RRPDG: A Graph Model to Enable AI-Based Production Reconfiguration and Optimization

Sebastiano Gaiardelli, *Student Member, IEEE*, Michele Lora, *Member, IEEE*, Stefano Spellini, and Franco Fummi, *Member, IEEE*

***Abstract*—This article introduces the regionalized resource process dependence graphs (RRPDGs): a manufacturing processes representation inspired by the regionalized value state dependence graphs traditionally used in software compilers. An RRPDG is an ordered sequence of nodes, each characterized by stereotyped input and output parameters, encapsulating a transformation of the process state (e.g., a manufacturing operation). RRPDG allow defining complex transformations by composing a set of nodes (i.e., regions), hiding the internal details. Then, RRPDGs are used to automatically reasoning over dynamic reconfiguration and process optimization: an instance of the A-star search algorithm is used to search for possible transformations while pursuing an optimization function. The rules defined in this article over RRPDG models enforce the transformations' correctness. We use RRPDGs to model a real production system while the transformation rules are applied to optimize the system's processes. The proposed representation reduced the search complexity in each experiment, allowing to reach an optimal solution also in the case for which classical approaches were unable to complete before reaching the timeout. In all the experiments, the cost of the solution produced by using the regionalized representation is minor than the the solution produced by using the classical representation.**

***Index Terms*—Modeling, smart manufacturing, process control in manufacturing automation.**

I. Introduction

AMONG the technical innovations of modern manufacturing, service-oriented architectures for manufacturing systems stand out to make production processes more flexible and efficient: functionalities are organized in "services" [1], [2] and distributed through manufacturing components and agents. A *manufacturing service* is a *minimal manufacturing operation* executed from a specific piece of equipment (e.g., the pick and place operation carried out by a robotic arm). Each service is exposed to other pieces of software (e.g., supervisory control) through a communication infrastructure and a compatible software architecture [3]. To specify the production process, a recipe can be represented as an ordered composition of manufacturing services [4]. To meet the flexibility of modern market trends, the complexity of the functionality implemented by *services* is constantly increasing [5], [6]. Therefore, a lot of research effort has been spent in the past two decades on developing modeling [7] and system verification [8] approaches.

To support the reconfiguration of modern manufacturing systems, the first condition to meet is a production process modeling strategy: it should provide the ability to represent process dependencies and the input/output constraints. Furthermore, a set of models' manipulation rules is needed to guide the process optimizations (e.g., makespan reduction) while guaranteeing that the same functionality is preserved.

To cope with such requirements, this article proposes regionalized resource process dependence graphs (RRPDGs): a region-based production processes representation inspired by regionalized value state dependence graphs (RVSDGs). The representation exploits the concept of "region," which is the implementation of a sequence of production services (i.e., nodes). Furthermore, regions may be composed of other subregions, to hierarchically structure a complex service.

We define a set of formal rules over RRPDGs to structurally manipulate models of production processes. The defined rules guarantee that a modified production process carries equivalent functionality with respect to the original model. Then, the set of transformation rules and the region-based process representation can be exploited to guide reconfiguration. Fig. 1 summarizes the entire contribution with an example: a production system made of four machines, i.e., M_a, M_b, M_c, and M_d, and each machine implements a set of services, i.e., a set of base manufacturing operations. For instance, M_a implements the services S_1 and S_2. The system carries on the production according to the production recipe depicted above, and represented by a RRPDG. Let us suppose M_b fails, triggering the reconfiguration of the production as the service S_3 becomes unavailable and thus making the red region in the figure unfeasible. The optimization

Manuscript received 11 April 2023; revised 27 September 2023; accepted 1 January 2024. Date of publication 25 January 2024; date of current version 4 April 2024. This work was supported in part by the European Union's Horizon 2020 research and innovation program through the Marie Skłodowska-Curie grant under Grant 894237, in part by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) "Missione 4 Componente 2, Investimento 1.5" D.D. 1058 23/06/2022) through the PNRR research activities of the consortium iNEST under Grant ECS_00000043, and in part by the PRIN 2022T7YSHJ SMART-IC-Next Generation EU project. Paper no. TII-23-1274. *(Corresponding author: Sebastiano Gaiardelli.)*

Sebastiano Gaiardelli, Michele Lora, and Franco Fummi are with the Department of Engineering for Innovation Medicine (DIMI), University of Verona, 37134 Verona, Italy (e-mail: sebastiano.gaiardelli@univr.it; michele.lora@univr.it; franco.fummi@univr.it).

Stefano Spellini is with the FACTORYAL S.r.l., 37057 San Giovanni Lupatoto, Italy (e-mail: stefano.spellini@factoryal.it).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2024.3352645.

Digital Object Identifier 10.1109/TII.2024.3352645

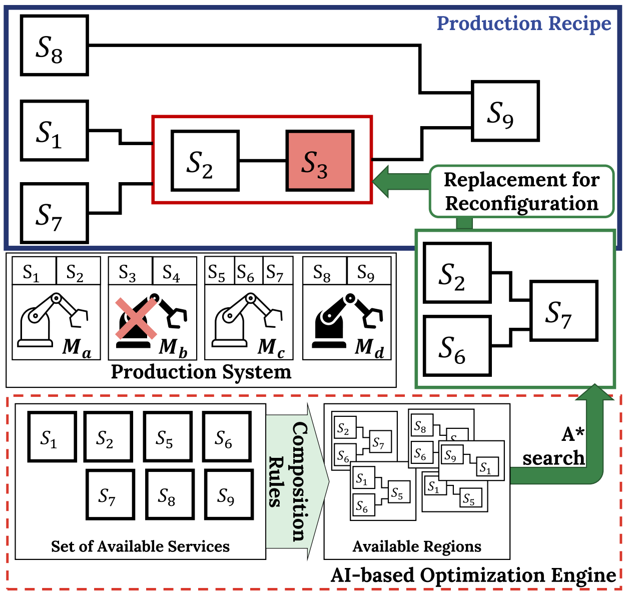
Production Recipe
Replacement for Reconfiguration
Production System
Set of Available Services
Composition Rules
Available Regions
A* search
AI-based Optimization Engine

Fig. 1. Overview of contribution: RRPDGs, along with the proposed transformation rules, support AI-based reconfiguration and optimization of production processes. A manufacturing system composed of four machines M_i, each one exposing a set of services S_j, must implement the depicted production recipe. Suppose a failure of machine M_b: the *red* region can no longer be executed due to the unavailability of S_3. The optimization engine exploits the composition rules defined for RRPDGs to generate a set of candidate regions. The A* algorithm is used to search the optimal solution. In this example, the algorithm identifies the *green* region as the best candidate to replace the red region in the production recipe, thus guiding reconfiguration.

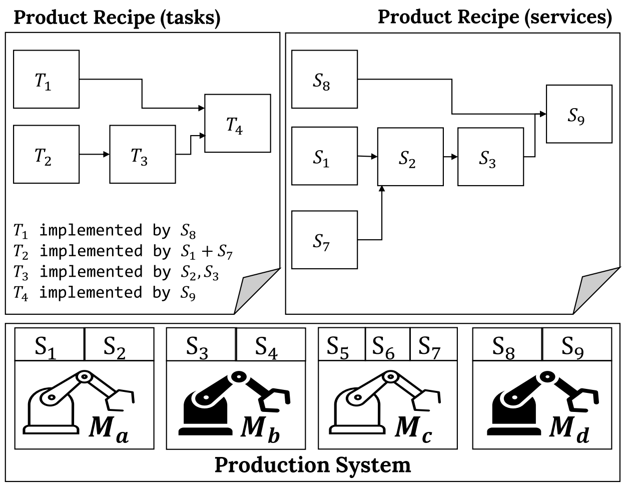
Product Recipe (tasks)
Product Recipe (services)
T_1 implemented by S_8
T_2 implemented by $S_1 + S_7$
T_3 implemented by S_2, S_3
T_4 implemented by S_9
Production System

Fig. 2. Production recipes in the context of SOM. The agents composing the system expose a set of manufacturing services (bottom). A recipe, typically specified as a partially ordered set of production tasks (on the left) is interpreted as a partially ordered set of services (on the right) provided by the agents available in the production system. For instance, Task T_2 is implemented by executing the service S_1 provided by the agent M_a, and the service S_7 provided by the agent M_c; the Task T_2 is implemented by agent M_b executing service S_3 after agent M_a has executed the service S_2.

engine can exploit the rules defined over the proposed formalism to build new regions functionally equivalent to the red region. Then, the engine relies on the A-star (A*) algorithm to find the best-suited region to replace the original one. In Fig. 1, the green region is chosen to replace the red region triggering the system reconfiguration.

The main contribution of this article is the definition of the RRPDG formalism, and the definition of the formal transformation over the proposed models. We advocate that the proposed formalism is well-suited to support different process reconfiguration and optimization techniques based on a set of well-known artificial intelligence (AI) techniques, such as *informed search* and *automated reasoning* over the composition rules. Furthermore, it is worth noticing that the hierarchical structure of RRPDGs allows to efficiently decompose the reconfiguration problem and reduce the time required to explore the state-space.

We evaluate the proposed representation effectiveness by modeling a case study based on a real manufacturing process. We assess the efficiency of the region-based representation in a search context carried out by an informed search algorithm: we compare the results produced by the search procedure using RRPDGs with a state-of-the-art process modeling approach. The proposed graph model is more efficient in providing near-optimal reconfiguration solutions.

The rest of this article is organized as follows. Section II introduces the main concepts used in this work. Then, Section III defines the proposed formalism, i.e., the RRPDG, and the main relations and operations defined over the formalism. Section IV describes a process optimization technique using the RRPDG formalism. Section V presents the case study, as well as the results achieved in our experimental setting. Finally, Section VI concludes this article.

II. Preliminaries

The proposed specification formalism can be used to specify production recipes in any production process. Still, it is specifically tailored for industrial scenarios implementing the service-oriented manufacturing (SOM) paradigm [1], [2]. The main idea of SOM is summarized in Fig. 2. The production process is intended as a set of *services* carried out by the agents operating in the production system. Each agent exposes the set of services that the agent can provide. Borrowing computer engineering concepts, we may say that in SOM the production system is seen as an application programming interface exposing all the functions performed by the agents in the system. An agent providing services may be either a piece of machinery physically executing a manufacturing operation, a piece of the computational infrastructure gathering or providing data, as well as a human–machine interaction device interacting with a human operator who concretely performs the actions necessary to implement the service. Thus, a SOM system may comprehend manufacturing and computation equipment, as well as human operators.

A. Regionalized Value State Dependence Graphs

An RVSDG [9] is an acyclic hierarchical multigraph consisting of *nodes*. A node represents a *computational block* that applies a transformation function from its inputs to its outputs.

There are two main categories of nodes: *simple* (i.e., representing primitive operations), and *structural* (i.e., containing internal RVSDGs, called *regions*). An RVSDG is a structural node itself. Edges model data flow between two nodes, connecting a node's output to another node's input.

A structural node N_R is a quadruple (I, O, N, E), where the following relationships hold.

1) I is the set of *typed inputs* of the region.
2) O is the set of *typed outputs* of the region.
3) N is the set of internal nodes composing the region. Internal nodes may be either simple or structural.
4) E is the set of edges connecting the nodes in N. Each edge $e \in E$ is defined as the tuple $e = (g, u)$, where g is the *origin* node and u is the *user* node.

A simple node can be defined as a structural node whose set of internal nodes is empty. Structural nodes represent complex constructs (*e.g.*, *root region*, *tail-controlled loops*, *conditional statements*, *functions*, *recursive functions*, and *global variables*). RVSDGs define the following structural nodes.

1) γ-node describes a conditional statement. It contains multiple regions $R_0, \ldots R_n$, with $n > 0$, one for each possible decision path. The first input of this node must be a *predicate*, which determines the region to traverse.
2) θ-node depicts a tail-controlled loop. It contains a single region with the loop's body. Its first output is a *Boolean* condition, which determines the loop iteration.
3) λ-node represents a function. It contains a single region with the function body. Its inputs are the function *parameters*, and its single output represents the λ-node itself. The outputs of its internal region specify the function result.
4) δ-node models a global variable. It contains a single region containing a constant value as output. Its inputs represent the *variables* on which the constant depends.
5) ϕ-node characterizes a mutually recursive environment. It contains a single λ-node within its internal region.
6) ω-node is the top-level of an RVSDG. It contains a single region without inputs and outputs.

RVSDGs allows creating acyclic hierarchical structures, thus allowing to partition of the analysis into smaller regions that are easier to analyze.

B. State-of-the-Art

An unexpected event stopping a production causes a cascading effect on the other processes, then directly impacting the overall plant efficiency. As such, production processes' optimization, and reconfiguration are increasingly gaining interest in the research community and among manufacturing companies, and so it is the representation of the information necessary to support such features [10]. The problem of modeling manufacturing processes while supporting process optimization and reconfiguration has been addressed following two main approaches. The first approach relies on graph-based models, such as resource task networks (RTNs) [11] and state task networks (STNs) [12]. RTNs and STNs formalize production recipes as direct graphs specifying the process's parameters and constraints. A STN expresses the sequence of material states associated with tasks, an RTN explicitly specifies the allocation of tasks and resources to physical machines. However, both RTNs and STNs do not support either compositional or hierarchical modeling. Therefore, while they enable reasoning when assuming the expected behavior for the complete production system, they lack support for task-level or finer reasoning [13].

A second kind of approach formalizes the machine's functionalities, often relying on ontologies [14], [15], [16]. Most models formalize the machine's atomic functionalities (i.e., services) through a set of capabilities and constraints [17] describing their effects on the environment. Production recipes are defined as the ordered composition of atomic functionalities [10]. However, the structure of the recipes is usually fixed, and the reconfiguration is done simply by selecting the machine and tools best suited for a given operation. These approaches do not allow modifying the set or the order of operations to complete the production process. Thus, limiting the set of available optimizations when reconfiguring the system. While these methodologies perform well in their low-level vision of the production processes, they are not suitable for addressing higher-level process optimization due to their fixed structure. This limitation is mitigated in [15] by considering a production process as a multiagent system. However, the proposed approach is limited to "plug and produce" production plants, which are still not the state of the practice in manufacturing. Meanwhile, our contribution aims at also applying to production processes carried out by legacy production systems.

III. Manufacturing-Oriented Region-Based Representation

In this section, we propose a representation inspired by RVSDGs to represent production processes. We first present the terminology that will be used in all the subsequent sections. Then, we use the terminology to introduce the definitions and compare our representation with RVSDGs.

A. Basic Concepts

A *manufacturing process* P is represented as a tuple (R, S, I, O, N, E), where the following relationships hold.

1) $R := M \cup C$ is the set of resources used by the production process. A resource $r \in R$ may be either *nonconsumable*, such as the machinery in the production system, or *consumable*, such as materials and machine tools. M is the *set of nonconsumable resources*, while C is *the set of consumable resources*, such that $M \cap C = \emptyset$.
2) $S := \bigcap_{i=1}^{|M|+|C|} S_i$ is the state-space of the production system. Each resource $r_k \in R$ is characterized by a state-space S_k, while $s_{k,t} \in S_k$ is the state of the resource r_k at time t. $\vec{s_t} = [s_{1,t}, \ldots s_{|R|,t}]'$ is the production system's state at time t, given by the vector of the states of the resources in R. The state of a resource $r \in R$ is characterized by a set of attributes. We refer to the attribute i of the resource r as $r.i$.
3) $I \subseteq M \cup C$ is the set of the input of the region, i.e., the set of consumable resources (e.g., material to be used) being

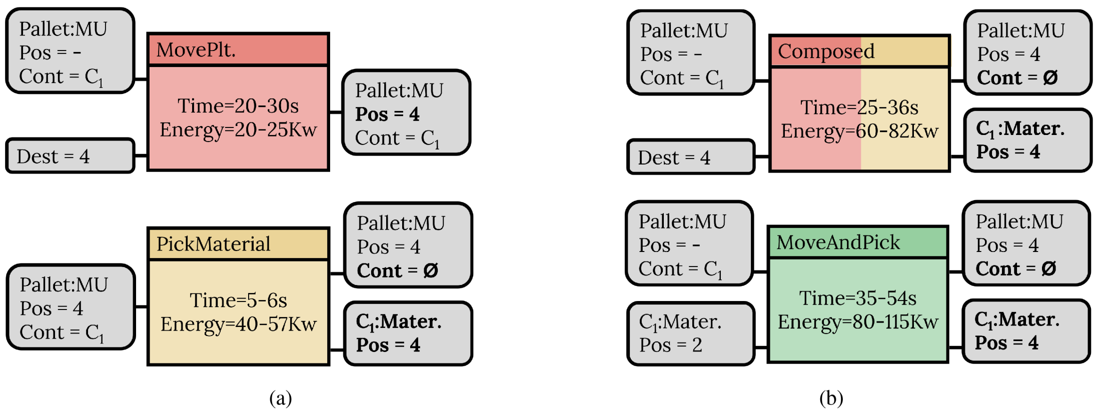

Fig. 3. Example of nodes composing the proposed representation. State transformations are highlighted with bold style. (a) Depicts two *atomic nodes* modeling machine services; (b) depicts two *composed nodes:* the `Composed` node is the composition of the two atomic nodes in (a); the `MoveAndPick` node is a node providing a functionality equivalent to the `Composed` node. The reported values have been measured in the research facility, the ICE laboratory, used in Section V to evaluate and validate the proposed approach. (a) Atomic Nodes. (b) Composition of atomic nodes.

input to the production process, and the nonconsumable resources being used.

4) $O \subseteq M \cup C$ is the set of the output of the region, i.e., the set of consumable resources created by the production process, and the nonconsumable resources made available by the process.

5) $N := A \cup Z$ is the set of nodes representing the *tasks* implementing the production process. A task may be either *atomic* and represented as a simple node, or composed by subservices and represented as a structural node. A is the set of nodes modeling *atomic tasks*, while Z is the set of the nodes modeling *structured tasks*, such that $A \cap Z = \emptyset$. A node is recursively defined as a manufacturing process. Given a node $n \in N$, I_n is its input set, and O_n is its output set; if n is atomic, then $N_n = \emptyset$ Each node $n \in N$ implements a function $f_n : S \to S$, which modifies the state S of the production system P.

6) $E \subseteq N \times N$ is the set of edges connecting the nodes in N, as defined for classic RVSDGs.

The first main difference between RVSDGs and our proposed representation RRPDGs resides in atomic nodes. RRPDGs use machine instruction (e.g., arithmetic instruction, bitwise operations, and memory operations) as atomic nodes, whereas RRPDGs uses manufacturing services (e.g., pallet movement), embedding their functionality into atomic nodes.

Fig. 3 exemplifies the proposed representation for production processes. In Fig. 3(a), `MovePlt` represents a service that moves a resource `Pallet:MU` from an initial position to the destination position identified by the constant value 4. Furthermore, each node has some *internal attributes* specifying the *physical features* of the physical process described by the node, such as the processing time or the energy it consumes. These attributes allow both modeling additional constraints (e.g., maximum time to complete) and node comparison. It is also possible to define *costs* functions based on these attributes, selecting the best nodes carrying out the functionalities. We refer to the internal attribute i of the node n with $n[i]$.

RVSDGs do not allow modeling *race conditions* between multiple processes. Still, manufacturing resources could require lock policies on certain resources, to avoid dangerous situations in which multiple production recipes act on the same resource. In our representation, *producer* and *consumer* nodes model reserve and release policy on a resource r, introducing the possibility to manage race conditions. A producer node reserves a resource of the production system and produces it within the represented production process. Such nodes have an output with the resource r but do not have an input with the same resource r ($r \notin I_n$, and $r \in O_n$). The second atomic node in Fig. 3(a) produces C_1`:Mater` as output. A consumer node is the opposite of a producer node: it consumes the resource r and releases it to the production system. Such nodes have an input with the resource r and do not have an output with the same resource ($r \in I_n$ and $r \notin O_n$).

From the definitions of producer and consumer node, we define the lock on a resource r as follows.

Definition 3.1 (Lock): Given two nodes n_1 and n_2, respectively, producing and consuming a resource r, the lock L_r on r is defined as the tuple $L_r = (n_1, n_2)$.

Nodes consuming and producing a resource r model a function transforming the state of the resource $s_{r,t}$ into a new state $s'_{r,t}$. We define the *transformation* and *identity* node as follows.

Definition 3.2 (Transformation node): Given a node n, with a set of inputs I_n and outputs O_n, n is a transformation node for the input resource r if and only if both of the following conditions stand:

1) $r \in I_n, r \in O_n$, and
2) exists an attribute i of r, such that $I_{n,r.i} \neq O_{n,r.i}$.

Thus, a transformation node modifies an input resource r by producing the same resource r, with at least one of its attributes modified. An identity node consumes and produces a resource r with the same attributes, and it is defined as follows.

Definition 3.3 (Identity node): Given a node n, with a set of inputs I_n and outputs O_n, n is an identity node for the input resource r if and only if both of the following conditions stand:

1) $r \in I_n, r \in O_n$, and

2) it does not exist an attribute i of r, s.t. $I_{n,r.i} \neq O_{n,r.i}$.

Based on these definitions, we can define a set of properties and transformations that allow to safely manipulate a production process implementation while keeping its results unaltered.

B. Equivalence, Consistency, and Transformations

To ensure that process manipulations preserve the overall result, it is necessary to define an *equivalence* relation for the proposed representation. Two manufacturing resources can have *total* or *partial* overlapping functionalities, meaning that both of these resources can be used within a manufacturing process to obtain the same objective result. For instance, consider two pallets with their sets of admissible destinations partially overlapping with each other. The two pallets can be equally used to reach any destination belonging to the intersection. At the same time, there are some destinations reachable only by one of the two pallets. This definition of equivalence better suits manufacturing processes as it defines when two resources can replace each other. The *semantic equivalence* between two resources is defined as follows.

Definition 3.4 (Semantic Equivalence): Given two resources r_1 and r_2, respectively, in a state $s_{r_1,t}$ and $s_{r_2,t}$ of a production system state $\vec{s_t}$, r_2 is semantically equivalent to r_1, with respect to a target production system state $\vec{sf'_t}$ reachable exploiting the resource r_1 if and only if both the following conditions stand:

1) for each transformation node n_1 that can be applied to r_1, transforming its state $s_{r_1,t}$ in $s'_{r_1,t'}$ and the production system state in $\vec{s'_{t'}}$, it exists a transformation node n_2 that can be applied to r_2 and it transforms its state $s_{r_2,t}$ in $s'_{r_2,t''}$ and the production system state in $\vec{s'_{t''}}$;
2) $\vec{s'_{t'}}$ is semantically or strictly equivalent to $\vec{s'_{t''}}$ ($\vec{s'_{t'}} \doteq \vec{s'_{t''}}$).

Given two resources r_1 and r_2, we can check whether they are semantically equivalent as follows. Let us assume r_1 is in an initial state $s_{r_1,t}$, and r_2 is in an initial state $s_{r_2,t}$, both included in a production system state $\vec{s_t}$. Let us consider a sequence of transformation nodes $\mathcal{T}$ transforming $\vec{s_t}$ in the objective state $\vec{sf'_t}$. For each transformation node $n \in \mathcal{T}$ is applied to r_1 and transforming the production system state $\vec{s_t}$ in $\vec{s'_{t'}}$ that can be also applied to r_2 transforming $\vec{s_t}$ in $\vec{s'_{t''}}$, we have the following conditions.

1) $\vec{s'_{t''}}$ is strictly equivalent to $\vec{s'_{t'}}$ or to an intermediate state $\vec{s'_{t+k}}$ (with a distance k from the initial state $\vec{s_t}$) in the sequence that transforms $\vec{s_t}$ in $\vec{sf'_t}$. Then, we can state that r_2 is semantic equivalent to r_1.
2) $\vec{s'_{t''}}$ is not equivalent to $\vec{sf'_t}$ or to an intermediate state $\vec{s'_{t+k}}$, then we can check $\vec{s'_{t''}} \doteq \vec{s_t}$ with the objective state $\vec{sf'_t}$.

If the search terminates without finding a *chain* of nodes for r_2 that transforms the production system's state $\vec{s_t}$ in $\vec{sf_{t'}}$, then r_1 and r_2 are not semantically equivalent. Briefly, a resource is semantically equivalent to another if both resources have a chain of transformation nodes able to transform the initial state $\vec{s_t}$ into the same objective final state $\vec{sf_{t'}}$. This definition implies that whenever two resources are semantically equivalent, a production system can choose to replace one with the other, to reach the same target state. Therefore, we introduce the following Lemma.

Lemma 3.1 (Resource Replaceability): If a manufacturing process p uses a resource r_1 to reach a production system state $\vec{sf_{t'}}$ from the state $\vec{s_t}$, and r_1 is semantically equivalent to a resource r_2, then the process p can replace the resource r_1 with the resource r_2 to reach the same state.

Proof: Suppose that r_1 is semantically equivalent to r_2 with respect to the target state $\vec{sf_{t'}}$, which can be reached from the current state $\vec{s_t}$ only by exploiting the resource r_1. This implies that for each node n acting on the resource r_2 and transforming the current state in a state $\vec{s'_{t'}}$ such that $\vec{s'_{t'}} \not\doteq \vec{s_t}$. This contradicts our hypothesis that r_1 is semantically equivalent to r_2, invalidating the conditions of the Definition 3.4. □

Resources *preemption* is also trivial in manufacturing processes. Since preempted resources in an initial state are modified by the external process and return in an *unknown* state, such a state must be semantically equivalent to the initial one, to enable a *well-formed* preemption. By exploiting the equivalence definition, we define when a lock is well-formed as follows.

Definition 3.5 (Lock consistency): Given a resource r, a lock L_r and two states $s_{r,t}$ and $s'_{r,t'}$ of r (respectively, before and after L_r), the lock L_r is "well-formed" if and only if $s_{r,t} \doteq s'_{r,t'}$ and $s'_{r,t'} \doteq s_{r,t'}$.

This means that for each node n that can be applied to r, and modifies its state $s_{r,t}$, there must be an opposite node n_1 that cancels the side-effect of n. Therefore, a process P lock-consistent with respect to a resource r can *safely* preempt the resource r from another process, guaranteeing that the resource is returned in a state semantically equivalent to the state, in which the resource has been preempted. The advantage of manufacturing services is that they can be assembled to create more complex services. Moreover, production processes comprise a sequence of services in a specific order that implements the represented functionality. By exploiting the node characterization introduced previously, we define the composition of two nodes as follows.

Definition 3.6 (Node composition): Given two nodes n_1 and n_2, their composition $n_1 \circ n_2$ is defined as the structural node z, such as follows:

1) $I_z = I_{n_1} \cup (I_{n_2} \backslash O_{n_1})$, and
2) $O_z = (O_{n_1} \backslash I_{n_2}) \cup O_{n_2}$.

The composition operation is noncommutative. Therefore, $n_1 \circ n_2$ can be different from $n_2 \circ n_1$. The resulting node z is a complex node, enclosing the functionalities of both nodes. Its inputs are the input of the first node and the input of the second node, which are not produced from the first node. Its outputs are the output of the second node and the output of the first node, which are not consumed by the second node. The internal parameters of the resulting node z are computed differently based on the parameter. For example, consider the internal parameter "time" t: $z[t]$ is equal to $n_1[t] + n_2[t]$ if an input $i \in O_{n_1} \wedge i \in I_{n_2}$ exists, $\max(n_1[t], n_2[t])$, otherwise. Now let us take the internal parameter "energy consumption" ec: $z[ec]$ is always equal to $n_1[ec] + n_2[ec]$. In Fig. 3(b), the top node is the composition of the two nodes of Fig. 3(a). Note that if the intersection $O_{n_1} \cap I_{n_2}$ is not empty, then the composition produces a new complex node $z \in Z$ for each permutation of the association tuples (o_{n_1}, i_{n_2}), where $o_{n_1}, i_{n_2} \in O_{n_1} \cap I_{n_2}$ and o_{n_1} is semantically equivalent to i_{n_2}.

Lemma 3.2 (Composition Equivalence): The execution of a complex node $z = n_1 \circ n_2$ in a production system state $\vec{s_t}$ produces a new plant state $\vec{s_t}'$ equivalent to the state $\vec{s_t}''$ obtained by executing the ordered sequence of the two atomic nodes n_1 and n_2 in $\vec{s_t}$.

To support nodes *replaceability* we must define how the nodes of a manufacturing process P can be replaced while preserving the overall functionality of the process. Before introducing replaceability, it is necessary to define *compatibility* between nodes first.

Definition 3.7 (Node Compatibility): Given two nodes n_1 and n_2, n_2 is compatible with n_1 if and only if $I_{n_2} \subseteq O_{n_p}$, where n_p is a region obtained from the composition of all the nodes preceding n_1 in the topological order.

That is, a node n_2 is compatible with a node n_1 if its inputs are in the output of the *environment* that contains the node n_1. The environment of n_1 consists of the composition of all the nodes that are topological predecessors for the replaced node. We define node *replaceability* as follows.

Definition 3.8 (Node Replaceability): Given two nodes n_1 and n_2, n_2 can replace n_1 ($n_2 \approx n_1$) in the state $\vec{s_t}$ if and only if the following conditions hold:

1) n_2 is compatible with n_1, and
2) n_2 transforms the state $\vec{s_t}$ in a state $\vec{s'_{t''}}$ semantically equivalent to the state $\vec{s'_{t'}}$ obtained by applying n_1 ($\vec{s'_{t''}} \mathrel{\hat{=}} \vec{s'_{t'}}$).

Lemma 3.3: Given a manufacturing process p using the node n_1 to transform the state $\vec{s_t}$ into a state $\vec{s'_{t'}}$ and a node n_2 such that $n_2 \approx n_1$, the replacement of the node n_1 with the node n_2 preserves the functionality of the process.

Therefore, to keep the process functionality unaltered, the replacing node must have at least all of its output to be semantically equivalent to the inputs of the replaced node. This guarantees that the subsequent nodes of the original node n_1 can still be applied after the node n_2. For example, in Fig. 3(b) the lower node can replace the upper one if C_1:`Mater` is available in the environment. On the other hand, the upper node cannot replace the lower node.

IV. AI-Based Process Optimization

In the previous section, we have defined the structure of the proposed representation. Real-world production processes are characterized by constraints and requirements that must be satisfied within their entire sequence of services. For example, let us take a pallet and a service that picks the material placed on the pallet. This service should be executed only if the pallet effectively has material on top of it. Therefore, we need to add rules that enable defining when a node $n \in N$ can be executed in a particular state s, avoiding transitions toward invalid states. We define a constraint on a node n as follows.

Definition 4.1 (Constraint): Given a node n with a set of inputs I_n and a set of outputs O_n, a constraint v is defined as a function $v : R \rightarrow \{0, 1\}$, where the following relationships hold.

1) $R \subseteq I_n \cup O_n$.
2) 0 identifies an unsatisfied constraint.
3) 1 identifies a satisfied constraint.

A constraint v is a function that maps the inputs and outputs of a node n into the value 0 or 1. This allows specifying a set of constraints describing when a node can be applied in a certain state. We identify the set of all the constraints of a node n with V_n. Using Definition 4.1, we can declare when a node n is able to produce a transition from a state $\vec{s_t}$ to a valid state $\vec{s'_{t'}}$. We define a "safe transition" as follows.

Rule 4.1 (Safe transition): Given a state $\vec{s_t}$, a node n, and a set of constraints V_n associated to n, we say that the node n produces a "safe transition" to the state $\vec{s'_{t'}}$ if and only if $\forall v \in V_n$ $v(I_n \cup O_n) == 1$.

This rule prevents transitioning toward invalid states, reducing the total number of neighbors in a state $\vec{s_t}$ to the ones satisfying the specified set of constraints V_n. Moreover, it reduces the total number of states that must be evaluated to find a certain solution. Thus, from Definition 4.1, we can extend Definition 3.6 previously introduced as follows.

Rule 4.2 (Node v-composition): Given two nodes n_1 and n_2, each possessing a set of constraints (V_{n_1} and V_{n_2}, respectively), their composition $n_1 \circ^v n_2$ is defined as the complex node z such that

1) $z = n_1 \circ n_2$, and
2) $V_z = V_{n_1} \cup V_{n_2}$.

In particular, the new set of constraints V_z will have: 1) all the constraints associated to a resource r not shared between n_1 and n_2 ($r \in O_{n_1} \land r \in I_{n_2}$); 2) the resulting constraints associated to each resource r, shared between n_1 and n_2 and obtained by solving the systems of equations $V_{z,r} = \{V_{n_1,r} = 1 \land V_{n_2,r} = 1\}$. This allows expressing each constraint $v \in V_z$ only on its inputs I_z and outputs O_z. To represent optimization choices between two nodes and to enable multiobjective search procedures, we extend Definition 3.8, defining the α-replaceability as follows.

Rule 4.3 (Node α-Replaceability): Given two nodes n_1 and n_2, n_2 can α-replace n_1 ($n2 \approx_\alpha n1$) with respect to an internal parameter i if and only if the following conditions hold:

1) $n1 \approx n2$, and
2) $\alpha \geq \frac{n2[i]}{n1[i]}$.

The scaling factor α specifies a deterioration that is tolerated with respect to an attribute. This enables optimization policies and dynamic reconfiguration under some constraints. As an example, in Fig. 3(b) the lower node is α-replaceable with respect to the upper node and the internal attribute *time* where $\alpha \geq 1,4$ for the lower bound and $\alpha \geq 1,5$ for the upper bound. By combining the rules above, we define the following theorem.

Theorem 4.1 (Process manipulation): Let us consider a set of nodes N representing the manufacturing services of a production system, and a manufacturing process p defined using a sequence of nodes $n_i \in \mathcal{T}$ such that $n_i \in N$. The application of the rules safe transition, v-composition, and α-replaceability transforms the process p into a new process p', which produces a final state $\vec{sp'_t}$ semantically equivalent to the final state $\vec{sp_t}$ produced by the process p ($\vec{sp_t} \mathrel{\hat{=}} \vec{sp'_t}$).

Rules 4.1–4.3, respectively, extend Definitions 4.1, 3.6, and 3.8, restricting their applicability to a smaller subset of the items satisfying the extended definitions. As such, they are supported by the same Lemmas 3.1–3.3, on which the definitions are based, guaranteeing the semantic equivalence of the rule's result.

A. Automated Reasoning Over Production Processes

When reasoning over production processes, both the generation and manipulation aim to find the answer for the same question: "which sequences of nodes allow transforming an initial state $\vec{si}_t$ into a final state $\vec{sf}_t$?". While the initial state $\vec{si}_t$ is usually known ($\vec{si}_t = \vec{s}_t$, where $\vec{s}_t$ identifies a generic state), the final state can be partially defined. This allows defining a set of all the possible final states SF, such that each state $\vec{sf}_t \in SF$ is a valid final state. Such a set can be represented through different constraints that identify a valid final state $\vec{sf}_t$.

Let $\vec{si}_t$ be the initial state and $\vec{sf}_t \in SF$ the target final state. Let $T = n_1, \ldots, n_k$ be the set of tasks that allows moving from the state $\vec{si}_t$ to the state $\vec{sf}_t$. Let V_n be the set of constraints associated to a task n. We want to find the sequence of tasks that allows reaching the final state $\vec{sf}_t$ with the minimum cost.

To find such sequence, we exploit the A* algorithm. The A* algorithm is a well-known method in path planning [18]. The classical algorithm uses a cost function $F(\vec{s}_t) = g(\vec{s}_t) + h(\vec{s}_t)$, in which $g(\vec{s}_t)$ is a function that returns the cost to reach the state $\vec{s}_t$ from the initial state $\vec{si}_t$, while $h(\vec{s}_t)$ is a function that returns the estimated cost to reach the target state $\vec{sf}_t$ from the state $\vec{s}_t$. This function allows finding and selecting the nodes with the lowest value, guiding the search to the most promising states. The cost ($g(\vec{s}_t)$) to reach a certain state is given by the sum of the costs of each node n in the sequence of nodes realizing the transition from $\vec{si}_t$ to $\vec{s}_t$. To compute $h(\vec{s}_t)$ we estimate the distance from the current state $\vec{s}_t$ to the goal state $\vec{sf}_t$ and return a proportional value based on the current $g(\vec{s}_t)$.

The algorithm used in this scenario differs from the classical version specifically in how it computes the available neighbors. In our case, the list of neighbors is computed by applying all the tasks $t \in T$ that produce a "safe transition" from the current state $\vec{s}_t$. We note that, in a certain state $\vec{s}_t$, a task n is able to produce multiple "safe transitions," generated by applying n on all the permutations on the inputs I_n in the current state $\vec{s}_t$ that satisfy the constraints V_n. To ensure that A* only selects the nodes necessary to reach the goal state for each transition from a state $\vec{s}_t$ to a new state $\vec{s}_{t'}$, $g(\vec{s}_t)$ must be lower than $g(\vec{s}_{t'})$. If this condition is not satisfied, the algorithm will always choose the transition with a cost of 0 as the next state to evaluate. Therefore, each task $t \in T$ must have a cost greater than zero.

The cost of each node is computed on the internal parameters i. This allows defining the objective function that A* minimizes (e.g., execution time). For all the tasks that reserve a resource, we set their cost equal to the cost of the most expensive task, to avoid reserving useless resources. Similarly, the release of a resource must happen only we the resource is no longer necessary. In this case, the cost of a task that releases a resource is proportional to the last time that the resource has been used in another node, decreasing to zero after a certain time. This principle also enables satisfying Definition 3.5, avoiding keeping the reserved resources.

The proposed representation also enables optimizing the solutions found by the algorithm. For example, after a solution has been found, the reserve and release operations can be optimized to avoid keeping unnecessary resources in a busy state. Thus, the edges between the nodes allow specifying dependencies between tasks and also identifying tasks that can be concurrently executed.

V. Evaluation

To demonstrate the feasibility and the soundness of the proposed representation, we model through RRPDG a case study based on a real manufacturing process. Then, we discuss the results obtained by executing A* on two scenarios with four different heuristics. Finally, we compare the results obtained by using A* with those obtained by using other two search algorithms, i.e., enforced hill climbing (EHC) [19] and single player Monte Carlo tree search (SP-MCTS) [20].

A. Case Study: Bricks Assembly

The case study has been developed in the Industrial Computer Engineering (ICE) laboratory, a research facility equipped with a full-fledged production line.[1] The case study consists of a production process that must assemble three plastic components. As depicted in Fig. 4, two out of three pieces are raw materials retrieved from the warehouse. Such pieces are loaded onto pallets, moved to the assembly station, and composed together to create the first semifinished material C_1. The third piece (C_2) is a semifinished product created with a fused deposition modeling (FDM) 3-D printer. The FDM technology is prone to defects, especially on the surface. Therefore, the material C_2 is processed by a quality control cell: it verifies that the piece meets all the requirements to be identified as the material C_2, such as dimensions, finishing, and color. Then, the materials C_1 and C_2 are assembled to create the final product C_3. Finally, C_3 is moved toward the warehouse to be stored.

The interaction between the abstract representation and the physical process can be accomplished by comparing the expected output of the nodes and the actual data coming from the production system. However, in a typical scenario, the system does not guarantee the availability of information related to the state of the physical process. For example, in Fig. 4, `QltyCtrl` allows determining the material characteristics and checking the physical state with the represented expected state. Moreover, in Fig. 4, if this node returns a different value, the production process is not feasible because the following nodes expect a specific state. For example, suppose that the output of `QltyCtrl` is a material C_4. In this case, there are two possibilities: 1) re-execute all the nodes that produced C_2; 2) generate a new process that leads to the same result.

In the first case, the solution is straightforward: the material C_4 is removed from the process with a consumer node and the previous nodes that produced C_2 are re-executed. The second

[1]The ICE laboratory: https://www.icelab.di.univr.it/

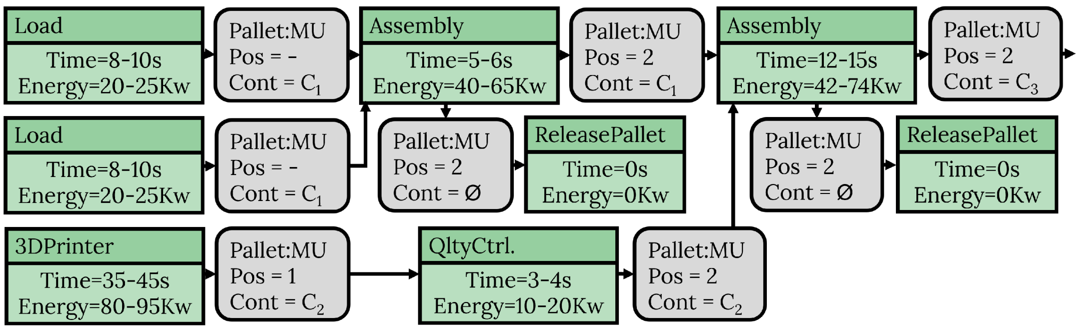

Fig. 4. Schematic representation of the case-study manufacturing process. It consists of different operations to retrieve materials from the warehouse and from a 3-D printer producing a missing piece. Then, the materials are assembled together to produce the final material C_3. Time and energy data have been collected from the real manufacturing system available in our research facility and further described in Section V.

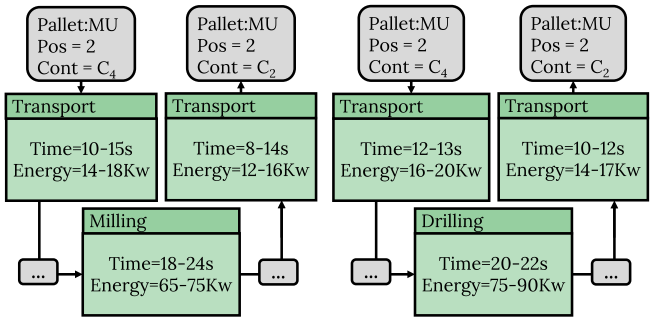

Fig. 5. Two semantic equivalent and replaceable transformation chains that are based on different resources (i.e., milling and drilling). On the left, there is a chain with less energy consumed. On the right, a chain with a smaller time upper bound.

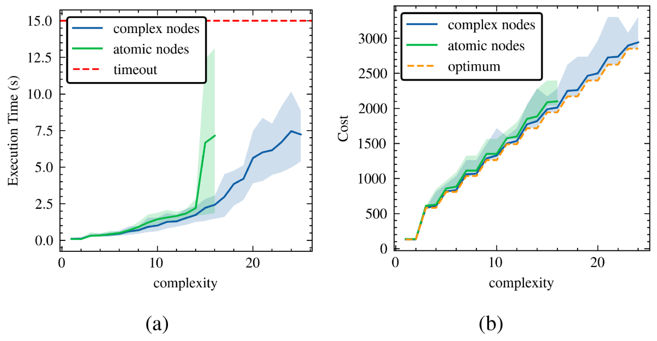

Fig. 6. Results of the first case study. (a) Shows the "execution time" necessary to find a solution. (b) Compares the "cost" of the solution found with the optimal value.

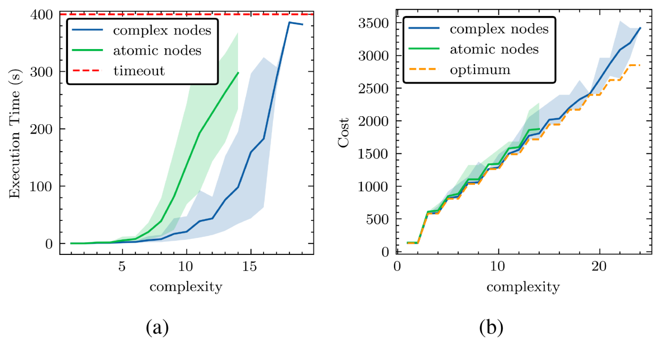

Fig. 7. Results of the second case study. (a) Shows the "execution time" necessary to find a solution. (b) Compares the "cost" of the solution found with the optimal value.

case requires generating a node that allows fixing the unexpected state, transforming C_4 into C_2 (e.g., nodes in Fig. 5). Therefore, the algorithm searches for a node n whose inputs equal the unexpected state, and whose output equals the desired state. This step can be achieved by exploiting the definition of semantic equivalence 3.4. Specifically, it checks that $C_4 \doteq C_2$, where the objective state is C_2. Therefore, the initial and objective states of Definition 3.4 are equal. A more expensive alternative consists in generating the portion of the process that has not already been executed. In this case, A* searches for a replaceable node: the procedure searches for a node n with its inputs equal to all the node outputs not already consumed, and its output equals to the process output.

The search for a transformation node n can also be guided by a cost function. For example, in Fig. 5, there are two possible transformation chains that transform the material C_4 into the material C_2. However, these nodes differ in their internal parameters. For example, the left sequence has a better lower bound but a worse upper bound for the time parameter. Meanwhile, it has better lower and upper bounds for the power consumption parameter. To choose between one of the two implementations, Rule 4.3 for α-replaceability can be exploited, searching between the left and right sequence and choosing the nodes with the desired α parameter. In this case, the outcome is $\alpha \geq 1, 16$ for the time lower bound and $\alpha \geq 0, 89$ for the time upper bound. Similarly, it outputs $\alpha \geq 1, 15$ for the power consumption lower bound and $\alpha \geq 1, 17$ for the upper bound. Therefore, the right sequence guarantees a lower maximum time but the other values favor the left one.

B. Experimental Results

The proposed methodology has been numerically evaluated in two instances of the case study described above. Both case

TABLE I
COMPARISON BETWEEN THE EXECUTION TIMES OF A*, EHC, AND SP-MCTS ON THE FIRST TEN INSTANCES OF THE FIRST CASE STUDY

Search Algorithm	Nodes Type	Execution Time (s)									
		1	2	3	4	5	6	7	8	9	10
A*	atomic	**0.0042**	**0.0769**	**0.3075**	**0.2409**	0.3331	0.3990	0.5831	0.7327	0.9536	1.4165
	complex	0.0467	0.2210	0.3273	0.2972	**0.2932**	**0.3543**	**0.4692**	**0.5092**	**0.6852**	**0.8105**
EHC	atomic	0.0207	0.0197	**0.0217**	0.0238	0.0251	0.0316	0.0341	0.0356	0.0488	0.0592
	complex	**0.0190**	**0.0194**	0.0225	**0.0223**	**0.0239**	**0.0250**	**0.0251**	**0.0240**	**0.0287**	**0.0275**
Monte Carlo Tree Search	atomic	0.7585	1.4179	14.5049	18.2934	33.8593	76.12408	93.7284	105.8163	139.0284	160.2648
	complex	**0.3720**	**0.6205**	**9.8060**	**10.3829**	**16.0613**	**27.6818**	**37.3998**	**49.3694**	**60.2374**	**65.4674**

Bold entities highlights the best results.

TABLE II
COMPARISON BETWEEN THE EXECUTION TIMES OF A*, EHC, AND SP-MCTS ON THE FIRST TEN INSTANCES OF THE SECOND CASE STUDY

Search Algorithm	Nodes Type	Execution Time (s)									
		1	2	3	4	5	6	7	8	9	10
A*	atomic	0.0863	0.2294	1.7376	1.9233	2.3685	4.6071	8.8863	15.4905	33.3744	90.0122
	complex	**0.0353**	**0.0984**	**1.0444**	**1.2506**	**1.2491**	**1.7898**	**2.6687**	**3.4976**	**7.0086**	**9.1611**
EHC	atomic	0.0198	**0.0195**	0.0355	0.0457	0.0578	0.0620	0.0753	0.0812	0.0967	0.1066
	complex	**0.0196**	0.0198	**0.0235**	**0.0250**	**0.0268**	**0.0254**	**0.0293**	**0.0305**	**0.0312**	**0.0307**
Monte Carlo Tree Search	atomic	0.8843	3.1328	27.7759	29.7075	45.6690	83.1213	114.9768	134.8130	-	-
	complex	**0.4252**	**0.6394**	**19.9750**	**17.8347**	**33.0904**	**31.2829**	**44.6783**	**55.3321**	**80.7728**	**122.7710**

Bold entities highlights the best results.

studies start from the same initial state. The two instances differ in the number of resources and tasks: the first consists of 18 atomic tasks and 12 complex tasks acting on 16 unique resources, and the second is composed of 24 atomic tasks and 16 complex tasks acting on 20 unique resources. Each complex task has been created by composing pairs of atomic nodes exploiting Rule 4.2. Both instances have been tested on 20 different final states, increasing the length of the sequence of intermediate states to execute (i.e., the complexity). To evaluate the scalability of the proposed approach and the impact of the introduction of complex nodes, the instances have been initially tested with atomic tasks only. In a second round of experiments, complex nodes have been introduced and tested. Furthermore, each final state has been searched with four different heuristics. The first set is composed of "pwXD" and "pwXU," which have been proposed in [21]. The second set contains "XDP" and "XUP," proposed in [22]. All experiments have been executed on a 3.60 GHz Intel Core i7 with 64 GB of RAM.

Fig. 6 depicts the results of the first case study. Fig. 6(a) compares the total "execution time" necessary to reach a goal state with and without complex nodes. For this test case, a timeout for the execution time was set to 15 s. The introduction of complex nodes allows for reducing the total execution time necessary to find a solution to a third of the initial time. This also allows solving more complex instances. Fig. 6(b) compares the cost of the solution found with respect to the optimal value. It shows that the cost of the solutions found with and without the complex nodes are near-optimal.

The second case study is depicted in Fig. 7. The complexity of this case study increases exponentially with the distance between the initial and the final states. Therefore, in Fig. 7(a), the timeout was increased to 400 s. The results show that with a complexity of 13, the necessary time to find a solution is 300 s without complex nodes, while it reduces to 75 s by exploiting the complex nodes. Thus, with complex nodes, it is possible to solve also instances with a complexity of 20. Moreover, Fig. 7(b) demonstrates that the solutions found with and without the complex nodes are near-optimal.

C. Comparative Analysis

RRPDG is a flexible framework that can be used as a representation formalism by many different optimization algorithms. To demonstrate the versatility and advantages of RRPDGs, we implemented two additional search algorithms: EHC [19] and SP-MCTS [20]. The algorithms have been implemented with different use cases in mind, allowing us to analyze the advantages of complex nodes obtained with the rules defined in the previous sections. EHC has been implemented with a greedy heuristic computed as the distance of the current state to the final goal, ignoring the cost needed to reach the current state. This allows finding a solution (without any guarantee of optimality) faster with respect to A*, enabling the application of our proposed representation in environments with strict timing constraints. SP-MCTS selects the best action to execute in the current state based on the win likelihood of the next states computed by randomly exploring it. Choosing one action at a time enables the application of our proposed representation in dynamic environments, in which the computation of a new solution must be called after each action. SP-MCTS has been implemented with the same heuristic of A*.

Table I reports the execution time necessary to find a solution for the first ten instances of the first case study with the three search algorithms. Notice that with "simple" instances, the introduction of complex nodes does not contribute to decreasing the necessary time to find a solution. Meanwhile, by increasing the complexity of the problems, the introduction of complex nodes leads to an improvement of up to two times for A*, three times for EHC, and three times for SP-MCTS. Among the algorithms, EHC is the fastest one (thanks to the greedy heuristic being used), A* placed second, whereas SP-MCTS provided the worst performance. The higher execution time of SP-MCTS depends on the solution's length (i.e., the number of actions in the solution), since the algorithm is called after each action.

Table II depicts the execution time necessary to find a solution for the first ten instances of the second case study. The higher complexity of the problem is reflected in slower execution times. SP-MCTS is unable to find a solution for the last two instances by relying only on atomic nodes. The algorithm reaches a plateau from which it is not able to escape, thus continuously executing the same set of actions in an infinite loop. By introducing complex nodes, we can observe a greater improvement in the execution time up to ten times for A*, three times for EHC, and three times for SP-MCTS. Thus, SP-MCTS is able to solve the last two instances by exploiting the complex nodes that allow escaping from the plateau.

In general, as a strong incentive toward the adoption of the RRPDG formalism to specify production processes, exploiting regions leads to better performance for all the considered algorithms.

VI. Conclusion

In this article, we introduced RRPDGs: a formalism with an algebra, useful to specify physical processes and production services. We defined a set of rigorous manipulations, aimed at proposing process transformations while preserving the original functionalities. The RRPDG formalism is meant to support different optimization strategies when reasoning on production systems reconfiguration. In this work, we showed an application of AI state-of-the-art search algorithms, integrating RRPDGs transformation rules to guide the procedure. Experiments showed that by exploiting regions algorithms are able to solve larger instances more efficiently, providing at the same time near-optimal solutions.

In the future, we aim to develop even more refined optimization strategies defined over the proposed formal framework. A possible direction may look toward defining methodologies to generate optimized complex nodes in advance. Thus, moving the search complexity offline and, consequently, speeding up the execution time of the proposed search algorithms. Furthermore, we plan to leverage the formal semantics of RRPDGs to develop monitoring and verification techniques useful for proving process properties at runtime.

Acknowledgment

This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

References

[1] F. Tao, L. Zhang, V. C. Venkatesh, Y. Luo, and Y. Cheng, "Cloud manufacturing: A computing and service-oriented manufacturing model," *Proc. Inst. Mech. Engineers, Part B, J. Eng. Manufacture*, vol. 225, no. 10, pp. 1969–1976, 2011.

[2] T. Lojka, M. Bundzel, and I. Zolotová, "Service-oriented architecture and cloud manufacturing," *Acta Polytechnica Hungarica*, vol. 13, no. 6, pp. 25–44, 2016.

[3] S. Gaiardelli, S. Spellini, M. Panato, M. Lora, and F. Fummi, "A software architecture to control service-oriented manufacturing systems," in *Proc. IEEE/ACM Des., Automat. Test Europe Conf. Exhib.*, 2022, pp. 40–43.

[4] J. Zhong-Zhong, F. Guangqi, Y. Zelong, and G. Xiaolong, "Service-oriented manufacturing: A literature review and future research directions," *Front. Eng. Manage.*, vol. 9, no. 1, pp. 71–88, 2022.

[5] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, and I. Luković, "Towards a formal specification of production processes suitable for automatic execution," *Open Comput. Sci.*, vol. 11, no. 1, pp. 161–179, 2021.

[6] G. Wang, D. Li, and H. Song, "A formal analytical framework for IoT-based plug-and play manufacturing system considering product life-cycle design cost," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1647–1654, Feb. 2023.

[7] E. Järvenpää, M. Lanz, and N. Siltala, "Formal resource and capability models supporting re-use of manufacturing resources," *Procedia Manuf.*, vol. 19, pp. 87–94, 2018.

[8] S. Spellini, R. Chirico, M. Panato, M. Lora, and F. Fummi, "Virtual prototyping a production line using assume-guarantee contracts," *IEEE Trans. Ind. Informat.*, vol. 17, no. 9, pp. 6294–6302, Sep. 2021.

[9] H. Bahmann, N. Reissmann, M. Jahre, and J. C. Meyer, "Perfect reconstructability of control flow from demand dependence graphs," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 1–25, Jan. 2015.

[10] M. Weser, J. Bock, S. Schmitt, A. Perzylo, and K. Evers, "An ontology-based metamodel for capability descriptions," in *Proc. IEEE 25th Int. Conf. Emerg. Technol. Factory Automat.*, 2020, pp. 1679–1686.

[11] C. C. Pantelides, "Unified frameworks for optimal process planning and scheduling," in *Proc. 2nd Conf. Foundations Comput. Aided Operations*, 1994, pp. 253–274.

[12] E. Kondili, C. Pantelides, and R. Sargent, "A general algorithm for short-term scheduling of batch operations—I. MILP formulation," *Comput. Chem. Eng.*, vol. 17, no. 2, pp. 211–227, 1993.

[13] J. Ren, Y. Cheng, F. Xiang, and F. Tao, "Platform-based manufacturing service collaboration: A supply-demand aware adaptive scheduling mechanism," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1768–1777, Feb. 2023.

[14] L. V. D. Ginste, A. D. Cock, A. V. Alboom, S. Huysentruyt, E.-H. Aghezzaf, and J. Cottyn, "A formal skill model to enable reconfigurable assembly systems," *Int. J. Prod. Res.*, vol. 61, no. 19, pp. 6451–6466, 2022.

[15] D. Scrimieri, O. Adalat, S. Afazov, and S. Ratchev, "An integrated data- and capability-driven approach to the reconfiguration of agent-based production systems," *Int. J. Adv. Manuf. Technol.*, vol. 124, pp. 1155–1168, 2022.

[16] G. Engel, T. Greiner, and S. Seifert, "Ontology-assisted engineering of cyber–physical production systems in the field of process technology," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2792–2802, Jun. 2018.

[17] J. Backhaus and G. Reinhart, "Digital description of products, processes and resources for task-oriented programming of assembly systems," *J. Intell. Manuf.*, vol. 28, no. 8, pp. 1787–1800, Dec. 2017.

[18] T. Zheng, Y. Xu, and D. Zheng, "AGV path planning based on improved A-star algorithm," in *Proc. IEEE 3rd Adv. Inf. Manageme., Communicates, Electron. Automat. Control Conf.*, 2019, pp. 1534–1538.

[19] S. Akramifar and G. Ghassem-Sani, "Fast forward planning by guided enforced hill climbing," *Eng. Appl. Artif. Intell.*, vol. 23, no. 8, pp. 1327–1339, 2010.

[20] M. P. D. Schadd, M. H. M. Winands, H. J. van den Herik, G. M. J. B. Chaslot, and J. W. H. M. Uiterwijk, "Single-Player Monte-Carlo Tree Search," in *Proc. Comput. Games: 6th Int. Conf.*, 2008, pp. 1–12.

[21] J. Chen and N. R. Sturtevant, "Necessary and sufficient conditions for avoiding reopenings in best first suboptimal search with general bounding functions," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 3688–3696.

[22] J. Chen and N. R. Sturtevant, "Conditions for avoiding node re-expansions in bounded suboptimal search," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019.

Sebastiano Gaiardelli (Student Member, IEEE) received the master's degree in computer science in 2021 and engineering from the University of Verona, Verona, Italy, where he is currently working toward the Ph.D. degree in computer science.

He is involved as a Cofounder and Scientific Advisor with FACTORYAL S.r.l., a startup specializing in factory automation software that originated as a spin-off from the University of Verona. His research interests include the development of new methodologies for the optimization, reconfiguration, and verification of cyber-physical production systems.

Michele Lora (Member, IEEE) received the Ph.D. degree in computer science from the University of Verona, Verona, Italy, in 2016.

From 2020 to 2023, he held a Marie Skłodowska-Curie Global Fellowship with dual appointments at the University of Verona and the University of Southern California, Los Angeles, USA. Previously, he held different research positions in Sweden, the United States, and Singapore. He is currently a Researcher with the University of Verona. He is involved as a Cofounder and Scientific Advisor for FACTORYAL S.r.l., a startup specializing in factory automation software that originated as a spin-off from the University of Verona. His research interests include modeling, simulation, and verification of cyber–physical systems.

Stefano Spellini received the Ph.D. degree in computer science from the University of Verona, Verona, Italy, in 2022, with a thesis proposing a unifying framework to model, verify, and optimize production systems.

He is the Development Team Leader and Co-Founder of FACTORYAL S.r.l., a spin-off company from the University of Verona. He is currently involved in the development of modeling methodologies and tools for cyber-physical production systems.

Franco Fummi (Member, IEEE) received the Laurea degree in electronic engineering and the Ph.D. degree in electronic and communication engineering, in 1990 and 1994, from the Polytechnic of Milan, Milan, Italy, respectively.

Since March 2001 he is a Full Professor in Computer Architecture with the Università di Verona, Verona, Italy. He is leading the Cyber–physical and IoT Systems Design (CISD) group of the Università di Verona, currently composed of more than 20 people, and working on hardware description languages and electronic design automation methodologies for modeling, verification, testing, and optimization of cyber–physical systems. He is also a Co-Founder of two spin-off companies: EDALab, focused on networked embedded systems design, and the automation control software company FACTORYAL.

Open Access funding provided by ‘Università degli Studi di Verona’ within the CRUI CARE Agreement