Preprint of paper: L. Fagiano, M. Tanaskovic, L. C. Mallitasig, A. Cataldo and R. Scattolini, "Advanced Hierarchical Predictive Routing Control of a Smart De-manufacturing Plant" 19th European Control Conference, Rotterdam (NL), 2021, accepted

Advanced Hierarchical Predictive Routing Control of a Smart De-manufacturing Plant

R. Boffadossi, L. Fagiano, A. Cataldo, M. Tanaskovic, and M. Lauricella

Abstract— **The application of a novel approach to the routing control problem of a real de-manufacturing plant is presented. Named Hierarchical Predictive Routing Control (HPRC) and recently proposed in the literature, the approach deals with large number of integer inputs and complex temporal-logic constraints by adopting a low-level path-following strategy and a high-level predictive path allocation. Several improvements are presented, including a novel search tree exploration method, lockout detection routines, and plant-specific handling constraints. Simulation results show very good performance and small computational times even with high number of pallets and long prediction horizon values.**

I. Introduction

Intelligent (or smart) manufacturing is a key concept of the Industry 4.0 paradigm [1]. It refers to the use of advanced information, communication and control technologies to increase the flexibility and efficiency of manufacturing plants and entire supply chains. Research and development in intelligent manufacturing encompasses many areas [2], ranging from sensors to data management systems, from machines and transportation modules to communication systems, from cybersecurity to artificial intelligence and decision-making logics.

In this context, we consider the intelligent part routing problem in discrete manufacturing plants [3]–[14], which is one of the key aspects of smart manufacturing. Our goal is to devise an *optimization-based*, *scalable*, *feedback control* approach to deal with this problem. The use of an optimization-based technique, as opposed to e.g. heuristic rules, arises from the need to take into account various conflicting objectives (e.g. plant throughput vs. consumed energy) and to efficiently and systematically design the decision-making logic. Scalability is required because of the possibly large number of involved parts and machines in the manufacturing process, resulting in a prohibitively complex optimization problem. Finally, the use of feedback control is motivated by the presence of uncertainty, deriving from high product customization, with the consequent push to increase the plant flexibility, and/or from the unpredictable outcome of certain jobs in the manufacturing process. In particular, in this paper we consider a de-manufacturing plant for electronic boards [13], [14], where individually-controlled modular transportation nodes must move pallets from a load/unload cell to testing, repairing and discarding machines. The routing control problem for this plant features 53 Boolean control inputs to be issued at each time step, in order to optimize an economic cost criterion while satisfying temporal-logic constraints to avoid conflicts among the pallets and plant lockout. One approach to address this problem is a Model Predictive Control (MPC) strategy that directly chooses the control inputs, as proposed and applied in [13]. This approach, based on a "Eulerian" model of the system where the state corresponds to the status of each node, meets the requirements of optimality and feedback and has been also successfully applied to the real plant. However, its scalability is limited: as an example, a prediction horizon of only 5 steps is used in [13] and with the growth of this parameter the resulting mixed-integer problem becomes quickly intractable.

In an effort to retain an optimization-based feedback approach but with better scalability, we recently proposed a new technique that exploits a hierarchical decomposition of the problem [15]. The approach, named here Hierarchical Predictive Routing Control (HPRC), is based on a novel "Lagrangian" description of the system, where the state contains information on the current path of each pallet and of its position along such a path. HPRC then adopts a low-level greedy path following strategy that automatically enforces the temporal-logic constraints, and a high-level predictive controller that selects the path that each pallet shall undertake, possibly shifting paths and/or each pallet's position on a given path, for example to enforce a waiting action or to take a different route. In this way, the optimal control problem results in an unconstrained integer program, which has been solved with a move-blocking strategy in [15], achieving very fast computation also with large prediction horizon values. However, the approach has been tested in [15] only on an academic case. Moreover, the adopted move-blocking strategy does not allow one to flexibly choose a trade-off between suboptimality and computational speed. To solve these issues, this paper presents the following contributions:

1) The HPRC approach is applied to the real de-manufacturing plant considered in [13]. To do so, the greedy path-following approach is expanded to take into account the specific operational constraints of the real plant;
2) A new strategy is proposed, to approximately solve the integer optimization program at the high-level control layer.

Roberto Boffadossi, Lorenzo Fagiano, and Marco Lauricella are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano. Marko Tanaskovic is with Singidunum University, Belgrade. Andrea Cataldo is with the Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing (STIIMA), National Research Council, Milano.

Corresponding author: L. Fagiano, lorenzo.fagiano@polimi.it.

This research was funded by a grant from the Italian Ministry of Foreign Affairs and International Cooperation (MAECI), project "Real-time control and optimization for smart factories and advanced manufacturing".

Based on the concept of "approaching direction", it yields an intuitive balance between optimality and complexity and between a more and a less aggressive plant behavior;
3) A lockout detection routine is introduced, allowing the high-level optimizer to efficiently discard alternatives that lead to plant blockage.
The proposed approach, representing an advanced version of HPRC, is tested here on a high-fidelity simulator of the real plant and compared with the previous HPRC version [15], showing a significant improvement in terms of number of parts that can be handled without incurring in a lockout.

II. Plant description, problem formulation, and background on HPRC

A. De-manufacturing plant description

We consider the automated de-remanufacturing pilot plant in the laboratory of the Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing (STIIMA), National Research Council (CNR), in Milano, see Fig. 1. The plant is designed to test, repair or de-manufacture electronic boards. It consists of four machines, each one executing a different task, connected by a modular transportation line that moves the electronic boards with a pallet handling system.

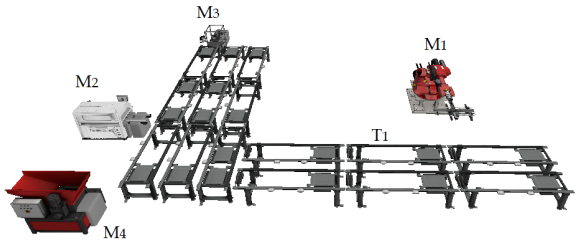

Fig. 1. Overview of the de-remanufacturing plant at the National Research Council in Milano.

Referring to Fig. 1, machine M_1 is the Load/Unload Robot Cell: its purpose is to unload the adjacent pallet (if not empty) and load, from an external buffer, a new electronic board to be tested. Machine M_2 is the Testing Machine, determining whether the boards need to be repaired, dismantled, or if they work correctly. M_3 is the Reworking Machine: if a board is found damaged but repairable, this machine has the task to rework the board to fix the issue detected by M_2. Finally, M_4 is the Discharge Machine: all boards that can not be repaired by M_3 are unloaded from the pallet by this machine, and de-manufactured. Then the empty pallet is sent back to M_1. The handling system is composed of fifteen transport modules T_n, $n = 1, \dots, 15$: they form a flexible modular transport line connecting the machines. The particular structure of the modules allows a part to reach its destination via different possible routes, thus providing the degrees of freedom that enable the optimization of the overall process. With a mathematical abstraction, all the positions that a pallet can occupy inside the transport modules and the machines are modeled as nodes of a directed graph. The graph features $N_n = 36$ nodes, of which $N_m = 4$ form the set of *machine nodes*:

$$\mathcal{M} = \{m : \text{node } m \text{ is a machine}\} \tag{1}$$

while the remaining $N_t = N_n - N_m$ are the *transportation nodes*. All the possible physical movements across the plant are then modeled as directed transitions (53 in total), that correspond to the low-level Boolean control signals. For example, $u_{n_a,n_b} = 1$ commands the relevant transportation modules to move a pallet from node n_a to node n_b. To preserve the plant integrity and avoid inconsistent control signals that may cause an emergency stop, the transitions must be activated only when suitable, i.e. when a pallet is really present in the departure node and its movement is possible depending on the status of the arrival node and on other specific plant constraints, detailed later on in Section III-C. Such conditions give rise to challenging temporal-logic constraints. The nodes and control inputs are presented in Fig. 2. Node 36 is connected to the external buffer of incoming parts.

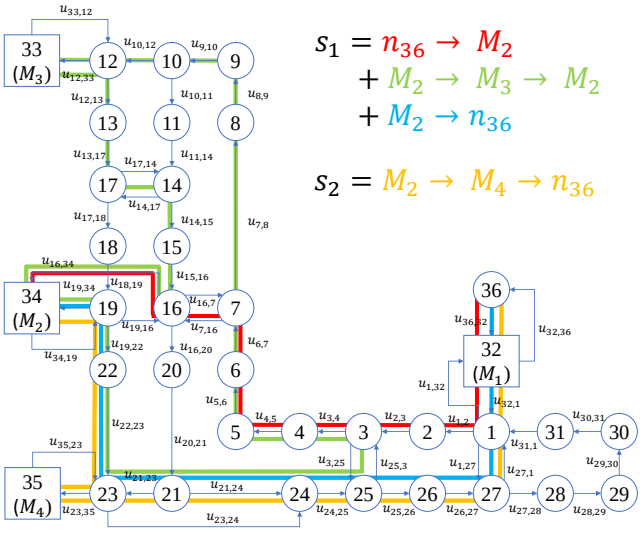

Fig. 2. Directed graph model of the plant, and node sequences s_1, s_2 employed by the HPRC strategy in our simulation tests.

B. Problem formulation

In addition to the presence of a large number of Boolean control signals and of temporal logic constraints, the control problem at hand is challenging due to uncertainty. In fact, the job scheduling of each board is not known a priori. To this regard, Fig. 3 presents the possible paths that each pallet may take. When a new board is loaded on a pallet, its initial target is always the Testing Machine M_2. Then, the test outcome yields one of three possible targets according to the (random) condition of the board: the Reworking Machine M_3, if the board can be repaired; the Discharge Machine M_4, if the board must be de-manufactured, or the Loading/Unloading

Cell M_1, if the board is healthy or repaired successfully. Moreover, when a part is reworked in M_3, a second random variable dictates if the repairing process was successful or not. This is discovered when the part is sent back to M_2 for testing, where there is now an outcome among two alternatives: the board shall either be de-manufactured (if not successfully repaired) or unloaded (if working properly).

The problem we consider in this paper is the design of an optimization-based, scalable, feedback control strategy to manage the described plant in order to maximize its throughput and avoid lockouts, i.e. situations where two pallets are blocked due to conflicting patterns. To solve this problem, we build on our recently proposed HPRC approach, recalled next, and introduce new features, described in Section III.

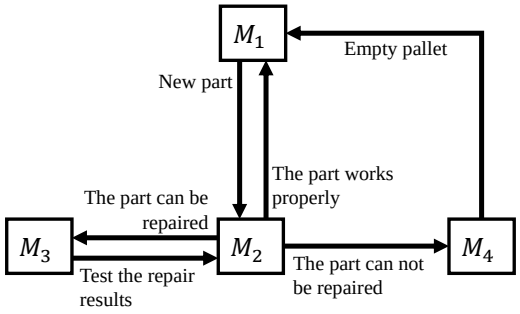

Fig. 3. Possible job sequences for each board, depending on its conditions and the results of the repairing operation.

C. Background on Hierarchical Predictive Routing Control

The HPRC approach proposed in [15] (Fig. 4) features a low-level path following strategy that attempts to move each pallet forward in its assigned path (or *sequence*), and a high-level predictive path allocation layer that optimally assigns sequences to each pallet in order to avoid lockouts and optimize the throughput. We recall the main features of HPRC for the sake of completeness.

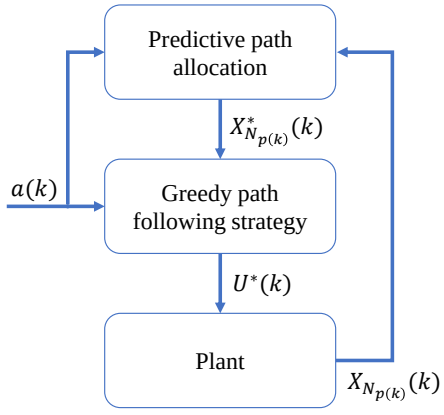

Fig. 4. Hierarchical layout of the HPRC approach.

Lagrangian system model and path following strategy. We denote with $S = \{s_1, \ldots, s_{N_s}\}$ the set of integers s_j, each one corresponding to a predefined sequence. The operator $\mathcal{S}(s_j)$ returns the sequence identified by s_j, which is an ordered set of M_s triplets:

$$\mathcal{S}(s_j) = \left\{ \begin{bmatrix} n_1 \\ g_1 \\ f_1 \end{bmatrix}, \ldots, \begin{bmatrix} n_p \\ g_p \\ f_p \end{bmatrix}, \ldots, \begin{bmatrix} n_{M_s} \\ g_{M_s} \\ f_{M_s} \end{bmatrix} \right\} \quad (2)$$

where, for each $p = 1, \ldots, M_s$, n_p corresponds to a node of the graph, g_p to the target node considered when the pallet is in node n_p and its assigned sequence number is s_i, finally $f_p \in \{0, 1\}$ denotes whether the pair (n_p, g_p) of sequence $\mathcal{S}(s_j)$ is not redundant with respect to other pairs in the same or in other sequences. Typically, f_p is set to 0 for entire parts of a sequence, when these are in common with sub-sequences belonging to other paths. As an example, the last part of sequence s_1 presented in Fig. 2 contains a sub-sequence from node 23 to 27, then node 1, 32 and 36 (target), which is the same as a sub-sequence of s_2: in this case, the triplets of one of these two sub-sequences will feature $f_p = 1$ and those of the other one will have $f_p = 0$. The introduction of the Boolean f_p is another novelty with respect to [15] and allows the optimization layer to discard redundant subsequences in a very efficient way. The choice/generation of sequences is not treated in this paper and is subject of ongoing research. Here, the paths s_1 and s_2 of Fig. 2 have been selected based on the prior knowledge of the intended functioning of the plant.

The index $i = 1, \ldots, N_p(k)$ identifies each one of the $N_p(k)$ parts inside the plant at the discrete time k. The state of a part i is:

$$\boldsymbol{x}_i(k) = \begin{bmatrix} s_i(k) \\ p_i(k) \\ t_i(k) \end{bmatrix} \quad (3)$$

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

$$t_i(k) = k - \underline{k}_i \quad (4)$$

The states of all parts are collected in a single vector representing the overall Lagrangian state of the plant:

$$X_{N_p(k)}(k) = [\boldsymbol{x}_1(k)^T, \ldots, \boldsymbol{x}_{N_p}(k)^T]^T \in \mathbb{N}^{3N_p(k)} \quad (5)$$

where $.^T$ denotes the matrix transpose operation.

The Lagrangian state is employed by a low-level Greedy Path Following Strategy to compute the Boolean control signals $u_{n_a,n_b}(k) \in \{0, 1\}$ sent to the various transport modules at time k, collected in a vector $\boldsymbol{U}(k) \in \{0, 1\}^{N_u}$ ($N_u = 53$ in our case). In practice, such a path following strategy attempts to move forward each part in the assigned sequence and detects if any conflicts occur. In the original approach of [15], these included, for example, the attempt of more than one part to move to the same node and the attempt of a part to move to an occupied node (e.g. if the latter is holding the part in place). If a conflict is detected, the path following strategy proceeds to remove it by a set

of predefined priority rules, see [15] for details. Temporal logic constraints due to the time required for a machine node to finish its job are also satisfied, through a suitable sequence generation, i.e. by repeating the machine node consecutively in the sequence for at least L_m positions, where $L_m \geq 1$ is the number of time steps needed by machine m to end the job. The close-loop plant under the greedy path-following strategy can be represented through a discrete-time dynamical model:

$$X_{N_p(k+1)}(k+1) = f_{(N_p(k+1),N_p(k))}(X_{N_p(k)}(k), a(k)) \quad (6)$$

where the exogenous Boolean variable $a(k)$ (also shown in Fig. 4) represents the availability of a new part to be loaded on the transport system, which in the considered plant can happen in node 36 (see Fig. 2). We refer to (6) as the *Lagrangian plant model*, because it propagates the trajectories of the involved parts, in analogy with the Lagrangian description in fluid dynamics. Note that the number of parts $N_p(k)$ (hence the state dimension) can change over time depending on new arrivals and on parts that are unloaded in M_1 or discarded in M_4.

High-level Model Predictive Path Allocation.

At each time step, the high-level control layer solves an unconstrained integer Finite Horizon Optimal Control Problem (FHOCP) to optimally (re-)assign to each part the sequence to be followed and their position in such a sequence. The FHOCP is unconstrained because it considers the Lagrangian model (6) to predict the parts' trajectories, so that the low-level path following strategy automatically enforces the operational constraints. In practice, such a sequence re-assignment corresponds to directly modifying the first two elements of the state (3) of each part i, selecting them among the set $\mathcal{X}_i(k)$ of *compatible states*, $i = 1, \ldots, N_p(k)$, computed as:

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

$$\mathcal{X}_i(k) = \left\{ \begin{array}{l} (s,p) \in S \times \mathbb{N} : \\ \mathcal{S}(s)^{(1,p)} = \mathcal{S}(s_i(k))^{(1,p_i(k))} \\ \wedge \mathcal{S}(s)^{(2,p)} = \mathcal{S}(s_i(k))^{(2,p_i(k))} \\ \wedge \mathcal{S}(s)^{(3,p)} = 1 \end{array} \right\} \quad (7a)$$

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

$$\mathcal{X}_i(k) = \left\{ \begin{array}{l} (s,p) \in S \times \mathbb{N} : \\ \mathcal{S}(s)^{(1,p-j)} = \mathcal{S}(s_i(k))^{(1,p_i(k)-j)}, \\ j = 0, \ldots, k - k_{\mathcal{S}(s_i(k))^{(1,p_i(k))}} \\ \wedge \mathcal{S}(s)^{(2,p)} = \mathcal{S}(s_i(k))^{(2,p_i(k))} \\ \wedge \mathcal{S}(s)^{(3,p)} = 1 \end{array} \right\} \quad (7b)$$

where the operators $\mathcal{S}(s)^{(1,p)}$, $\mathcal{S}(s))^{(2,p)}$ and $\mathcal{S}(s)^{(3,p)}$ return the first, second, and third value, respectively, of the p-th element of sequence $\mathcal{S}(s)$, and $k_{\mathcal{S}(s_i(k))^{(1,p_i(k))}}$ is the time step when the i-th part has started the operation in machine $m = \mathcal{S}(s_i(k))^{(1,p_i(k))}$. In practice, according to (7) the set $\mathcal{X}_i(k)$ contains all sequence-position pairs that **a)** correspond to the current node where part i is, **b)** indicate the same target and **c)** are such that $f_{p_i} = 1$ (to avoid testing redundant sequences, see (2) and the comments thereafter). Additionally, **d)** when part i is in a machine node (i.e., $\mathcal{S}(s_i(k))^{(1,p_i(k))} \in \mathcal{M}$), the compatible states must also preserve the correct number of remaining time steps of the corresponding job.

The FHOCP solved at each time step by the predictive path allocation then reads (see [15] for the whole receding-horizon implementation):

$$\min_{(\sigma_i,\pi_i),\, i=1,\ldots,N_p(k)} \sum_{o=0}^{N} \ell_{N_p(o|k)} \left(X_{N_p(o|k)}(o|k) \right) \quad (8a)$$

subject to

$$\boldsymbol{x}_i(0|k) = [\sigma_i, \pi_i, t_i(k)]^T, \; i = 1, \ldots, N_p(k) \quad (8b)$$

$$X_{N_p(0|k)}(0|k) = \left[\boldsymbol{x}_1(0|k)^T, \ldots, \boldsymbol{x}_{N_p(k)}(0|k)^T \right]^T \quad (8c)$$

$$X_{N_p(o+1|k)}(o+1|k) = f_{(N_p(o+1|k),N_p(o|k))}(X_{N_p(o|k)}(k), a(o|k)), \quad o = 0, \ldots, N-1 \quad (8d)$$

$$(\sigma_i, \pi_i) \in \mathcal{X}_i(k), \; i = 1, \ldots, N_p(k) \quad (8e)$$

Where $X_{N_p(o|k)}(o|k)$ denotes the prediction of the Lagrangian state computed at time k and pertaining to time $k+o$, and $\ell_{N_p(o|k)} \left(X_{N_p(o|k)}(o|k) \right)$ is a user-defined stage cost function (e.g., the total number of remaining steps in each part's sequence).

III. Advanced HPRC and application to the de-manufacturing plant

The FHOCP (8) of the original HPRC corresponds to a move-blocking strategy, since the optimizer chooses the state of each part only at $o = 0$ (see (8c)) and the corresponding prediction is then propagated forward for the whole horizon N without further sequence re-assignments. Albeit computationally efficient, this approach yields suboptimal performance and may not be able to prevent lockouts in a complex plant. Moreover, the real-world system considered in this work features specific routing constraints that were not taken into account in the original path following strategy of [15]. In this section, we address these issues by introducing several novelties, described next.

A. Search tree exploration with Approaching Direction method

A first improvement we present aims to increase the degrees of freedom in the optimization program, by adopting a search tree structure instead of the move blocking approach of (8). In this way, the optimizer can re-assign sequences to each part at each step of the prediction. This makes it possible to combine few simple precomputed sequences to obtain a large number of part trajectories. For example, with this approach it is sufficient to repeat a node only twice within a sequence to allow a part to wait in that node for a large number of steps, e.g. until other parts have left a congested section of the plant. The nodes repeated twice in a sequence are named *waiting nodes*, and they are chosen according to their strategic position, typically in proximity of a machine.

On the other hand, extensive exploration of the search tree

is a hard problem generally computationally prohibitive, resulting in very poor scalability. We therefore propose a particular approach to partially explore the search tree exploiting our insight of the underlying control engineering problem. The idea is to order the various branches of the search tree on the basis of how aggressive or cautious the corresponding control moves are. To illustrate the concept, we present a simple example with seven nodes, shown in Fig. 5. There is a main sequence $S(s_1)$ with four waiting nodes $(1, 3, 5, 6$, circled in red). Let us assume that at time k there are $N_p(k) = 4$ parts, respectively at nodes $1, 3, 4$ and 6 (colored in orange in the figure).

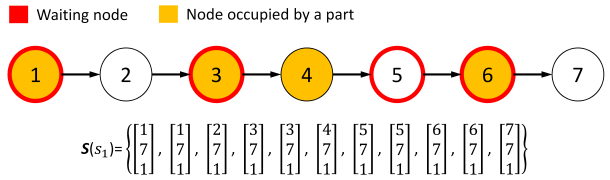

Fig. 5. Illustrative example used to describe the Approaching Direction method. Nodes with a part are colored in orange, and waiting nodes are circled in red.

Assuming that part 1 is in node 1, part 2 in node 3, part 3 in node 4, and part 4 in node 6, the sets of compatible states are: $\mathcal{X}_1(k) = \{(s_1, 1), (s_1, 2)\}$, $\mathcal{X}_2(k) = \{(s_1, 4), (s_1, 5)\}$, $\mathcal{X}_3(k) = \{(s_1, 6)\}$, $\mathcal{X}_4(k) = \{(s_1, 9), (s_1, 10)\}$. Note that all parts in a waiting node have two alternatives at $k + 1$: either to hold their position or to move forward. We denote the Lagrangian state of a part i in a waiting node and in the "hold" position as $\boldsymbol{x}_i(k)[1]$, and that one in the "move" position as $\boldsymbol{x}_i(k)[2]$. Thus, for example we have $\boldsymbol{x}_2(k)[1] = [s_1\ 4\ t_2(k)]^T$ and $\boldsymbol{x}_2(k)[2] = [s_1\ 5\ t_2(k)]^T$.
Now, we can build all possible compatible Lagrangian states $X_4(o|k)$ (recall that the sub-index $_4$ refers to the number of parts, compare (5)) at time k, which the HPRC strategy can test (see (8c)), corresponding to the predicted step $o = 0$. Their number is $2^{N_w(k)}$, where $N_w(k)$ is the number of occupied waiting nodes. Moreover, we can order these Lagrangian states from the most cautious (i.e., trying to hold parts in place as much as possible) to the most aggressive one:

$$
\begin{aligned}
X_4^1(0|k) &= [\boldsymbol{x}_1(k)[1]^T, \boldsymbol{x}_2(k)[1]^T, \boldsymbol{x}_3(k)[1]^T, \boldsymbol{x}_4(k)[1]^T]^T \\
X_4^2(0|k) &= [\boldsymbol{x}_1(k)[2]^T, \boldsymbol{x}_2(k)[1]^T, \boldsymbol{x}_3(k)[1]^T, \boldsymbol{x}_4(k)[1]^T]^T \\
X_4^3(0|k) &= [\boldsymbol{x}_1(k)[1]^T, \boldsymbol{x}_2(k)[2]^T, \boldsymbol{x}_3(k)[1]^T, \boldsymbol{x}_4(k)[1]^T]^T \\
X_4^4(0|k) &= [\boldsymbol{x}_1(k)[2]^T, \boldsymbol{x}_2(k)[2]^T, \boldsymbol{x}_3(k)[1]^T, \boldsymbol{x}_4(k)[1]^T]^T \\
X_4^5(0|k) &= [\boldsymbol{x}_1(k)[1]^T, \boldsymbol{x}_2(k)[1]^T, \boldsymbol{x}_3(k)[1]^T, \boldsymbol{x}_4(k)[2]^T]^T \\
X_4^6(0|k) &= [\boldsymbol{x}_1(k)[2]^T, \boldsymbol{x}_2(k)[1]^T, \boldsymbol{x}_3(k)[1]^T, \boldsymbol{x}_4(k)[2]^T]^T \\
X_4^7(0|k) &= [\boldsymbol{x}_1(k)[1]^T, \boldsymbol{x}_2(k)[2]^T, \boldsymbol{x}_3(k)[1]^T, \boldsymbol{x}_4(k)[2]^T]^T \\
X_4^8(0|k) &= [\boldsymbol{x}_1(k)[2]^T, \boldsymbol{x}_2(k)[2]^T, \boldsymbol{x}_3(k)[1]^T, \boldsymbol{x}_4(k)[2]^T]^T
\end{aligned}
$$

Then, we can program the search tree exploration routine to test these sequences following different possible strategies in terms of aggressiveness vs. cautiousness, i.e. with different *Approaching Directions* (AD). In our research, we tested the following ADs:
AD1: From the most aggressive to the most cautious: $X_4^8(0|k), X_4^7(0|k), \ldots, X_4^1(0|k)$.
AD2: From the most cautious to the most aggressive: $X_4^1(0|k), X_4^2(0|k), \ldots, X_4^8(0|k)$.
AD3: Inward alternation: $X_4^1(0|k), X_4^8(0|k), X_4^2(0|k), X_4^7(0|k), \ldots, X_4^5(0|k)$.
AD4: Outward alternation: $X_4^5(0|k), X_4^4(0|k), X_4^6(0|k), X_4^3(0|k), \ldots, X_4^1(0|k)$.
AD5: Alternation directed to the most aggressive: $X_4^5(0|k), X_4^1(0|k), X_4^6(0|k), X_4^2(0|k), \ldots, X_4^4(0|k)$.

After choosing an AD, the search tree exploration starts by selecting the corresponding first compatible state and the plant prediction is advanced by one step. Let us assume for example that **AD1** is used: state $X_4^8(0|k)$ is thus selected and propagated forward to $o = 1$. Then, the resulting predicted compatible states $X_4^j(1|k)$, $j = 1, \ldots, 2^{N_w(1|k)}$ are computed, and the same Approaching Direction (**AD1** in this example) is applied again. The process is repeated until either the prediction horizon is reached ($o = N$), or a lockout is predicted (see Section III-B). If $o = N$ is reached, the branch corresponding to the chosen compatible state at $o = 0$ (i.e., $X_4^8(0|k)$ in this example) is marked "explored", and a new predicted trajectory is computed considering the next state compatible with the current one ($X_4^7(0|k)$). If a lockout is reached, the optimizer backtracks the last moves and tries the available alternative compatible states, again using the same AD, until either reaching $o = N$ or backtracking down to $o = 0$, in which case that branch is marked "unfeasible". The exploration ends when all branches pertaining to states compatible with the current one (i.e. $X_4^8(0|k)$ to $X_4^1(0|k)$) are either explored or unfeasible. At that point, the branch with smallest cost is selected. The whole process is repeated in a receding horizon implementation as in [15], not reported here for the sake of brevity. Note that this procedure guarantees that a lockout-free sequence is always found, if it exists. The proposed AD method results in a good trade-off between extensive search-tree exploration and computational time, as it exploits prior knowledge of the practical problem underlying the mathematical abstraction of the search tree. For example, when the handling system is very congested or the prediction horizon is very long, a more cautious AD is more likely to avoid a lockout. On the other hand, with a less congested plant the more aggressive ADs provide usually higher throughput.

B. Lockout detection

A second improvement we introduce is aimed to speed-up the high-level optimization process by detecting lockouts and stopping the corresponding prediction. Then, the optimizer can backtrack a predefined number of steps to try different moves. We identified two different types of lockouts: the *local lockout* and the *theoretical lockout*.

1) Local lockout: it occurs when two or more parts are blocking each other. Indeed, when the motion of a part along its current sequence is inhibited, there is always another part

with a higher priority blocking it. The local lockout arises when a part is indirectly inhibiting its own motion. This necessarily happens at the same time for at least two parts, and a chain of conflicts can also take place, starting and ending with the same part. The condition to be checked to detect a local lockout is, for $i, j = 1 \ldots, N_p(k)$:

$$\exists (i,j) : \mathcal{Q}(i)^j = i \tag{9}$$

where the operator $\mathcal{Q}(i)$ returns the index q_i of the part that is blocking part i. The integer j indicates how many times the operator is iterated (e.g, $\mathcal{Q}(i)^3 = \mathcal{Q}(\mathcal{Q}(\mathcal{Q}(i)))$). According to (9), a local lockout occurs when the same part index is detected going backwards along the chain of conflicts. This condition is thus checked for each part whose motion is inhibited at each predicted time step. If no lockout is present, the repeated application of operator $\mathcal{Q}$ returns $\emptyset$.

2) Theoretical lockout: it occurs when the Lagrangian state of the whole system does not change over time, i.e. all the parts remain in the same node and possible parts in a machine node do not advance along their sequences. This can happen, for example, if all parts are in waiting nodes and the most cautious AD is being explored. To detect this situation, we need to compare the one-step-ahead prediction of the state with the current one. Then, three conditions identify a theoretical lockout:

$$N_p(k) = N_p(k+1) = N_p \tag{10}$$

$$\begin{aligned} \mathcal{S}(s_i(k))^{(1,p_i(k))} &= \mathcal{S}(s_i(k+1))^{(1,p_i(k+1))} \\ \wedge \mathcal{S}(s_i(k))^{(1,p_i(k))} &\notin \mathcal{M},\ \forall i = 1 \ldots, N_p \end{aligned} \tag{11}$$

$$\begin{aligned} \mathcal{K}(s_j(k), p_j(k)) &= \mathcal{K}(s_j(k+1), p_j(k+1)), \\ \forall j : \mathcal{S}(s_j(k))^{(1,p_j(k))} &\in \mathcal{M}, j = 1 \ldots, N_p \end{aligned} \tag{12}$$

Where the operator $\mathcal{K}(s_j, p_j)$ returns the number of positions prior to p_j in the sequence $\mathcal{S}(s_j)$, with a node index identical to the current node $\mathcal{S}(s_j)^{(1,p_j)} = m$ (i.e., it indicates how many steps the part j has been inside the machine node m before the position p_j in sequence $\mathcal{S}(s_j)$). Condition (10) checks whether the number of parts is the same at k and $k+1$. Condition (11) checks if the parts in transportation nodes are held in position. Finally, condition (12) verifies if the parts in machine nodes are not advancing in their working sequence.

C. Plant specific constraints and resulting subroutines

As anticipated in Section II-C, the HPRC approach of [15] takes into account a number of common temporal-logic constraints that are present in any discrete manufacturing plant. The specific system considered here features additional ones, grouped in three classes. The first two depend on the configuration and strucutral limits of the transport modules. The third pertains to the possible clash between two pallets when a transition traverses an occupied node. To comply with these new constraints, we added subroutines to the greedy path-following strategy. With the same conceptual approach of [15], in each subroutine the state is first propagated forward, then possible conflicts/constraint violations are detected and solved by backtracking according to predefined priority rules.

1) Constrained Nodes: (CNs) the transitions involving these specific nodes must follow certain limitations, in particular:

- If there is a part moving out of a CN, it is forbidden for another part to enter in the CN at the same time step;
- A pair of nodes (n_{out}, n_{in}) can be assigned to a CN, the first one being the destination of a part leaving the CN and the second one being the current node of a part entering the CN. Then the transitions from the CN to n_{out} and n_{in} to CN can occur simultaneously (not respecting the first rule).

Referring to Fig. 2, an example of CN that must obey the first rule without exceptions is node 5, while in node 3 only the transitions $u_{2,3}$ and $u_{3,4}$ can occur simultaneously. Subroutine 1 checks, for each part i, whether one of these limitations must be enforced, and corrects accordingly the state $\hat{\boldsymbol{x}}_i(k+1)$ at the next time step. For simplicity, we present the routine for just one CN, denoted as n_{CN}.

Subroutine 1

1. For all $i = 1, \ldots, N_p(k)$, compute the current and next nodes, $n_{a_i} = \mathcal{S}(s_i(k))^{(1,p_i(k))}$ and $n_{n_i} = \mathcal{S}(s_i(k))^{(1,\hat{p}_i(k+1))}$ respectively;

2 For each part i such that $n_{n_i} = n_{CN}$ and $\exists j : n_{a_j} = n_{CN}$:
`If` $n_{n_j} = n_{out}$ and $n_{a_i} = n_{in}$ update normally its state;
`Else`, correct its state as:

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

2) Forbidden pairs of transitions: these are pairs of inputs that can not take value 1 at the same time, even if they have no node in common. For example, in the considered plant only one between $u_{13,17}$ and $u_{12,33}$ can be set to 1 at each step, because the physical handling system is shared among the four nodes 12, 13, 17, 33. Subroutine 2 checks when two forbidden transitions would be triggered at the same time and computes the one with the highest priority. To this end, the same ranking already adopted in the path following algorithm of [15] is used. Again, for simplicity we consider in the abstract just one of such transition pairs and denote the two involved inputs as u_{n_a,n_b}, u_{n_c,n_d}.

Subroutine 2

For each $i, j = 1, \ldots, N_p(k)$ such that $\mathcal{S}(s_{i(k)})^{(1,p_i(k))} = n_a \wedge \mathcal{S}(s_{i(k)})^{(1,p_i(k+1))} = n_b \wedge \mathcal{S}(s_{j(k)})^{(1,p_j(k))} = n_c \wedge \mathcal{S}(s_{j(k)})^{(1,p_j(k+1))} = n_d$, compute the index $\underline{i}$ of the part with the *lowest* priority:
`If` $r_i(k) < r_j(k)$ `then` $\underline{i} = j$
`Elseif` $r_i(k) > r_j(k)$ `then` $\underline{i} = i$
`Elseif` $r_i(k) = r_j(k)$ `then` $\underline{i} = \arg \min_{y \in \{i,j\}} t_y(k)$
where $r_\ell(k)$ is the number of remaining nodes for the

generic part ℓ to complete its sequence. Then, correct:

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

3) Transitions traversing a node: in the considered plant there are two transitions, $u_{16,34}$ and $u_{23,24}$, that physically traverse an intermediate node, 19 and 21 respectively, and thus can not be actuated if such node is occupied. To describe Subroutine 3, which deals with this constraint, we generally denote one such transition as u_{n_e,n_f} and the related intermediate node as n_g .

Subroutine 3
For each $i, j = 1, \ldots, N_p(k)$ such that $\mathcal{S}(s_i(k))^{(1,p_i(k))} = n_e \wedge \mathcal{S}(s_i(k))^{(1,p_i(k+1))} = n_f \wedge \mathcal{S}(s_j(k))^{(1,p_j(k))} = n_g$, hold in position part i:

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

In practice, Subroutine 3 blocks any one of these transitions if the corresponding intermediate node is occupied.

IV. Simulation results

The model of the plant and the controller have been implemented and tested via Matlab using a laptop with 16GB RAM and an Intel Core i7-7700HQ at 2.8 GHz. The machine working times L_m are $L_1 = 1;\ L_2 = 5;\ L_3 = 4;\ L_4 = 3$. The first time a board is processed by the testing machine M_1, the probabilities associated to the part conditions are:

- $P_{M_4} = 0.4$: the board can not be repaired;
- $P_{M_1} = 0.2$: the board works properly;
- $P_{M_3} = 0.4$: a repair attempt can be carried out.

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

- $P_{M_4} = 0.4$: the board is broken;
- $P_{M_1} = 0.6$: the part works properly.

In all our tests with the advanced HPRC approach we used the two sequences s_1 and s_2 shown in Fig. 2. The chosen waiting nodes are $\{2, 10, 15, 22, 27\}$. In each sequence, each machine node m is repeated L_m times according to the corresponding required working time. The main sequence s_1 corresponds to the scheduling of a damaged board that is fixed by the reworking machine and then unloaded after the second test in M_2. The additional sequence s_2 corresponds to the trajectory of a board that has been processed by M_2 and found damaged and not repairable anymore, thus needs to reach M_4 and the pallet must then return empty to the Load/Unload robot Cell, M_1. Along the sequence s_2 the Boolean variable f_p has been set to zero in the sub-sequence $\{23 - 27, 1, 32, 36\}$ to avoid redundancy in path evaluation (see Section II-C). In all simulations, the exogenous signal $a(k)$ is always considered as known and equal to one. In this way, when the maximum number of pallet is reached, a new pallet can be loaded only after another one has been unloaded. The maximum number of parts (i.e., the number of pallets in the plant) is $N_{p,max}$.
Since the goal is to maximize the throughput, the stage cost is chosen as:

$$\ell_{N_p(o|k)} = \lambda_1 \sum_{i=1}^{N_p(o|k)} r_i(o|k) + \lambda_2 \sum_{i=1}^{N_p(o|k)} t_i - \lambda_3 u_{32,36}(o|k) \tag{13}$$

where λ_1, λ_2, $\lambda_3 > 0$ are weighting coefficients. The first term of the stage cost is the sum of the remaining nodes for each part and ensures that the parts are pushed forward along their sequences. The second is the sum of the time counter values t_i, and penalizes holding a part in the plant for too long. The last, negative term is to force an unloaded pallet to reach node 36 to take another part ($\lambda_3 >> \lambda_1, \lambda_2$).
The initial condition is with in one pallet in the robot cell (node 32) and the simulation is interrupted at $\bar{k} = 2000$. As performance metrics, we consider the total number of processed parts in the simulation, N_f, the average computational time needed to compute the control action, C_t, and the peak time observed during the simulation, C_{peak}.

In the first test, different ADs have been compared, see Table I. The most aggressive one (**AD1**, see Section III-A) achieves the smallest computational times, while all directions obtain a similar throughput. This can be expected, since all strategies eventually explore all the compatible states at each time step, and then the optimization is repeated at the following one, closing the loop. The observed differences in N_f are due to slightly different transients from the initial condition to full regime. Furthermore, we evaluated the performance obtained by stopping the search-tree exploration as soon as a feasible sequence is found ("No optimization" in Table I): we note that in this case the performance depends significantly on the chosen approaching direction. When there are few pallets in the plant ($N_{p,max} \leq 5$) and **AD1** is adopted, the number of the finished parts is identical with or without optimization.

TABLE I
Comparison among different Approaching Directions with $N_{p,max} = 10$ pallets and $N = 30$. The results with an approach where the first feasible sequence found is used (denoted with "No optimization") are presented, too.

	$C_t\ [s]$	$C_{peak}\ [s]$	N_f
AD1	0,499	3,596	211
AD2	1,462	14,084	204
AD3	1,198	9,097	206
AD4	0,665	6,105	208
AD5	0,667	4,320	205
No optimization			
AD1	0,192	2,287	203
AD2	0,547	6,167	105

We carried out a second test to verify the change in performance due to the uncertainty: the testing machine results

are first assumed to be known a-priori ("Deterministic" case), then they are only estimated, by generating predicted random samples that are different from those that determine the part conditions in the plant simulation ("Uncertain" case). In Table II it can be noticed that for the uncertain case the performance deteriorate only slightly, except when many pallets are present, and the controller is still able to avoid the lockout. This result depends both on feedback and on the employed sequences: in fact, the initial path segment after M_2 is identical independently from the new target assigned to the part, thus allowing the controller to correct the control action in the following steps thanks to the receding horizon strategy.

TABLE II

COMPARISON BETWEEN DETERMINISTIC AND UNCERTAIN CASES, WITH $N = 30$ AND **AD1**

Deterministic			
	$C_t\ [s]$	$C_{peak}\ [s]$	N_f
$N_{p,max} = 3$ pallets	0,075	0,636	149
$N_{p,max} = 5$ pallets	0,108	1,090	211
$N_{p,max} = 10$ pallets	0,499	3,596	211
$N_{p,max} = 13$ pallets	1,368	34,177	216
Uncertain			
$N_{p,max} = 3$ pallets	0,075	0,768	149
$N_{p,max} = 5$ pallets	0,189	1,689	206
$N_{p,max} = 10$ pallets	0,462	3,313	201
$N_{p,max} = 13$ pallets	2,000	56,285	200

Finally, we ran several tests with the move blocking strategy of [15], instead of the Approaching Direction method proposed here. To obtain a fair comparison, we had to modify the sequences by adding more waiting nodes, thus enabling the move blocking approach to also generate predictions where a part can wait more than one time step in the same node, e.g. to wait for a machine to finish an ongoing job. Specifically, node 10 is repeated L_3 times and node 22 is repeated L_4 times. The results are reported in Table III. For $N_{p,max} > 5$ the move blocking approach is not able to avoid plant lockout, while for lower number of pallets the results are comparable to those of the Approaching Direction method. This shows that the new method is able to handle much larger numbers of parts. Moreover, as pointed out above, with the move blocking approach the sequence choice is more critical and needs to be carefully thought and/or tested with trial and error procedures, since the optimization routine has fewer degrees of freedom.

TABLE III

RESULTS OBTAINED WITH THE MOVE BLOCKING APPROACH OF [15] AND $N = 30$

Deterministic			
	$C_t\ [s]$	$C_{peak}\ [s]$	N_f
$N_{p,max} = 3$ pallets	0,082	1,604	147
$N_{p,max} = 5$ pallets	0,1233	1,6584	213
$N_{p,max} > 5$ pallets	/	/	/
Non-deterministic			
$N_{p,max} = 3$ pallets	0,079	0,817	148
$N_{p,max} = 5$ pallets	0,118	1,660	205
$N_{p,max} > 5$ pallets	/	/	/

V. CONCLUSIONS

We presented the application of a novel HPRC approach to the routing control problem of a real de-manufacturing plant. Contextually, we introduced several improvements to the approach, including a novel search tree exploration method, lockout detection routines, and plant-specific handling constraints. Simulation results show very good performance and small computational times even with high number of pallets and long prediction horizon values. Current research is devoted on the theoretical side to the derivation of recursive feasibility conditions, particularly accounting for the plant uncertainty, and on the experimental side to the test of the approach on the real de-manufacturing plant.

REFERENCES

[1] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, "Intelligent manufacturing in the context of industry 4.0: A review," *Engineering*, vol. 3, no. 5, pp. 616 – 630, 2017.

[2] B.-H. Li, B.-C. Hou, W.-T. Yu, X.-B. Lu, and C.-W. Yang, "Applications of artificial intelligence in intelligent manufacturing: a review," *Frontiers of Information Technology and Electronic Engineering*, vol. 18, no. 1, pp. 86–96, 2017.

[3] Y. P. Gupta, M. C. Gupta, and C. R. Bector, "A review of scheduling rules in flexible manufacturing systems," *International Journal of Computer Integrated Manufacturing*, vol. 2, no. 6, pp. 356–377, 1989.

[4] C. Saygin, F. Chen, and J. Singh, "Real-time manipulation of alternative routeings in flexible manufacturing systems: A simulation study," *The International Journal of Advanced Manufacturing Technology*, vol. 18, pp. 755–763, 2001.

[5] R. Bucki, B. Chramcov, and P. Suchánek, "Heuristic algorithms for manufacturing and replacement strategies of the production system," *Journal of Universal Computer Science*, vol. 21, no. 4, pp. 503–525, apr 2015.

[6] M. Souier, A. Hassam, and Z. Sari, *Meta-heuristics for Real-time Routing Selection in Flexible Manufacturing Systems*. London: Springer London, 2010, pp. 221–248.

[7] S. K. Das and P. Nagendra, "Selection of routes in a flexible manufacturing facility," *International Journal of Production Economics*, vol. 48, no. 3, pp. 237 – 247, 1997.

[8] K. Kouiss, H. Pierreval, and N. Mebarki, "Using multi-agent architecture in fms for dynamic scheduling," *Journal of Intelligent Manufacturing*, vol. 8, pp. 41–47, 1997.

[9] C. Peng and F. Chen, "Real-time control and scheduling of flexible manufacturing systems: An ordinal optimisation based approach," *The International Journal of Advanced Manufacturing Technology*, vol. 14, pp. 775–786, 1998.

[10] A. R. Moro, H. Yu, and G. Kelleher, "Hybrid heuristic search for the scheduling of flexible manufacturing systems using petri nets," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 2, pp. 240–245, 2002.

[11] F. D. Vargas-Villamil and D. E. Rivera, "Multilayer optimization and scheduling using model predictive control: application to reentrant semiconductor manufacturing lines," *Computers & Chemical Engineering*, vol. 24, no. 8, pp. 2009 – 2021, 2000.

[12] A. Cataldo, A. Perizzato, and R. Scattolini, "Production scheduling of parallel machines with model predictive control," *Control Engineering Practice*, vol. 42, pp. 28 – 40, 2015.

[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, Sep. 2016.

[14] A. Cataldo, M. Morescalchi, and R. Scattolini, "Fault tolerant model predictive control of a de-manufacturing plant," *The International Journal of Advanced Manufacturing Technology*, vol. 9, no. 12, pp. 4803–4812, 2019.

[15] L. Fagiano, M. Tanaskovic, L. C. Mallitasig, A. Cataldo, and R. Scattolini, "Hierarchical routing control in discrete manufacturing plants via model predictive path allocation and greedy path following," in *Decision and Control (CDC), 2020 IEEE Conference on*, 2020, preprint available on arXiv: https://arxiv.org/abs/2011.04341.