Jointly Assigning Processes to Machines and Generating Plans for Autonomous Mobile Robots in a Smart Factory

Chistopher Leet[1], Aidan Sciortino[2], Sven Koenig[3]

***Abstract*— A modern smart factory runs a manufacturing procedure using a collection of programmable machines. Typically, materials are ferried between these machines using a team of mobile robots. To embed a manufacturing procedure in a smart factory, a factory operator must a) assign its processes to the smart factory's machines and b) determine how agents should carry materials between machines. A good embedding maximizes the smart factory's throughput; the rate at which it outputs products. Existing smart factory management systems solve the aforementioned problems sequentially, limiting the throughput that they can achieve. In this paper we introduce ACES, the Anytime Cyclic Embedding Solver, the first solver which jointly optimizes the assignment of processes to machines and the assignment of paths to agents. We evaluate ACES and show that it can scale to real industrial scenarios.**

I. INTRODUCTION

Modern smart factories are designed to enable flexible manufacturing [1]. A flexible manufacturing system is a system which can produce a variety of different products with minimal reconfiguration [2]. Flexibility can improve a manufacturer's ability to customize products, reduce the time that it takes to fulfill new orders, and lower the costs of producing a new product. Today, a wide range of industries practice flexible manufacturing, including the automotive, medical, and textile industries [3].

To permit flexible manufacturing, a smart factory needs the following two components:

1) Flexible Machines. Flexible machines are general-purpose machines such as CNC machines which can be programmed to carry out a range of manufacturing processes [4]. Their programmability makes it easy to change the process assigned to each machine when the product produced by the smart factory changes. Their programmability also makes it easy to relieve bottlenecks in a manufacturing procedure by changing the number of machines assigned to its processes.

2) Flexible Transport System. A flexible transport system makes it easy to adjust the flow of parts through a smart factory when the product produced by the smart factory changes. Typically, flexible transport systems use a team of agents to ferry parts between machines [5]. In a traditional smart factory, agents are autonomous mobile robots [5]. In a mag-lev based smart factory, such as BOSCH's ctrlX Flow^{6D} [6], agents are magnetically levitating shuttles. With such a system, adjusting the flow of parts between machines is as simple as adjusting the circulation of agents through the factory.

[1]University of Southern California, cjleet@usc.edu
[2]University of Rochester asciorti@u.rochester.edu
[3]University of Southern California, skoenig@usc.edu

To embed a manufacturing procedure into a smart factory, a factory operator needs to:

1) assign the processes in the manufacturing procedure to the smart factory's machines.
2) find a transport plan which specifies how the smart factory's agents should ferry materials between machines.

We term the problem of embedding a manufacturing procedure into a smart factory the Smart Factory Embedding Problem (SFEP). A good embedding maximizes a smart factory's throughput, the rate at which it outputs products.

To date, the SFEP is open. To our knowledge, no existing smart factory management system [7], [8] jointly optimizes the assignment of processes to machines and paths to agents. Solving these problems separately limits the overall throughput that these systems achieve.

In this paper, we address this lacuna. First, we present the first formal model of the SFEP as a combinatorial optimization problem. We then solve the SFEP by introducing ACES, the Anytime Cyclic Embedding Solver. ACES models the SFEP as a mixed-integer linear program (MILP). Naïvely modeling the SFEP as a MILP does not generate a practical solution. Multi-Agent Path Finding (MAPF), the problem of finding collision-free paths for a team of agents, is an important component of the SFEP. MILP-based solutions to MAPF often struggle to scale [9], limiting their applicability to industry. ACES addresses this problem as follows.

ACES generates cyclic transport plans. A cyclic transport plan starts and ends with its team of agents in the same state. As a result, it can be looped indefinitely. Fixing the length of a cyclic transport plan limits the number of decision variables required to generate it. Decreasing the number of decision variables in a MILP decreases its difficulty. ACES generates cyclic transport plans with incrementally longer loop lengths. The more time that ACES is given, the more loop lengths it considers and the better the throughput of its best plan.

ACES represents the parts being ferried between machines as tokens. ACES considers a variant of the SFEP where these tokens are modelled as "agent-tokens", abstract agents which move between machines under their own power. Two agent-tokens which represent the same part are indistinguishable. Finding a plan for a team of largely indistinguishable agent-tokens requires fewer decision variables than finding a plan for a team of regular agents. A transport plan for token agents can be converted into a transport plan for regular agents in linear time. We evaluate ACES on 6 industrial scenarios and show that it can be applied to real instances of the SFEP.

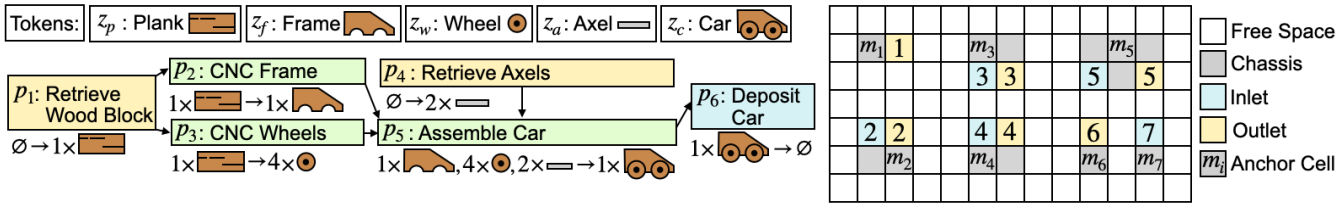

Fig. 1. (left) An example manufacturing procedure. Source process are yellow; sink processes blue. (right) An example smart factory.

II. PROBLEM FORMULATION

A. Manufacturing Procedure

Token. Each raw material, part or assemblage produced or consumed during a manufacturing procedure is modelled as a token. We denote the set of tokens associated with a manufacturing procedure $Z := \{z_1, z_2, \ldots\}$.

Process. A process p_i is an atomic operation in a manufacturing procedure. A process transforms a multiset of input tokens $in(p_i)$ into a multiset of output tokens $out(p_i)$. We denote the number of copies of token $z_j \in Z$ that a process p_i consumes and emits $in_j(p_i)$ and $out_j(p_i)$ respectively.

Processes which does not consume tokens are source processes. They represent operations which retrieve raw materials. Processes which does not emit tokens are sink processes. They represent operations which export finished products or remove waste. We denote a manufacturing procedure's set of processes $P := \{p_1, p_2, \ldots\}$. Exactly one of these processes must be an output process $\text{OUT}(P)$, a sink process which exports finished products.

Manufacturing Procedure. A manufacturing procedure $(Z, P, \text{OUT}(P))$ is a set of processes P which consume and emit tokens from the set Z. It has the output process $\text{OUT}(P)$.

Example. Fig. 1. (left) depicts a manufacturing procedure that makes toy cars. It has two source processes: p_1 and p_4. Its output process is p_6. Process p_2 consumes 1 plank token z_p and emits 1 frame token z_f. Arrows show the flow of tokens through the manufacturing procedure.

B. Smart Factory

Machines. A smart factory contains a set of machines $M := \{m_1, m_2, \ldots\}$. Each machine $m_i \in M$ can run a subset $\mathcal{P}(m_i)$ of the processes in P. Time is discretized. The number of timesteps that machine m_i takes to run a process $p_j \in \mathcal{P}(m_i)$ is denoted $\text{RUNTIME}(m_i, p_j) \in \mathbb{N}$.

A machine has an input and an output buffer. When a machine starts to run a process p_j, it consumes the multiset of tokens $in(p_j)$ from its input buffer. If its input buffer does not contain these tokens, it cannot run process p_j. When it finishes p_j, it emits the multiset of tokens $out(p_j)$ into its output buffer. At timestep t, machine m_i's input and output buffers contain the multisets of tokens $\mathcal{I}(m_i, t)$ and $\mathcal{O}(m_i, t)$.

Layout. We model a factory layout as a 4-connected grid of cells. A cell is traversable if an agent can enter it and non-traversable if it contains an obstacle such as a machine chassis. The set of traversable cells is denoted $C := \{c_1, c_2, \ldots\}$.

Any machine which can run a process that consumes tokens has an input cell $C_{in}(m_i)$. Any machine which can run a process that emits tokens has an output cell $C_{out}(m_i)$. We denote the set of all input and output cells C_{in} and C_{out}. Tokens can only be placed in a machine's input buffer from its input cell and and removed from a machine's and output buffer from its output cell. A machine's input and output cells must be traversable. All input and output cells must be distinct. A machine without an input cell is a source machine. Source machines only run source processes. They represent bins of raw materials. A machine without an output cell is a sink machine. Sink machines only run sink processes. They represent output chutes and waste bins.

Machine Type	**Instances**	**Supported Processes**
Bin of Planks	m_1	p_1
CNC Machine	m_2, m_3, m_4	p_2, p_3
Assembler	m_5	p_5
Bin of Axles	m_6	p_4
Output Chute	m_7	p_6

TABLE I

THE MACHINES IN THE EXAMPLE SMART FACTORY.

Example. Fig. 1. (right) depicts a smart factory. Machine m_2 has the inlet and outlet cell $(1, 2)$ and $(2, 2)$. Table I describes its machines and lists the processes that they can run. There are two source machines: m_1, a bin of planks, and m_5, a bin of axles, and one sink machine: m_6, an output chute.

C. Agents

Tokens are carried between machines by a team of agents $A := \{a_1, \ldots a_n\}$. At the start of a timestep t, an agent a_i occupies a traversable cell. We denote this cell $\pi(a_i, t)$. Each timestep, an agent must wait at its current cell or move to a traversable cell which shares an edge with its current cell.

We represent the actions that an agent can take at each cell with an undirected graph called the movement graph $G := (C, E)$. Each vertex in this graph is a traversible cell. There is an edge $(c_i, c_j) \in E$ between two cells $c_i, c_j \in C \times C$ iff an agent at cell c_i can be at cell c_j on the following timestep. Consequently, each traversible cell c_i is connected to itself by a loop edge and to any traversible cell it shares a side with. Two agents may not occupy the same cell or traverse the same edge in the movement graph on the same timestep.

An agent can carry a single token. We term the token that an agent a_i is carrying on timestep t its cargo and denote it $\sigma(a_i, t)$. If agent a_i is not carrying a token on timestep t, its cargo is the null token z_0. The set of all tokens that an agent can carry $Z \cup \{z_0\}$ is denoted Z_0. The state $(\pi(a_i, t), \sigma(a_i, t))$ of agent a_i on timestep t is its location and its cargo.

An agent with a (non-null) token on a machine's input cell may deposit its token into the machine's input buffer. An agent without a (non-null) token on a machine's output cell may pick up a token from the machine's output buffer.

Picking up and depositing a token takes a single timestep. An agent cannot move during this timestep.

D. Embedding

An embedding describes how a manufacturing procedure is implemented by a smart factory. An embedding is a 6-tuple $(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{O}}, \boldsymbol{\pi}, \boldsymbol{\sigma})$ with the following components:

Assignment Matrix. The assignment matrix $\boldsymbol{\mathcal{A}}$ is an $|M| \times P|$ matrix. The field $\mathcal{A}(m_i, p_j) \in \{0, 1\}$ contains a binary variable which indicates iff machine m_i has been assigned process p_j. A machine can be assigned at most one process.

Rate Matrix. The rate matrix $\boldsymbol{\mathcal{R}}$ is also a $|M| \times P|$ matrix. The field $\mathcal{R}(m_i, p_j)$ indicates the rate, in runs per timestep, that machine m_i runs process p_j at. The rate matrix allows the rate that a machine runs its process at to be decreased, synchronizing it with the rest of the factory. A machine m_i can only run a process p_j at a non-zero rate iff it is assigned that process. The maximum rate that machine m_i can run a process $p_j \in \mathcal{P}(m_i)$ at is $\text{RUNTIME}(m_i, p_j)^{-1}$. A machine runs a process at less than maximum rate by idling for a short time after each run.

Transport Plan. A transport plan $(\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{O}}, \boldsymbol{\pi}, \boldsymbol{\sigma})$ describes how tokens move through the factory. It specifies the state $(\pi(a_i, t), \sigma(a_i, t))$ of each agent $a_i \in A$ and the tokens in the input and output buffers $(\mathcal{I}(m_i, t), \mathcal{O}(m_i, t))$ of each machine $m_i \in M$ at each timestep t.

A factory may need to run a manufacturing procedure for an indefinite amount of time. We thus need to find a cyclic transport plan [10], a transport plan which can be looped repeatedly. A cyclic transport plan has a cycle time T_c and an agent permutation $\Omega : A \to A$. It runs from timestep $t = 0$ to $t = T_c$. The tokens in each machine's input buffer and output buffer at $t = 0$ and $t = T_c$ must be the same:

$$\forall\ m_i \in M,\ \mathcal{I}(m_i, 0) = \mathcal{I}(m_i, T_c) \wedge \mathcal{O}(m_i, 0) = \mathcal{O}(m_i, T_c).$$

An agent $a_i \in A$ must be in the same state at timestep $t = 0$ as the agent $\Omega(a_i)$ at timestep $t = T_c$:

$$\forall\ a_i \in A, \pi(a_i, 0) = \pi(\Omega(a_i), T) \wedge \sigma(a_i, 0) = \sigma(\Omega(a_i), T).$$

A cyclic transport plan can be looped by following the plan from $t = 0$ to $t = T_c$, relabelling the agents in A according to the permutation Ω, and repeating.

E. The Smart Factory Embedding Problem

The throughput $\theta(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{O}}, \boldsymbol{\pi}, \boldsymbol{\sigma})$ of an embedding is the total rate at which its machines run its output process:

$$\theta(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{O}}, \boldsymbol{\pi}, \boldsymbol{\sigma}) := \sum_{m_i \in M} \frac{1}{\mathcal{R}(m_i, \text{OUT}(P))}.$$

In the Smart Factory Embedding Problem (SFEP), we are given a manufacturing procedure $(Z, P, \text{OUT}(P))$ and a smart factory (M, G, A) and asked to find a maximal throughput embedding $(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{O}}, \boldsymbol{\pi}, \boldsymbol{\sigma})$.

III. RELATED WORK

The Multi-Agent Path Finding (MAPF) problem is the problem of moving a team of agents from their starting positions to their goal positions without a collision. MAPF is an important component of the SFEP. The MAPF problem has been solved in a number of different ways, including prioritized planning [11], search [12], answer set programming [13], rule-based AI [14] and SAT solving [15].

MAPF has been studied in the context of other optimization problems. One problem that involves MAPF which is related to the SFEP is the Collective Construction Problem [16]. In the CCP, a team of agents is asked to construct a structure out of building blocks. These building blocks are the same size as the robots, forcing the robots to scale the structure to position the blocks. The CCP is modelled as a combinatorial optimization problem and solved for a single agent using dynamic programming in [17]. This algorithm is extended to multiple agents in [18]. Its solutions, however, achieve little parallelism. The reinforcement learning approach developed in [19] improves parallelism. Optimal solutions to the CCP based on constraint set programming and mixed integer linear programming are proposed in [20].

To our knowledge, the problem of jointly optimizing the assignment of processes to machines and the paths that agents follow to carry materials between machines in a smart factory is not well studied. We believe that we are the first to model this problem as a combinatorial optimization problem. A number of systems for coordinating mobile robots in a manufacturing plant have been proposed. For example, in [7], the authors coordinate mobile robots in a manufacturing plant using a traffic system. In [8], a distributed petri-net assigns tasks to mobile robots and coordinates traffic in a flexible manufacturing system. Neither of these systems, however, optimizes the assignment of processes to machines.

IV. APPROACH

We solve the SFEP in two stages. First, we construct a simplified variant of the SFEP, which we term the Fixed cycle Length, Agent-Token SFEP (FLAT SFEP). We use this solution to construct ACES, a solution to the full SFEP.

A. The Fixed Cycle Length, Agent-Token SFEP

There are 3 differences between the FLAT and full SFEP.

Difference 1. Fixed Cycle Length. In the full SFEP, a transport plan can have any cycle length. In the FLAT SFEP, the transport plan's cycle length is specified in the problem.

Difference 2. Agent-Tokens. In the FLAT SFEP, tokens are modeled as agents which move under their own power. Tokens in the FLAT SFEP move analogously to agents in the full SFEP. Each timestep, a token must move to an adjacent cell, remain at its current cell, or enter a machine's input buffer. Two tokens may not occupy the same cell or traverse the same edge in the movement graph on the same timestep.

A token may only enter a machine's input buffer from its input cell. When a token enters a buffer, it is replaced by a null token. Null tokens may not enter a buffer. Each

timestep, if a machine's output cell contains a null token, the machine may replace the null token with a token from its output buffer. A token may not move on the timestep in which it is placed on the factory floor. There may only be n tokens on the factory floor at any time.

Difference 3. Embedding. A FLAT SFEP embedding has a transport plan for tokens instead of agents. It is an 8-tuple $(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{O}}, \boldsymbol{at}, \boldsymbol{mv}, \boldsymbol{pl}, \boldsymbol{rm})$ with 4 new components:

Position Tensor. The position tensor $\boldsymbol{at}$ is a $|C| \times (T_c + 1) \times |Z_0|$ tensor which describes the positions of the tokens on the factory floor. Let $[0..x]$ be the set of integers $\{0, 1, \ldots, x\}$. The field $at(c_i, t, z_j) \in [0..n]$ specifies the number of copies of token z_j at cell c_i on timestep t. If there is more than one token at cell c_i on timestep t, a collision has occured.

Movement Tensor. Knowing the position of each token on each timestep does not tell you how the tokens move. Since copies of the same token are indistinguishable, there may be multiple ways to produce the configuration of tokens seen at timestep $t + 1$ from the configuration seen at timestep t. The movement tensor $\boldsymbol{mv}$, a $|E| \times (T_c + 1) \times |Z_0|$ tensor, eliminates this ambiguity. Recall that each vertex in the movement graph has a self-loop. The field $mv(c_i, c_j, t, z_k) \in \{0, 1\}$ indicates iff a copy of token z_k:

- moves from c_i to c_j on timestep t when $c_i \neq c_j$
- waits at c_i on timestep t when $c_i = c_j$

Placement and Removal Tensors. + The fields $pl(c_i, t, z_j) \in \{0, 1\}$ and $rm(c_i, t, z_j) \in \{0, 1\}$ indicate that a copy token z_j was removed and placed on cell c_i at timestep t.

B. Why is introducing the FLAT SFEP helpful?

The SFEP can be formulated as an Mixed Integer Linear Program (MILP). Unfortunately, this approach scales poorly. MAPF is an key part of the SFEP. Using a MILP to solve MAPF instances with dozens of agents takes a long time. As a result, solving SFEP instances with many agents using a MILP is impractically slow. The FLAT SFEP is easier to solve as a MILP than the SFEP because it involves fewer variables. A MILP formulation of the SFEP needs to contain a binary variable indicating if cell c_i contains agent a_j on timestep t for every (cell, timestep, agent) combination. A SFEP instance may have dozens of agents. Its optimal embedding may contain dozens of timesteps. As a result, there may be thousands of these variables. The FLAT SFEP needs fewer of these variables since there are usually fewer tokens than agents and it has a limited number of timesteps.

C. Solving the FLAT SFEP

We solve the FLAT SFEP by formulating it as an MILP. The MILP determines the contents of the tensors $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{R}}$, $\boldsymbol{at}$, $\boldsymbol{mv}$, $\boldsymbol{pl}$ and $\boldsymbol{rm}$. We then select the contents of each buffer at timestep $t = 0$, fixing their contents on all other timesteps and thus the contents of the matrices $\boldsymbol{\mathcal{I}}$ and $\boldsymbol{\mathcal{O}}$.

Objective. We maximize the throughput θ of our embedding:

$$\max \sum_{m_i \in M} \mathcal{R}(m_i, \text{OUT}(P))$$

Constraints. Our formulation has three types of constraints: machine configuration constraints, buffer entry and exit constraints, and token movement constraints.

Machine Configuration Constraints. These constraints specify how each machine can be configured.

Constraint 1. A machine is assigned at most 1 process.

$$\forall\ m_i \in M, \sum_{p_j \in P} \mathcal{A}(m_i, p_j) \leq 1.$$

Constraint 2. A machine must able to run its process.

$$\forall\ m_i \in M,\ \forall\ p_j \in P \setminus \mathcal{P}(m_i),\ \mathcal{A}(m_i, p_j) = 0.$$

Constraint 3. Machine m_i can only run a process $p_j \in \mathcal{P}(m_i)$ once every $\text{RUNTIME}(m_i, p_j)$ timesteps.

$$\forall m_i \in M,\ \forall p_j \in \mathcal{P}(m_i), \mathcal{R}(m_i, p_j) \leq \text{RUNTIME}(m_i, p_j)^{-1}$$

Constraint 4. Machine m_i can only run process p_j at a non-zero rate if machine m_i is assigned process p_j.

$$\forall\ m_i \in M,\ \forall\ p_j \in P,\ \mathcal{R}(m_i, p_j) - \mathcal{A}(m_i, p_j) \leq 0.$$

Token Movement Constraints. These constraints specify how tokens move through the factory.

Constraint 5. Over the course of a transport plan, the number of copies of each non-null token $z_j \in Z$ that machine m_i consumes and that enter its input buffer must be the same.

$$\forall\ m_i \in M,\ \forall\ z_j \in Z,$$
$$\sum_{t \in [0..T_c]} rm(C_{in}(m_i), t, z_j) = \sum_{p_k \in P} \mathcal{R}(m_i, p_j) \cdot in_j(p_k) \cdot T_c.$$

Constraint 6. Over the course of a transport plan, the number of copies of each non-null token $z_j \in Z$ that machine m_i emits and that exit its output buffer must be the same.

$$\forall\ m_i \in M,\ \forall\ z_j \in Z,$$
$$\sum_{t=[0..T_c]} pl(C_{out}(m_i), t, z_j) = \sum_{p_k \in P} \mathcal{R}(m_i, p_j) \cdot out_j(p_k) \cdot T_c.$$

Constraint 7. Each timestep t, a token $z_j \in Z_0$ must wait at its cell, move to an adjacent cell, or be removed from the factory floor. Recall that each vertex in the movement graph has a self-loop, and that the field $mv(c_i, c_i, t, z_j)$ indicates if token z_j waits at cell c_i on timestep t.

$$\forall\ c_i, t, z_j \in C \times [0..T_c] \times Z_0,\ at(c_i, t, z_j) =$$
$$\sum_{(c_i, c_k) \in E} mv(c_i, c_k, t, z_j) + \begin{cases} rm(c_i, t, z_j) & c_i \in C_{io} \\ 0 & \text{otherwise} \end{cases}.$$

Constraint 8. The number of copies of token $z_j \in Z_0$ at cell c_i on timestep $t + 1$ is equal to the number of copies of z_j that wait at, move to, or are placed on cell c_i on timestep t.

$$\forall\ c_i, t, z_j \in C \times [0..T_c] \times Z_0,\ at(c_i, t + 1, z_j) =$$
$$\sum_{(c_k, c_i) \in E} mv(c_k, c_i, t, z_j) + \begin{cases} pl(c_i, t, z_j) & c_i \in C_{io} \\ 0 & \text{otherwise} \end{cases}.$$

Algorithm 1 ACES($Z, P, \text{OUT}(P), M, G, timer$)

1: $emb^* \leftarrow$ NULL // Best embedding constructed
2: $\theta^* \leftarrow 0$ // Throughput of best embedding
3: $T_c \leftarrow 1$ // Current cycle time
4: **while** $timer$ has not run out of time **do**
5: $emb, \theta \leftarrow$ FLATSFEP($Z, P, \text{OUT}(P), M, G, T_c, timer$)
6: **if** $\theta \neq$ NULL $\wedge\ \theta > \theta^*$ **then**
7: $emb^*, \theta^* \leftarrow emb, \theta$
8: $T_c \leftarrow T_c + 1$
9: **return** ToSFEPEmb(emb^*)

Constraint 9. A cell may not contain two tokens at once.

$$\forall\ c_i, t \in C \times [0..T_c],\ \sum_{z_k \in Z_0} at(c_i, t, z_j) \leq 1.$$

Together, constraints 7, 8 and 9 imply that at most one token can be placed on or removed from a cell on any timestep.

Constraint 10. If a non-null token $z \in Z$ is removed from an input cell, it must be replaced by a null token z_0.

$$\forall\ c_i, t \in C_{in} \times [0..T_c],\ \sum_{z_k \in Z} rm(c_i, t, z_j) = pl(c_i, t, z_0).$$

Constraint 11. If a non-null token $z \in Z$ is placed on an output cell, it must replace a null token z_0.

$$\forall\ c_i, t \in C_{out} \times [0..T_c],\ \sum_{z_k \in Z} pl(c_i, t, z_j) = rm(c_i, t, z_0).$$

Constraint 12. A non-null token $z_k \in Z$ cannot be placed on an input cell or removed from an output cell.

$$\forall\ c_i, t, z_j \in C_{in} \times [0..T_c] \times Z,\ pl(c_i, t, z_j) = 0.$$
$$\forall\ c_i, t, z_j \in C_{out} \times [0..T_c] \times Z,\ rm(c_i, t, z_j) = 0.$$

Constraint 13. A null token z_0 cannot be placed on an output cell or removed from an input cell.

$$\forall\ c_i, t \in C_{out} \times [0..T_c],\ pl(c_i, t, z_0) = 0.$$
$$\forall\ c_i, t \in C_{in} \times [0..T_c],\ rm(c_i, t, z_0) = 0.$$

Constraint 14. Two tokens may not traverse the same edge in the movement graph on the same timestep.

$$\forall\ (c_i, c_j),\ t \in E \times [0..T_c],\ \sum_{z_k \in Z_0} mv(c_i, c_j, t, z_k) + mv(c_j, c_i, t, z_k) \leq 1.$$

Constraint 15. At most n tokens may be on the factory floor.

$$\sum_{c_i \in C} \sum_{z_j \in Z_0} at(c_i, t, z_j) \leq n$$

Selecting the Initial Contents of a Buffer. We initialize each machine's input buffer with the multiset of tokens that it consumes during the transport plan. As a result, even if replacement tokens arrive late, a machine will always have enough tokens to run its process. We initialize each machine's output buffer with the multiset of tokens that it emits during the transport plan for similar reasons.

Algorithm 2 ToSFEPEmb($\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{O}}, \boldsymbol{at}, \boldsymbol{mv}, \boldsymbol{pl}, \boldsymbol{rm}$)

1: $\boldsymbol{\pi} \leftarrow$ an empty $n \times (T+1)$ matrix // Stores agent positions
2: $\boldsymbol{\sigma} \leftarrow$ an empty $n \times (T+1)$ matrix // Stores agent cargo
3: $i \leftarrow 0$ // Agent index
4: **for** $(c_j, z_k) \in C \times Z$ **do**
5: **if** $at(c_j, 0, z_k) = 1$ **then**
6: $\pi(a_i, 0) \leftarrow c_j$
7: $\sigma(a_i, 0) \leftarrow z_k$
8: **for** t from 0 to T_c **do**
9: **for** $(\pi(a_i, t), c_l) \in E$ **do**
10: **if** $mv(\pi(a_i, t), c_l, t, \sigma(a_i, t)) = 1$ **then**
11: $\pi(a_i, t+1) \leftarrow c_l$
12: $\sigma(a_i, t+1) \leftarrow \sigma(a_i, t)$
13: **if** $rm(\pi(a_i, t), t, \sigma(a_i, t))$ **then**
14: **for** $z_m \in Z_0$ **do**
15: **if** $pl(\pi(a_i, t), t, z_m)$ **then**
16: $\pi(a_i, t+1) \leftarrow \pi(a_i, t)$
17: $\sigma(a_i, t+1) \leftarrow z_m$
18: $i \leftarrow i + 1$
19: **return** $(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{O}}, \boldsymbol{\pi}, \boldsymbol{\sigma})$

D. Solving the SFEP

We use our solution to the FLAT SFEP to construct ACES. ACES is given in Algorithm 1. Let $timer$ be a timer which triggers an interrupt after a specified amount of time. Let FLATSFEP($Z, P, \text{OUT}(P), M, G, T_c, timer$) (Line 5) be a implementation of our solution to the FLAT SFEP which returns an embedding and its throughput (emb, θ) if successful and the tuple (NULL, NULL) if interrupted by the timer. Finally, let ToSFEPEmb(emb^*) be a function which converts a FLAT SFEP embedding into a full SFEP embedding with the same throughput (Line 9). ACES's main loop (Lines 4-8) solves the FLAT SFEP problem for incrementally higher values of T_c until time runs out. ACES then converts the best embedding found into a full SFEP embedding and returns it (Line 9).

Converting a FLAT SFEP Embedding to a SFEP Embedding. To convert a FLAT SFEP embedding into a SFEP embedding, we need to convert its transport plan for agent-tokens into a transport plan for regular agents. Our conversion algorithm is shown in Algorithm 2. If cell c_j contains a copy of token z_k at $t = 0$ in the FLAT SFEP embedding, we position an agent a_i carrying a copy of token z_k on cell c_j at $t = 0$ (Lines 5-7). Agent a_i follows this token as it moves through the factory (Lines 9-12). If this token is replaced with a copy of a new token z_m, agent a_i replaces its cargo with a copy of token z_m by picking up or depositing a token and then starts following this new token (Lines 13-17).

E. Optimizing ACES

ACES is optimized in two ways.

Ignoring Short Cycle Lengths. A cyclic transport plan must move each agent $a_i \in A$ from its start position to agent $\Omega(a_i)$'s starting position. Consequently, agent a_i and $\Omega(a_i)$'s starting positions can be at most T_c cells apart. When T_c is small, a transport plan must use relays of agents to cross large distances. These relays require an impractical number of agents to construct. Consequently, we only generate transport plans with a cycle length greater than 4.

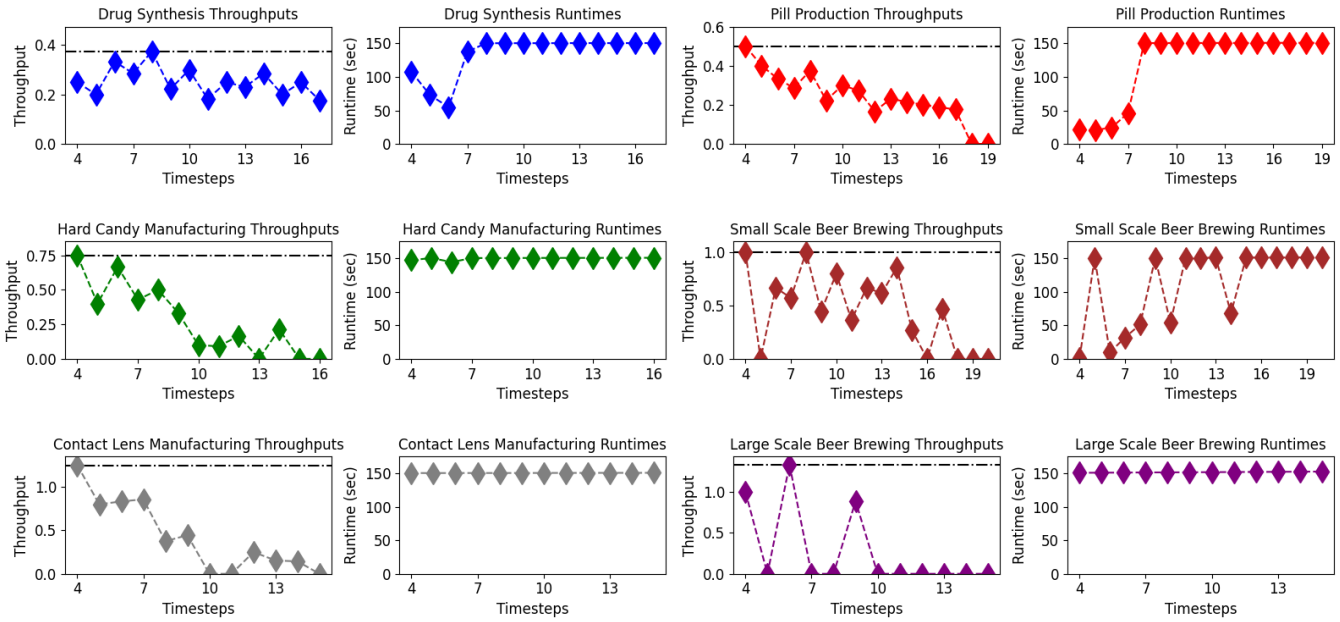

Fig. 2. The FLAT SFEP solver's throughput and runtime on the Benchmark Scenarios.

Scenario Name	Processes	Machines	Max Agents
Drug Synthesis [24]	8	16	100
Pill Production [25]	8	24	100
Hard Candy Man. [26]	8	20	100
Small-Scale Brewing [27]	12	16	100
Contract Lens Man. [28]	6	24	100
Large-Scale Brewing [27]	12	23	100

TABLE II
BENCHMARK SCENARIO DETAILS

Limiting the FLAT SFEP's Runtime. A FLAT SFEP instance's cycle length can make it difficult to solve. ACES often gets stuck on these hard instances. Limiting the FLAT SFEP's runtime by asking it to return the best solution found in a given time frame increases the range of cycle length that ACES examines, which can improve its solution quality.

V. EVALUATIONS

Implementation. We implement ACES in Python 3.11 [21]. We represent the layout graph with the NetworkX [22] library, and solve the FLAT SFEP as an ILP using Gurobi [23].

Methodology. ACES is evaluated on 6 scenarios taken from the pharmaceutical and food manufacturing industries. These industries were chosen since they often require manufacturers to produce many slightly different variations of the same product. A hard candy manufacturer, for example, often wants to produce range of candy with different flavors. As a result, these industries benefit heavily from flexible manufacturing. Table II lists the number of processes and machines and maximum number of agents allowed in each scenario. ACES was given a time limit of 30 minutes in each experiment. This time limit is realistic since embeddings are computed offline. Its FLAT SFEP solver was set to time out after 2.5 minutes.

Experimental Hardware. Each evaluation was performed on a 3.2 GHz, 8 Core AMD Ryzen 5800H CPU with 14 GB of RAM running Ubuntu 20.04.6 LTS.

Results. The 1st and 3rd column of graphs show how the cycle length of a FLAT SFEP instance affects the throughput that the FLAT SFEP solver achieves in each scenario. A throughput of 0.0 indicates that the FLAT SFEP solver could not solve that instance. The throughput of the FLAT SFEP solver's overall best embedding in each scenario is indicated with a black line. There is little correlation between the cycle length of a FLAT SFEP instance and the throughput of its best solution. Our FLAT SFEP solver's solution quality decreases as the cycle length of a FLAT SFEP instance increases, however, because it begins to time out before finding an optimal solution. The FLAT SFEP solver never produced a solution that used all 100 agents, suggesting that its throughput was limited by its machines.

The 2nd and 4th column of graphs show how the cycle length of a FLAT SFEP instance affects the runtime of the FLAT SFEP solver. Note that the FLAT SFEP solver timed out after 150 seconds. Increasing the cycle length of an FLAT SFEP instance usually increases the FLAT SFEP solver's runtime. Interestingly, however, this is not always true. Certain cycle lengths make some FLAT SFEP instances very easy to solve.

VI. CONCLUSION

In this paper, we introduced the SFEP, formulated it as a combinatorial optimization problem, and addressed it with ACES, the Anytime Cyclic Embedding Solver. We see two directions for future work. First, we plan to address larger SFEP instances. Large automotive plants may operate hundreds of machines. ACES cannot scale to SFEP instances of that size. Second, we plan to allow machines to multi-task between processes. Multi-tasking is increasingly common in modern smart factories, but we have neglected it in this paper.

REFERENCES

[1] J. M., A. Haleem, R. P. Singh, and R. Suman, "Enabling Flexible Manufacturing System (FMS) through the Applications of Industry 4.0 Technologies," *Internet of Things and Cyber-Physical Systems*, vol. 2, pp. 49–62, 2022.

[2] K. Höse, A. Amaral, U. Götze, and R. Peças, "Manufacturing Flexibility through Industry 4.0 Technological Concepts—Impact and Assessment," *Global Journal of Flexible Systems Management*, vol. 24, p. 289, 2023.

[3] A. Owen-Hill, "10 Fantastic Examples of Flexible Manufacturing," 2023, accessed: 2024-09-16. [Online]. Available: https://robodk.com/blog/10-examples-of-flexible-manufacturing/

[4] L. D. Evjemo, T. Gjerstad, E. I. Grotli, and G. Sziebig, "Trends in Smart Manufacturing: Role of Humans and Industrial Robots in Smart Factories," *Current Robotics Reports*, vol. 1, pp. 35–41, 2020.

[5] X. Zhao and T. Chidambareswaran, "Autonomous Mobile Robots in Manufacturing Operations," *The International Conference on Automation Science and Engineering*, pp. 1–7, 2023.

[6] BOSCH, "ctrlX Flow: Everything in the flow: Intralogistics Solutions for the Smart Factory," 2024, accessed: 2024-09-14. [Online]. Available: https://apps.boschrexroth.com/microsites/ctrlx-automation/en/portfolio/ctrlx-flow/

[7] M. Jasprabhjit, N. Mauludin, and R. Y. Zhong, "Smart Automated Guided Vehicles for Manufacturing in the Context of Industry 4.0," *Procedia Manufacturing*, vol. 26, pp. 1077–1086, 2018.

[8] D. Herrero-Perez and H. Martinez-Barbera, "Modeling Distributed Transportation Systems Composed of Flexible Automated Guided Vehicles in Flexible Manufacturing Systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 2, pp. 166–180, 2010.

[9] P. Surynek, "Unifying Search-based and Compilation-based Approaches to Multi-agent Path Finding through Satisfiability Modulo Theories," *The Twenty-Eighth International Joint Conference on Artificial Intelligence*, p. 1177–1183, 2019.

[10] C. Leet, C. Oh, M. Lora, S. Koenig, and P. Nuzzo, "Co-Design of Topology, Scheduling, and Path Planning in Automated Warehouses," *The Design, Automation and Test in Europe Conference and Exhibition*, pp. 1–6, 2023.

[11] P. Lozano-Pérez and M. Erdmann, "A novel approach to path planning for multiple robots in bi-connected graphs," *Algorithmica*, vol. 2, pp. 477–521, 1987.

[12] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-Based Search for Optimal Multi-Agent Path Finding," *Artificial Intelligence*, vol. 219, p. 40–66, 2012.

[13] V. Nguyen, P. Obermeier, T. C. Son, T. Schaub, and W. Yeoh, "Generalized Target Assignment and Path Finding Using Answer Set Programming." *The International Joint Conference on Artificial Intelligence*, pp. 1216—-1223, 2017.

[14] B. De Wilde, A. W. Ter Mors, and C. Witteveen, "Push and Rotate: A Complete Multi-Agent Pathfinding Algorithm," *Journal of Artificial Intelligence Research*, vol. 51, no. 1, pp. 443—492, 2014.

[15] J. Yu and S. LaValle, "Multi-agent Path Planning and Network Flow," *Algorithmic Foundations of Robotics X: Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics*, pp. 157—-173, 2013.

[16] H. Durrant-Whyte, N. Roy, and P. Abbeel, "TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction," *Robotics: Science and Systems*, pp. 257–264, 2012.

[17] T. K. S. Kumar, S. Jung, and S. Koenig, "A Tree-Based Algorithm for Construction Robots," *The International Conference on Automated Planning and Scheduling*, pp. 481–489, 2014.

[18] T. Cai, D. Y. Zhang, T. K. S. Kumar, S. Koenig, and N. Ayanian, "Local Search on Trees and a Framework for Automated Construction Using Multiple Identical Robots," *The 2016 International Conference on Autonomous Agents and Multiagent Systems*, p. 1301–1302, 2016.

[19] G. Sartoretti, Y. Wu, W. Paivine, T. Kumar, S. Koenig, and H. Choset, "Distributed Reinforcement Learning for Multi-Robot Decentralized Collective Construction," *The International Symposium on Distributed Autonomous Robotic Systems*, pp. 35—-49, 2018.

[20] E. Lam, P. J. Stuckey, S. Koenig, and T. K. S. Kumar, "Exact Approaches to the Multi-agent Collective Construction Problem," *The International Conference on Principles and Practice of Constraint Programming*, pp. 743—758, 2020.

[21] Python Software Foundation, "Python 3.11.10 Documentation," 2024, accessed: 2024-09-15. [Online]. Available: https://docs.python.org/3.11/

[22] A. A. Hagberg, D. A. Schult, and P. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," *The 7th Python in Science Conference*, pp. 11–15, 2008.

[23] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024, accessed: 2024-09-15. [Online]. Available: https://www.gurobi.com

[24] S. D. Schaber, D. I. Gerogiorgis, R. Ramachandran, J. M. B. Evans, P. I. Barton, and B. L. Trout, "Economic Analysis of Integrated Continuous and Batch Pharmaceutical Manufacturing: A Case Study," *Industrial and Engineering Chemistry Research.*, pp. 10 083–10 092, 2011.

[25] R. Singh, "System Engineering for a Novel Continuous Pharmaceutical Manufacturing Process," 2019, accessed: 2024-09-15. [Online]. Available: https://www.pharmafocusasia.com/manufacturing/system-engineering-pharmaceutical-manufacturing-process

[26] N. Efe and P. Dawson, "A Review: Sugar-Based Confectionery and the Importance of Ingredients," *European Journal of Agriculture and Food Sciences*, vol. 4, pp. 1–8, 2022.

[27] L. H. G. van Donkelaar, J. Mostert, F. K. Zisopoulos, R. M. Boom, and v. A-J., "The Use of Enzymes for Beer Brewing: Thermodynamic Comparison on Resource Use," *Energy*, pp. 519–527, 2016.

[28] J. Xu, Y. Xue, G. Hu, T. Lin, J. Gou, T. Yin, H. He, Y. Zhang, and X. Tang, "A comprehensive review on contact lens for ophthalmic drug delivery," *Journal of Controlled Release*, vol. 281, pp. 97–118, 2018.