of the stage cost (considering $\lambda_3 >> \lambda_1, \lambda_2$).

Furthermore two new constraints have been introduced in the plant model. They have been detected during the tests with the real plant simulator and have been added to the path following algorithm via a specific sub-routine. These constraints will be presented in the next chapter, dealing with the communication protocol of the real plant and with the validation of the HRC.

6.1 Test 1

In this test and in the following ones the throughput of the plant is computed as $T_{hr} = N_f(k)/k$ and indicates the number of finished part per step processed in the time interval $[0, k]$. The initial condition consists in one pallet in the robot cell (node 32) and the simulation is interrupted at $k = 2000$. C_t denotes the mean of the computational time $[\,s\,]$ needed to provide the control action for one step, while C_{peak} indicates the computational peak $[\,s\,]$ measured in the simulation. H indicates the length of the prediction horizon.

$N_{p,max}$ = 3 pallets

	$H = 10$			$H = 15$		
	$C_t\ [s]$	$C_{peak}\ [s]$	N_f	$C_t\ [s]$	$C_{peak}\ [s]$	N_f
BF	0,564	10,520	152	8,041	180,962	152
IMB	0,018	0,180	152	0,026	0,267	152
IMB(0)	0,010	0,143	153	0,014	0,175	153

Table 6.2: Extensive search compared with IMB

As anticipated the extensive search (BF) is a non implementable method, because the computational time increases tremendously as function of the number of parts and of the horizon length. In table 6.2 is shown that with very few parts C_t is low, but it is sufficient to increase the length of the horizon by a few steps to obtain high computational peak and to significantly grow C_t. The settings $N_{p,max} = 5$ and $H = 20$ are already enough to need tens of minutes to find the solution at each step (with computational peak of hours). Furthermore it can be noticed that IMB and IMB(0) reach the same throughput of BF, but with almost instantaneous times. Sometimes the IMB(0) seems even a little better than IMB, but that is an not really true because the optimization could choose a worse scenario in the short term in order to obtain better results in the future. Thus the comparison depends on when the simulation has been interrupted. When the plant becomes more congested the importance of optimization is verified.

The extensive search has been abandoned in the next tests, where a greater number of parts is introduced in the plant and a longer horizon length is adopted.

$N_{p,max}$ = 5 pallets

	IMB			IMB(0)		
	C_t [s]	C_{peak} [s]	N_f	C_t [s]	C_{peak} [s]	N_f
$H = 10$	0,036	0,272	208	0,018	0,158	210
$H = 30$	0,108	1,090	211	0,049	0,283	210
$H = 50$	0,247	2,686	211	0,076	0,352	210

$N_{p,max}$ = 10 pallets

	IMB			IMB(0)		
	C_t [s]	C_{peak} [s]	N_f	C_t [s]	C_{peak} [s]	N_f
$H = 10$	0,157	3,824	209	0,062	0,589	203
$H = 30$	0,499	3,596	211	0,192	2,287	203
$H = 50$	0,811	6,252	215	0,292	3,026	203

$N_{p,max}$ = 13 pallets

	IMB			IMB(0)		
	C_t [s]	C_{peak} [s]	N_f	C_t [s]	C_{peak} [s]	N_f
$H = 10$	0,330	28,663	199	0,363	69,346	196
$H = 30$	1,368	34,177	216	0,927	16,088	199
$H = 50$	/	$> 10^3$	/	/	$> 10^3$	/

Table 6.3: Test 1 results

The test results 6.3 shows that the optimization is not needed when the plant in not congested. In fact for $N_{p,max} \leq 5$ the number of finished parts is almost identical both for IMB and IMB(0). It can be noticed that the optimization provides surely a better result when $N_{p,max} \geq 10$. The computational peaks rarely occur, see figure 6.4, but in the worst cases the algorithm may take minutes to provide the control action when too many sequences has been generated, there are too many parts in the plant or the prediction horizon is too long (for example $H \geq 50$ and $N_{p,max} \geq 13$).

As you can see from 6.3 the IMB(0) method seems to solve the problem of computational peaks, but for a greater number of part in the plant, for example $N_{p,max} = 13$, this problem affects also the non-optimal solution. That happens because sometimes the parts are in particular configuration for which is more difficult to find a non-blocking solution and the lockout-avoidance backtracking causes the generation of computational peaks. In some cases the non-optimal solution incurs even higher peaks because choosing the most aggressive AD the controller try to move forward as many parts as possible, increasing lockout probability. Anyway the algorithm is very fast and when the plant is not congested excellent performances are guaranteed even with no optimization, thanks to a proper definition of the main sequence and to the choice of an aggressive AD.

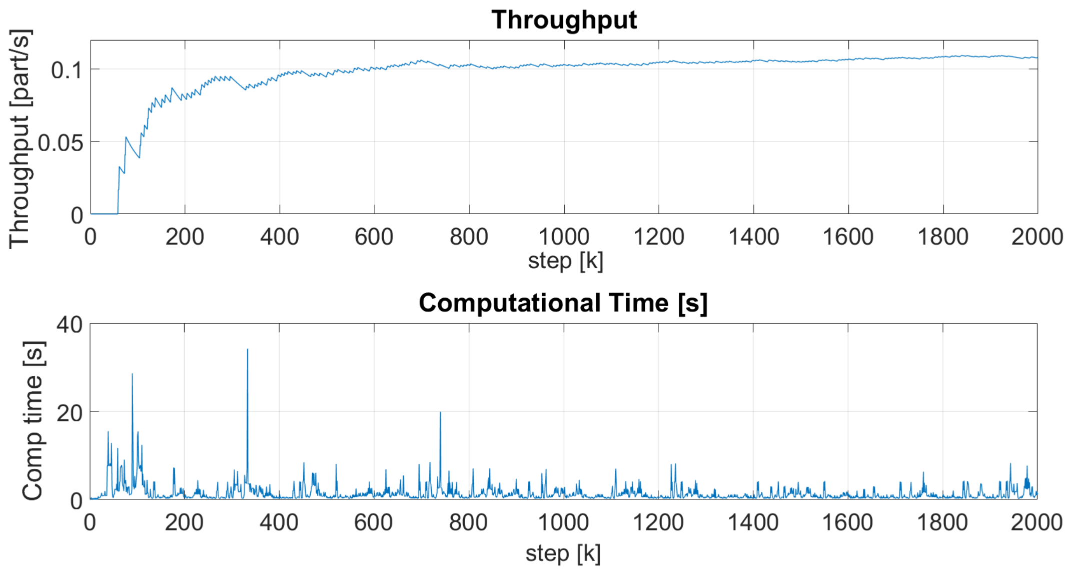


Figure 6.4: Intelligent Move Blocking with $N_{p,max} = 13$ and $H = 30$: throughput and computational time per step

6.2 Test 2

In the second test the different Approaching Directions, previously defined in the section 5.3.1, are compared: AD1, AD2, AD3, AD4, AD5.

The simulation is interrupted at $k = 2000$ and the initial condition is one pallet in node 32. Moreover it has been set $N_{p,max} = 10$ and $H = 30$.

$N_{p,max} = 10$ pallets, $H = 30$

	IMB			IMB(0)		
	C_t [s]	C_{peak} [s]	N_f	C_t [s]	C_{peak} [s]	N_f
AD1	0,499	3,596	211	0,062	0,589	203
AD2	1,462	14,084	204	0,547	6,167	105
AD3	1,198	9,097	206	0,214	0,613	180
AD4	0,665	6,105	208	0,161	1,325	194
AD5	0,667	4,320	205	0,238	1,294	118

Table 6.4: Test 2: ADs comparison

It is possible to observe from table 6.4 that the softest Approaching Direction (AD2) provides the worst throughput but not the better computational time. In fact it can be assumed that a more aggressive control action determines a better pallet flow, avoiding the formation of congestion that would cause computational peaks. Furthermore a soft AD incurs more frequently theoretical lockouts. In fact the most aggressive AD (AD1) is the one that allows to obtain the greater number of finished parts and the better computational time performances. Some of of the tested Approaching Directions has demonstrated to be a trade-off, for example AD3 and AD4 obtain good computational time and an excellent number of finished parts. AD5 has shown good results only of the optimizing method IMB.

The following table 6.5, indicates how the C_t, C_{peak}, N_f change as function of

ξ. The test are performed setting AD2: it is observed that the soft Approaching Direction is compensated by the optimization and excellent throughput performances are reached anyway with IMB.

$\boldsymbol{N_{p,max}}$ **= 10 pallets,** $\boldsymbol{H}$ **= 30**

	AD2		
	C_t $[s]$	C_{peak} $[s]$	N_f
IMB	1,462	14,084	204
IMB(8)	1,144	6,281	199
IMB(4)	0,623	5,981	144
IMB(0)	0,547	6,167	105

Table 6.5: Test 2: AD2 performances with respect to IMB(ξ)

Unfortunately the choice of a less aggressive AD does not always solve the problem of high C_{peak}, in particular when the number of parts increases, see table 6.6. The problem could be solved by parallelizing the exploration of different branches or by anticipating the calculation of the future control actions while pallet movements are executed in the real plant. In case a mismatch between the predicted behaviour and the measured one is detected, the control algorithm returns to compute the actual one.

$\boldsymbol{N_{p,max}}$ **= 13 pallets,** $\boldsymbol{H}$ **= 50**

	IMB		
	C_t $[s]$	C_{peak} $[s]$	N_f
AD1	/	$> 10^3$	/
AD2	13,089	217,474	202
AD3	3,109	15,842	200

Table 6.6: Reduction of C_{peak} depending on ADs

6.3 Test3

The purpose of the third test is to verify if dividing the prediction horizon into sub-horizons can improve the quality of the solution. The controller has been set with AD2 and three different horizon configurations have been tested: a single horizon of 30 steps, two sub-horizon of 15 steps, five sub-horizons of 6 steps. The simulation has been interrupted at $k = 2000$ and the maximum number of parts has been set to 10. In this case the IMBSH method has been implemented nesting IMB(4) problems. The results are reported in the following table 6.7:

$N_{p,max}$ = 10 pallets

	IMBSH(4), AD2		
	C_t $[s]$	C_{peak} $[s]$	N_f
$H = 1 \times 30$	0,623	5,981	144
$H = 3 \times 10$	1,753	16,026	199
$H = 5 \times 6$	3,129	37,370	200

Table 6.7: IMBSH compared with IMB

The throughput grows depending on the number of sub-horizons. In fact at the end of each sub-horizon a new control problem is set and the number of optimization variables grows. The improvement of the solution corresponds to greater computational times. The IMBSH method can be considered as a *tunable extensive tree search* and can be very useful when the optimization has a fundamental role, for example when many sequences has been generated and the the controller has more decision-making degrees of freedom.

6.4 Test4

The following test has been performed to prove the robustness of the control algorithm, adopting a non-deterministic model of the plant, i.e the results of the testing machine are not known a priori. The same probability distributions indicated in figure 6.2 have been implemented along the prediction in order to estimate the future target machine of a part. The simulation has been interrupted at $k = 2000$ and the controller operates with the IMB strategy.

$H = 30$, IMB

	Deterministic			Non-deterministic		
	C_t $[s]$	C_{peak} $[s]$	N_f	C_t $[s]$	C_{peak} $[s]$	N_f
$N_{p,max}$ = 3 pallets, AD1	0,075	0,636	149	0,075	0,768	149
$N_{p,max}$ = 5 pallets, AD1	0,108	1,090	211	0,189	1,689	206
$N_{p,max}$ = 10 pallets, AD1	0,499	3,596	211	0,462	3,313	201
$N_{p,max}$ = 13 pallets, AD1	1,368	34,177	216	2,000	56,285	200
$N_{p,max}$ = 13 pallets, AD2	1,462	14,084	204	3,265	6,339	193

Table 6.8: Robustness test

As shown in table 6.8, the controller is able to avoid lockout formation even when the results of the testing machine are provided online. Moreover for an aggressive Approaching Direction (AD1) the throughput is almost the same with $N_{p,max} \leq 5$, while decreases when the plant is more congested. Moreover the computational cost increases due to the unexpected congestion of the non-deterministic model. When the optimization has a more important role, such as with a soft Approaching Direction AD2, the number of finished part is reduced more significantly.

It is important to underline that the sequence definition is fundamental to guarantee the robustness of the controller. In fact the assignable sequences must allow to avoid the lockout immediately after a part exits from the testing machine (M2). Considering the sequences defined for the previous tests: a part exiting from M2 performs for some nodes the same path both to go to M3, to go to M4 or M1. In this way the controller has time to correct the route of all the parts in order to avoid the lockout. On the contrary, if immediately after the end of the test the processed part would take two or more different paths, it would not be possible to avoid the lockout. For example if a part in M2 to go to M3 moves through the nodes $19, 16, 7, 8...$, in case of a wrong prediction a lockout can occur between the part that is entering in the testing machine and the part that is going out. The only way to ensure that there is always a solution for the control problem would be to evaluate all alternative future scenarios and check that the lockout is avoided for every M2 results.

In addition the tests have demonstrated that, for example, with $N_{p,max} \geq 10$ the prediction horizon must be greater than eight steps to be able to avoid a lockout. This value is empirically derived and change depending on the number of parts in the plant and on the assignable sequences.

6.5 Test5

This test deals with the Move Blocking with controllable exogenous-input and emptying condition. Different assumption must be introduced.

- The prediction horizon is no more fixed because the length of the simulation depends on how many steps are needed to empty the plant.
- The signal $a(k)$ is controlled in order to let a new part in only if it will not cause a lockout.

In this case sequences represent the control design variable and the plant performances are directly related to the number of sequences defined. The average number of parts present in the plant at each time step has been denoted by $\bar{N_p}$ and no limitation has been set on the maximum $N_p(k)$. The cost to be minimized corresponds to the length of the prediction horizon, thus the optimal solution is the one that allows to empty the plant in the shortest possible time.

6.5.1 Case 1

Firstly the same sequences of the previous tests are adopted and the simulation is stopped at $k = 500$, with a part in node 32 as initial condition. In the following table 6.9 are reported the results of the test and in figure 6.5 are shown the values of $N_p(k)$ and the computational time per time step.

One main sequence			
$C_t\ [s]$	$C_{peak}\ [s]$	N_f	$\bar{N_p}$
0,856	4,708	47	6,850

Table 6.9: Move blocking with one main sequences test results.

This control method provides a non stable throughput, because it tries to load the maximum possible number of parts ensuring that the system does not run into a lockout. When it is no longer able to insert new parts those already in the plant are unloaded (the priority is given always to the part that is leaving than the one that is entering the plant).

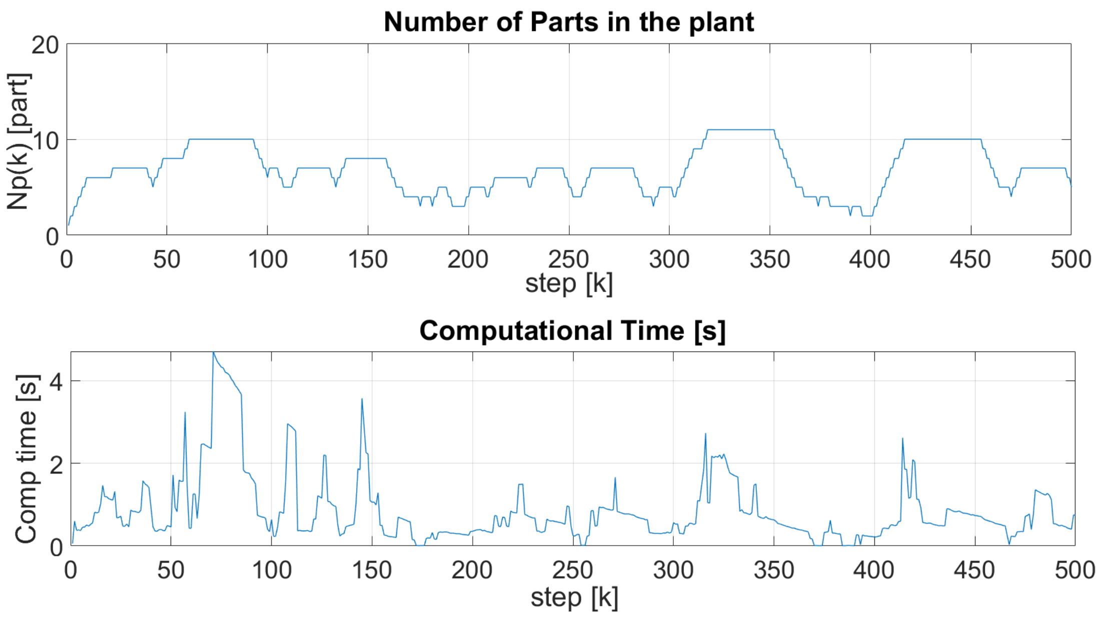


Figure 6.5: Move Blocking with one main sequence: plot of the number of parts $N_p(k)$ and computational time per step $C_t(k)$.

6.5.2 Case 2

Then the main sequence has been replaced with two sequences, both identical to the previous one but with these differences:

- Main sequence 1: the waiting nodes have been repeated twice except for the ones in correspondence of a machine, which have been repeated as many time as the specific working time L_m.
- Main sequence 2: no node is repeated along the sequence.

The simulation has been stopped at $k = 500$ and the results are indicated in the table 6.10 below, in figure 6.6 are shown the values of $N_p(k)$ and the computational time per time step.

Two main sequences			
C_t $[s]$	C_{peak} $[s]$	N_f	$\bar{N}_p$
6,129	72,315	50	8,222

Table 6.10: Move blocking with two main sequences test results.

In this second test $\bar{N}_p$ increases and the controller is able to load simultaneously a greater number of parts in the plant. In fact assigning many different sequences there are more possibility to manage a new part. Unfortunately the computational times increases and would become unsustainable with too many sequences. This depends on the fact that the length of the variable prediction horizon usually is very high due to the emptying condition ($H \geq 100$).

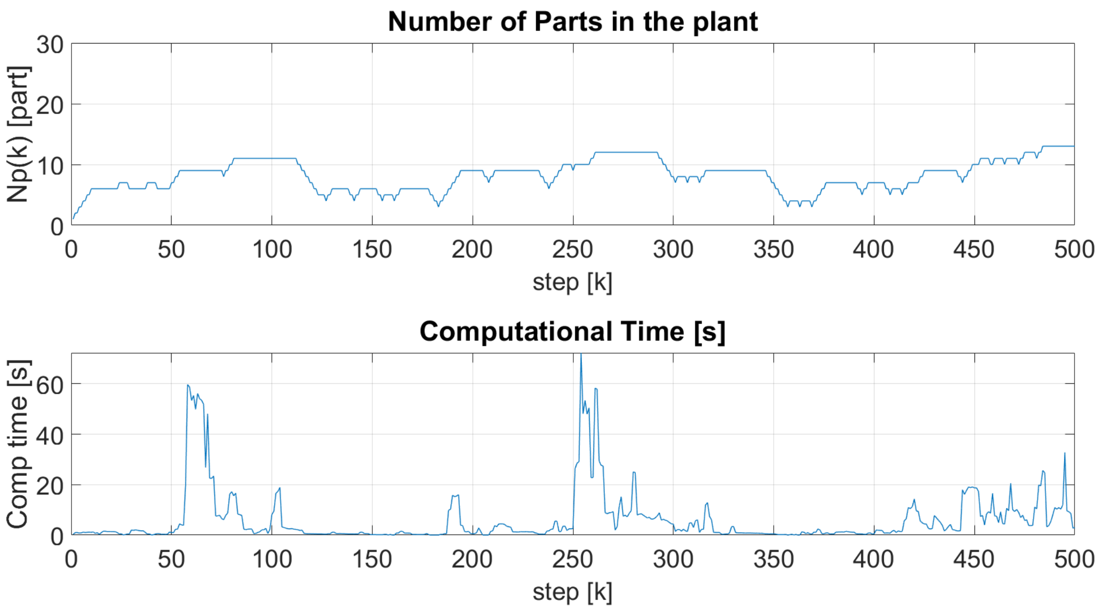


Figure 6.6: Move Blocking with two main sequences: plot of the number of parts $N_p(k)$ and computational time per step $C_t(k)$.

6.6 Throughput results

The variability of the processes performed by a part inside the plant makes it difficult to theoretically calculate the ideal maximum throughput value. Considering a line with infinite buffer, the maximum throughput corresponds to the bottleneck machine, i.e. the machine with the higher L_m. In the tests performed the slowest machine is M2 with $L_m = 5$. Moreover, if M2 is processing a part, due to the constraint of the handling system, the part in the node 16 has to wait two more steps before entering in M2 in order to get the other part out. Thus the bottleneck must be evaluated summing these additional steps to the working time. The derived maximum throughput is $T_{hr} = 0,14$ [part/step]. This is an optimistic approximation because the following issues have not been taken into account: other constraints on transitions, conflicts between different paths, variability of the targets.

In the previous tests the best throughput value is $T_{hr} = 0,11$ (i.e. 216 parts in 2000 steps), obtained with IMB, AD1, $N_{p,max} = 13$, H= 30. In most cases the throughput is greater than 0,1, in table 6.11 the T_{hr} values for AD1 are indicated at steady state (i.e., from step $k = 1600$ to step $k = 2000$). It is possible to conclude that the HRC strategy allows to reach excellent throughput performances with really low computational cost for the considered application.

Throughput [part/k], $k \in [1600, 2000]$, AD1

	IMB			IMB(0)		
	$H = 10$	$H = 30$	$H = 50$	$H = 10$	$H = 30$	$H = 50$
$N_{p,max}$ = 5 pallets	0,1075	0,1075	0,1075	0,1075	0,1075	0,1075
$N_{p,max}$ = 10 pallets	0,11	0,1125	0,1125	0,11	0,11	0,11
$N_{p,max}$ = 13 pallets	0,11	0,1125	/	0,11	0,095	/

Table 6.11: Throughput AD1 comparison

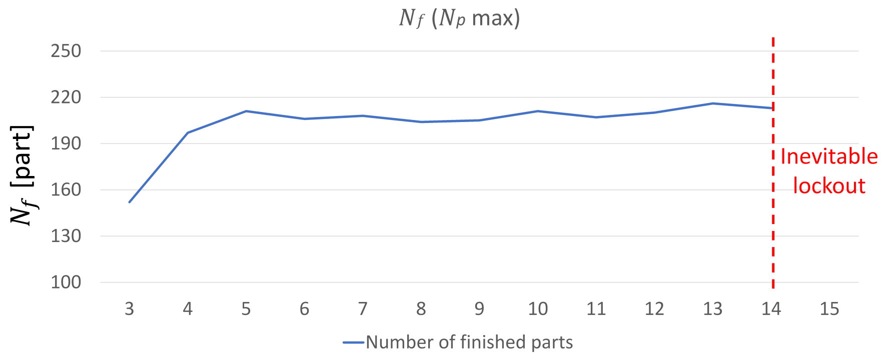


Figure 6.7: N_f depending on $N_{p,max}$ (tested with IMB, AD1 and H=30)

In figure 6.7 the number of finished parts is represented as a function of the maximum number of parts in the plant $N_{p,max}$. It can be noticed that with $N_{p,max} = 5$ almost the best throughput performances are reached, thus it makes no sense to insert a greater number of pallets. Moreover $N_{p,max} = 14$ is the limit beyond which the controller is no more able to avoid a lockout. To overcome this limit it is necessary to generate a greater number of sequences.

Even if almost the same N_f is obtained at the end of the test, a more aggressive AD ensures to reach the operative condition faster than a softer one. In figure 6.8 the throughput of the most aggressive AD and the softest one are compared (AD1 and AD2 respectively).

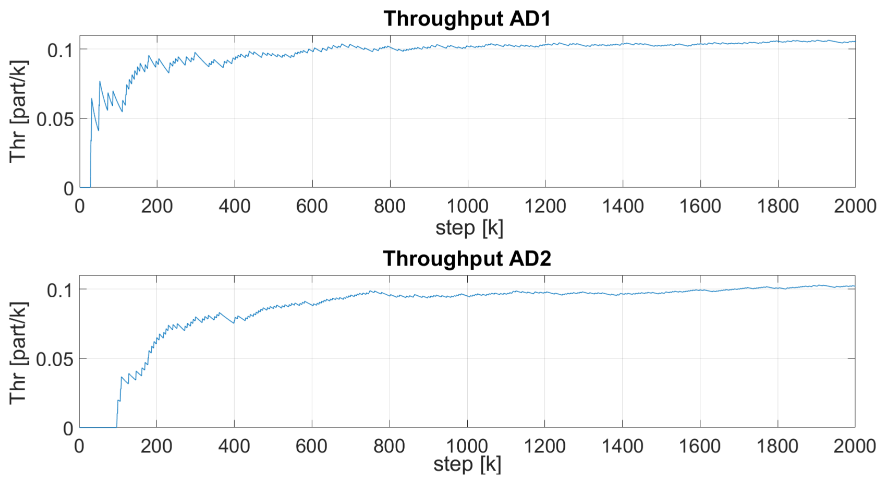


Figure 6.8: T_{hr} of IMB with H=30 and $N_{p,max} = 10$, AD1 and AD2 comparison

Chapter 7

Test on the real plant simulator

Finally the algorithm has been tested on the simulator of the pilot plant, developed on SIMIO and provided by the STIIMA-CNR laboratory. The simulator identically replicates the plant operations, because the actuators and sensors behavior and the corresponding I/O exchanged signal are faithfully reproduced. Anyway two differences with the real plant should be highlighted:

- **The communication time** needed to exchange data between the plant components is always constant, while in the real case it is be subjected to possible delay and/or signal faults. On the pilot plant that fact occurs because the instrumentation communicates by means of a fieldbus, which behaviour is not replicated in the SIMIO model.

- **The simulator operates in nominal condition**, in fact the pallets always correctly perform the movements imposed by the control inputs, while in the real plant they could get stuck or their position could not be properly detected by means of any sensor failure.

These two approximations do not affect the validation of the control algorithm developed by this thesis. The fault detection and recovery problem has been already solved on the plant and the regarding algorithm has been directly included in the real plant low-level logic control, see [19]. In any case the purpose of this thesis is to verify the Hierarchical Routing Control in nominal condition, no communication or transport faults have been taken into account, thus the simulator is perfectly suitable with the assumptions of the considered control problem.

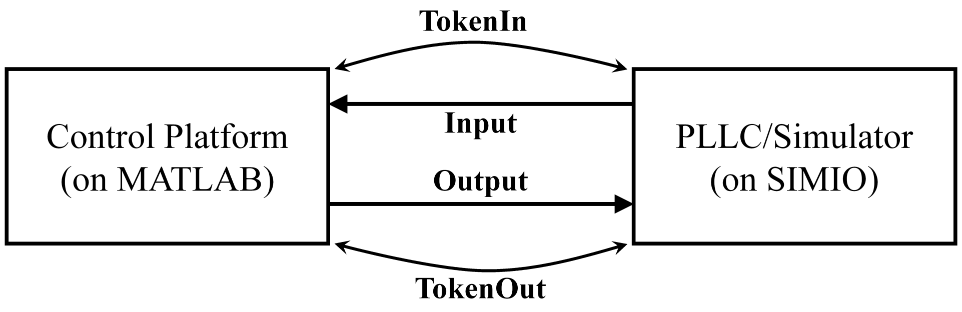


Figure 7.1: I/O data connection

In order to be able to interact with the plant, the control algorithm developed on MATLAB has to be set up for exchanging I/O data. The simulator adopts the same I/O interface of the real plant. The Control Platform (CP), in our case MATLAB, communicates with the Plant Low Level Control (PLLC) system, in our case the implemented Low Level control logics on SIMIO, in order to read the state of the system and write the control inputs. The data communication is performed via text files shared by CP and PLLC and it is regulated by a token exchange protocol, figure 7.1. It is characterized by two token: *TokenIn* and *TokenOut*, the first one for the input signal and the second for the output signal. Their functioning is the following:

- *TokenIn*: the PLLC assign the value 1 the file “TokenIn.txt” to indicates that is in writing phase. The CP can not access the input data file until the PLLC writes the integer 2 into the file “TokenIn.txt”. When the CP terminates to read the input data files, it returns the token to the PLLC and writes the value 1 in the file “TokenIn.txt”.

- *TokenOut*: the CP assigns the value 1 the file "TokenOut.txt" to indicates that is in writing phase. The PLLC can not access the output data file until the CP writes the integer 2 into the file "TokenOut.txt". When the PLLC terminates to read the output data files, it returns the token to the CP and writes the value 1 in the file "TokenOut.txt".

In the input data files the PLLC writes the Eulerian overall state of the plant $\boldsymbol{z}(k)$ (indicating the target assigned to the parts), while in the output data files the CP writes the vector of the control inputs $\boldsymbol{U}(k)$. Thus the PLLC communicates to the CP the state of the plant, the CP computes the control action and send it to PLLC, then the CP waits for computing the next control action until the imposed motions has been performed, i.e. when the PLLC communicates the next step state.

The control loop is closed on the Eulerian state, thus the simulator is just used to validate the plant model implemented in the control algorithm. The Lagrangian state is known and updated only by the control algorithm. Because the plant is assumed to work in nominal conditions there is expected to be no mismatch between the measured Eulerian state and the predicted one.

Thanks to this validation two missed constraints have been detected:

$$\textit{if } \exists\, i = 1, \dots, N_p(k) \ : \ \mathcal{S}(s_i(k))^{(1,p_i(k))} = 19 \ \textit{ then } \ u_{16,34}(k) = 0 \tag{7.1a}$$

$$\textit{if } \exists\, i = 1, \dots, N_p(k) \ : \ \mathcal{S}(s_i(k))^{(1,p_i(k))} = 21 \ \textit{ then } \ u_{23,24}(k) = 0. \tag{7.1b}$$

The condition 7.1a indicates that the transition related to the control input $u_{16,34}(k)$ can no occur if the node 19 is occupied. That because the part moving from the node 16 to the node 34 would crash with the part in node 19. The condition 7.1b indicates the same type of constraint, but for the node 21 and the transition associated to the control input $u_{23,24}(k)$. In the Greedy Path Following Strategy these new constraints have been included modifying the subroutine of Algorithm 5: if the node 19 is occupied and a part has to move from the node 16

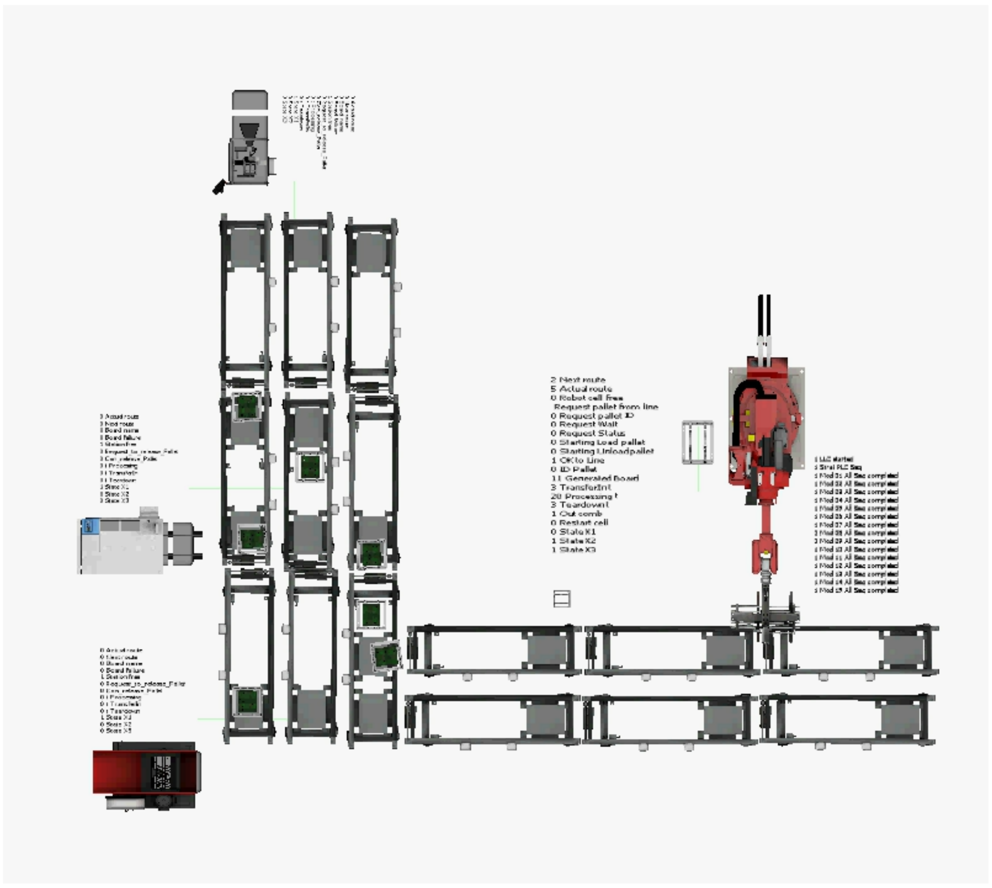


Figure 7.2: Frame from the simulator view on SIMIO

to the node 34, the part in node 16 is held in position (the same for the node 21 and a part in node 23 that has to move in node 24). The vector of the blocking index $\boldsymbol{Q}$, defined in 5.1, is updated in order to assign the part in node 19 as blocking index for the part in node 16 (the same for the part in node 21 and in node 23).

Finally the constraint on the pair of transitions $u_{19,22}$ and $u_{16,34}$ has been removed (from the set 3.12), because is redundant. In fact if there is a part in the node 19 the part in node 16 is not able to move first.

Several tests have been performed in order to demonstrate the correct operating of the communication between the control algorithm and the PLLC. Furthermore the control action computed by the Greedy Path Following Strategy is always feasible and consistent with the plant constraints. The model of the plant developed by this thesis and implemented in Matlab has been validated, thus is possible to confirm the reliability of the tests proposed in the previous chapter (performed considering the two new constraints 7.1).

Chapter 8

Conclusions

In this thesis the Hierarchical Routing Control for discrete manufacturing plants has been implemented and tested on a real case for the first time. The low-level control logic (GPFS) has been improved in order to consider the specific constraints of the demanufacturing pilot plant in the CNR laboratory. In particular two subroutines have been added to the path following algorithms:

- **Algorithm 4** has the task of ensuring that all transitions comply with the constraints collected in the *constrained node* property.
- **Algorithm 5** has the task of computing the priority for couple of transitions that can not occur at the same time. In addition this subroutine deal with two specific constraints defined in 7.1.

A new mathematical sequence formulation has been proposed, in order to eliminate the redundancy in the evaluation of compatible sequences. Thanks to that it is possible to obtain the same results but with a much lower computational cost. Furthermore some guidelines to generate the sequences have been proposed and three types of sequences has been defined: *the main sequences*, *the additional sequences*, *the supporting sequences*. Some nodes along the sequences have been named *waiting nodes*, characterized by the fact that are repeated twice, allowing to hold a part in position for an unlimited number of steps. They are useful especially to avoid lockout and to force the part priority at the crossroads. The advantage of a path allocating controller is that the optimization problem can be partially solved a priori by choosing as main sequences the one that connect all the targets with the shortest path with the smallest number of crossroads.

Then two conditions have been introduced that allow to detect the formation of *local lockout* and *theoretical lockout*, respectively. A local lockout occurs when at least two parts are reciprocally blocking each other, while a theoretical lockout consist in the evolution of the plant state to an equivalent one. The ability to prevent a lockout is very important to ensure the correct plant behaviour along the prediction horizon and to interrupt the exploration of wrong branches when exploring the search tree.

This ability has been adopted in the Move Blocking formulation of the FHOCP to introduce a lockout-avoidance backtracking, i.e. when a lockout is detected along the prediction horizon the state is returned one step back and a different compatible state is chosen. In this way additional optimization variables are introduced only if needed. Furthermore considering the assumption that a main sequences has been generated with no supporting sequences and the pallet flow is managed only by means of the waiting nodes, the concept of *Approaching Direction* has been introduced. These assumptions allow to ordinate the set of the compatible states and thus it is feasible to direct the exploration in the search tree. The formulation of the control problem with these two improvements has been called *Intelligent Move Blocking* (IMB). In addition some variants have been implemented to provide a larger number design possibilities: IMB(ξ) and IMBSH.

The results of the tests have shown that choosing the most aggressive Approaching Direction (AD1) it is possible to reach an high throughput value even with no optimization (IMB(0)), obtaining also better computational times. In this case, when there are few part in the plant, the number of finished part is identical to the optimization case. Considering IMB and all its variants, generally the obtained computational time is very low, except for few peaks that occur when the plant is too congested.

The control algorithm has been set up to interact with the real plant. In order to do that a simulator replicating the same communication structure and the same plant features has been used. A token exchange protocol has been implemented on the control algorithm to be able to read the actual state and send to the plant the corresponding control action. The functioning of the Hierarchical Routing Control has been successfully verified: the control action provided is feasible and all the plant constraints are satisfied. Furthermore the developed Eulerian and Lagrangian models have been shown to be faithful to the real system.

In conclusion the new HRC approach for discrete manufacturing plant has been proven to be very effective in managing a flexible handling system. All the constraints related to the specific transportation module has been properly translated into low-level logic and the temporal constraints due to the machine tasks have been satisfied thanks to a correct sequence definition. The controller ensures lockout avoidance even for long prediction horizon when the sequences has been generated properly. The obtained control algorithm is very fast considering the complexity of the problem and the amount of constraints. Furthermore the computational time depends more on the number of parts and sequences than on the length of the prediction horizon. This approach proved to be extremely agile in organizing the flow of pallets with different targets and conflicting paths, obtaining a stable and excellent throughput.

Future research works in this area could address the following issues:

- The sequence generation process could be automated defining a standard procedure or solving a specific *shortest path problem.* In this thesis some guidelines have been proposed, but the process is still knowledge based and the paths are empirically produced. Furthermore the concept of Approaching Direction has been developed on the assumption that there is only one

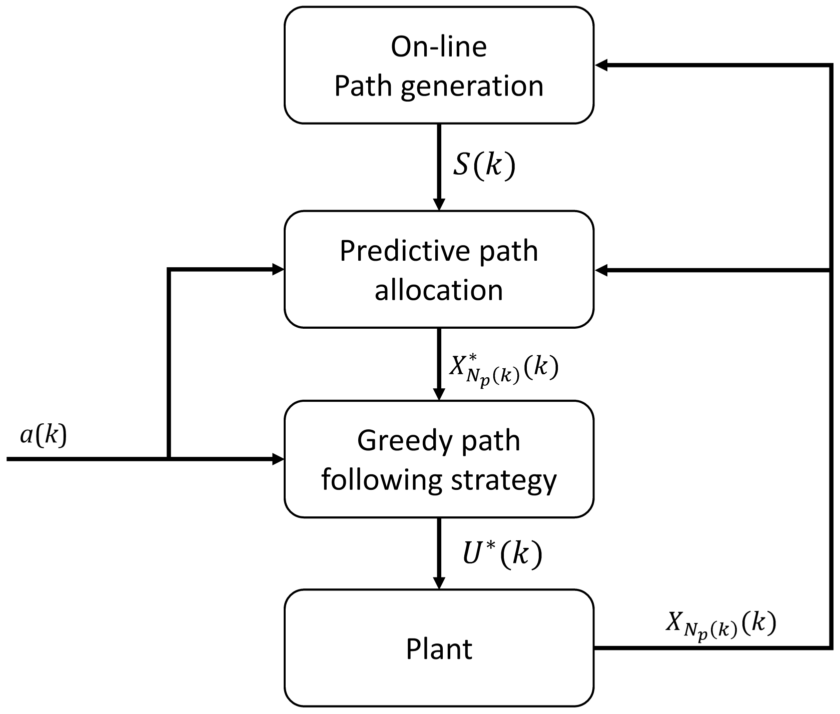


Figure 8.1: 3-Layer Hierarchical Routing Control

main sequences with waiting nodes. This concept has to be expanded also in case more sequences are provided or there are loops inside a sequence.

- The computational peak problem must be solved. When there are many parts in the plant sometimes the controller struggle to find the non blocking solution. One could try to parallelize the search in different branches or to anticipate the calculation of the control action for the future steps, if it is verified that the behavior of the plant corresponds to the prediction.

- The robustness to target uncertainty must be improved. A solution could be to branch out the search tree including a different scenario for each assignable target. To find the robust solution the lockout must be avoided in all the possible scenarios. Obviously the dimension of the search tree increases significantly and more advanced tree search could be adopted. For example the *Monte Carlo tree search* is an heuristic tree search used to develop artificial intelligence for board game [20], such as chess or Go, where the number of board configuration is around 10^{170} possibilities.

- The control algorithm must be adapted to work even in non nominal condition. In fact in the real plant sometimes a pallet could get stuck or stop at a wrong node. In those cases the controller must be able to assign a new path to correct the trajectory of that pallet.

- A third control layer could be added to the Hierarchical Routing Control structure. If the path generation is automated it is possible to implement an *On-line Path Generation* that at each step receives as input the actual state $\boldsymbol{X}_{N_p(k)(k)}$ and provide to the Model Predictive Path Allocation the set of the sequences to be assigned $S(k)$. That can be useful to compute new paths when some nodes are no more available (for malfunctions or programmed maintenance operations) or can be useful to test different sequences and remove the inefficient ones from the set. The block diagram of the three-layer Hierarchical Routing Control with On-line Path Generation has been indicated in figure 8.1.

Bibliography

[1] A. Cataldo and R. Scattolini, "Modeling and model predictive control of a de-manufacturing plant," in *2014 IEEE Conference on Control Applications (CCA)*, pp. 1855–1860, 2014.

[2] T. Tolio, A. Bernard, M. Colledani, S. Kara, G. Seliger, J. Duflou, O. Battaia, and S. Takata, "Design, management and control of demanufacturing and remanufacturing systems," *CIRP Annals*, vol. 66, no. 2, pp. 585 – 609, 2017.

[3] J. Williams, "A review of electronics demanufacturing processes," *Resources, Conservation and Recycling*, vol. 47, no. 3, pp. 195 – 208, 2006.

[4] M. Colledani, G. Copani, and T. Tolio, "De-manufacturing systems," *Procedia CIRP*, vol. 17, pp. 14 – 19, 2014. Variety Management in Manufacturing.

[5] M. Bailey-Van Kuren, "Automated demanufacturing studies in detecting and destroying, threaded connections for processing electronic waste," in *Conference Record 2002 IEEE International Symposium on Electronics and the Environment (Cat. No.02CH37273)*, pp. 295–298, 2002.

[6] M. B. . Kuren, "A lean framework for prototyping demanufacturing work cell automation," in *Proceedings 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, vol. 1, pp. 663–668 vol.1, 2003.

[7] L. Fagiano, M. Tanaskovic, L. Cucas Mallitasing, A. Cataldo, and R. Scattolini, "Hierarchical routing control in discrete manufacturing plants via model predictive path allocation and greedy path following," in Proceedings of the 59th IEEE Conference on Decision and Control, Jeju Island, Republic of Korea - December 14th-18th 2020.

[8] Lalo Magni and Riccardo Scattolini, *ADVANCED and MULTIVARIABLE CONTROL*, ch. 12. Pitagora Editrice Bologna, 2014.

[9] P. Wenzelburger and F. Allgöwer, "A novel optimal online scheduling scheme for flexible manufacturing systems," *IFAC-PapersOnLine*, vol. 52, no. 10, pp. 1 – 6, 2019. 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019.

[10] F. D. Vargas-Villamil and D. E. Rivera, "A model predictive control approach for real-time optimization of reentrant manufacturing lines," *Computers in Industry*, vol. 45, no. 1, pp. 45 – 57, 2001. COMPUTERS IN THE SEMICONDUCTOR INDUSTRY.

[11] W. Wang and D. E. Rivera, "Model predictive control for tactical decision-making in semiconductor manufacturing supply chain management," *IEEE Transactions on Control Systems Technology*, vol. 16, no. 5, pp. 841–855, 2008.

[12] M. A. Bermeo and C. Ocampo-Martinez, "Energy efficiency improvement through mpc-based peripherals management for an industrial process test-bench," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 648 – 653, 2019. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

[13] A. Cataldo and R. Scattolini, "Dynamic pallet routing in a manufacturing

transport line with model predictive control," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 5, pp. 1812–1819, 2016.

[14] R. Cagienard, P. Grieder, E. C. Kerrigan, and M. Morari, "Move blocking strategies in receding horizon control," in *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, vol. 2, pp. 2023–2028 Vol.2, 2004.

[15] Jørgen Bang-Jensen and Gregory Z. Gutin, *Digraphs: Theory, Algorithms and Applications*, ch. 3.3.3. Springer - Monographs in mathematics, Springer, 2010.

[16] Takahashi Jun-ya, Suzuki Hitoshi, and Nishizeki Takao, "Algorithms for finding non-crossing paths with minimum total length in plane graphs," in *Algorithms and Computation*, pp. 400–409, 1992.

[17] Zhengxin Chen, *Computational Intelligence for Decision Support*, ch. 2.8.1.1. International series on computational intelligence, CRC press, 2001.

[18] Zhongzhi Shi, *Advanced Artificial Intelligence*, ch. 3.2. Series on intelligent science; Vol 4, World scientific Publishing, 2019.

[19] A.Cataldo, M.Morescalchi, and R. Scattolini, "Fault-tolerant model predictive control of a de-manufacturing plant," *The International Journal of Advanced Manufacturing Technology 104*, pp. 4803–4812, 2019.

[20] M. C. Fu, "Monte carlo tree search: A tutorial," in *2018 Winter Simulation Conference (WSC)*, pp. 222–236, 2018.

Ringraziamenti

Ringrazio il professor Lorenzo Fagiano per avermi fatto appassionare agli argomenti trattati in questa tesi e per il supporto ricevuto durante la stesura. Inoltre ringrazio il dr. Andrea Cataldo per la grande disponibilità e per il prezioso aiuto fornitomi. Un immenso ringraziamento è rivolto alla mia famiglia, in particolare ai miei genitori e ai miei fratelli, per avermi sostenuto e per essermi stati sempre vicini. Infine ringrazio tutti gli amici, specialmente chi ho conosciuto durante questi indimenticabili anni al Politecnico di Milano, con cui ho potuto condividere difficoltà e traguardi.