An Anytime, Scalable and Complete Algorithm for Embedding a Manufacturing Procedure in a Smart Factory

Christopher Leet[1], Aidan Sciortino[2], and Sven Koenig[3]

***Abstract*— Modern automated factories increasingly run manufacturing procedures using a matrix of programmable machines, such as 3D printers, interconnected by a programmable transport system, such as a fleet of tabletop robots. To embed a manufacturing procedure into a smart factory, an operator must: (a) assign each of its processes to a machine and (b) specify how agents should transport parts between machines. The problem of embedding a manufacturing process into a smart factory is termed the Smart Factory Embedding (SFE) problem. State-of-the-art SFE solvers can only scale to factories containing a couple dozen machines. Modern smart factories, however, may contain hundreds of machines. We fill this hole by introducing the first highly scalable solution to the SFE, TS-ACES, the Traffic System based Anytime Cyclic Embedding Solver. We show that TS-ACES is complete and can scale to SFE instances based on real industrial scenarios with more than a hundred machines.**

I. INTRODUCTION

Flexible manufacturing is a key objective of the modern manufacturing industry [1]. A smart factory is flexible if it can be easily reconfigured to produce different products. Flexible manufacturing systems can reduce the cost of producing new products, lower the time required to fulfill orders, and allow products to be customized. To perform flexible manufacturing, a smart factory needs two components:

Flexible Machines. Flexible machines are general purpose machines such as CNC machines which can be used to perform a range of manufacturing processes. Flexible machines can be easily reprogrammed with a new process when a smart factory's manufacturing procedure changes.

Flexible Transport System. Flexible transport systems make it easy to adjust the materials that the machines in a smart factory are supplied with. Most flexible transport systems transport materials with a team of agents [2]. These agents are generally autonomous mobile vehicles [2]. However, recent mag-lev based systems such as BOSCH's ctrlX Flow^{6D} [3].

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

1) assign each process in the manufacturing procedure to one or more machines in the smart factory.
2) construct a transport plan that specifies how the smart factory's agents should carry parts between machines.

The problem of embedding a manufacturing procedure in a smart factory is termed the Smart Factory Embedding (SFE) problem. A good embedding maximizes the smart factory's throughput, that rate that it makes finished products.

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

Most existing systems for coordinating agents in a smart factory assume processes have already been assigned to the smart factory's machines [4], [5]. Assigning processes to machines and paths to agents separately, however, limits the throughput that these solvers can achieve.

One recent approach to the SFE problem, ACES [6], jointly optimizes its two components. ACES models a smart factory as a grid of cells. Time is discretized. ACES uses a Mixed Integer Linear Program (MILP) to jointly assign processes to machines and find a transport plan to its agents. ACES generates a transport plan which loops after a certain number of timesteps, allowing it to be run continuously.

Unfortunately, ACES scales poorly. ACES's MILP has a binary variable which indicates if a given cell contains an agent carrying a given component on given timestep for every possible (cell, component, timestep) combination. A SFE instance may have hundreds of cells and tens of components. A transport plan may have tens of timesteps. As a result, when ACES is used to solve a large SFE instance, it may generate a MILP with 100,000s of these variables. A MILP with 100,000s of variables is often difficult to solve. Thus, to date, there is no solver which jointly optimizes both components of the SFE problem that works at scale.

We address this hole by proposing the Traffic System based Cyclic Embedding Solver, TS-ACES. TS-ACES is based on the following observation: most smart factories coordinate agents using a traffic system, a network of roads. TS-ACES aggregates timesteps into epochs. It uses a MILP to construct a traffic system based embedding, an embedding which moves agents through a traffic system at the rate of one road per epoch. A traffic system often has several times fewer roads than cells. A traffic system based embedding often has several times fewer epochs than timesteps. TS-ACES's MILP thus often has dozens of times fewer variables than ACES's MILP, making it much easier to solve.

The throughput and runtime of TS-ACES's MILP are dependent on its hyperparameters, the number of epochs and the length of an epoch in the embedding that it generates. We introduce a novel, principled search algorithm to find good values for these hyperparameters. TS-ACES uses its traffic system based embedding to construct a transport plan generator. This generator can generate timestep by timestep instructions for the smart factory's agents in real time, allowing it to run its manufacturing procedure indefinitely.

We analyze TS-ACES and show that it is complete. Our evaluations show that TS-ACES can scale to SFE instances based on real scenarios with more than 100 machines.

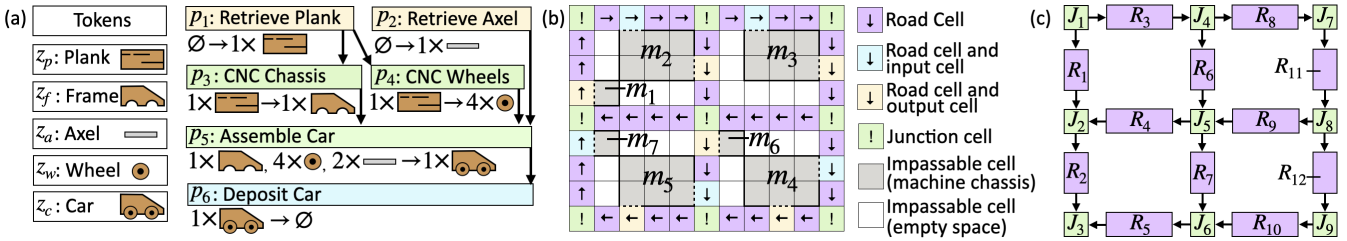

Fig. 1. (a) An example manufacturing procedure. Source and sink process are yellow and blue. (b) An example smart factory and (c) its traffic system.

II. Problem Formulation

Our model of the SFE problem adapts the model given in [6] to smart factories that use traffic systems.

A. Manufacturing Procedure

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

Process. A process p_i is an atomic operation in a manufacturing procedure. Each process p_i consumes a multiset of input tokens $\text{INTK}(p_i)$ and emits a multiset of output tokens $\text{OUTTK}(p_i)$. The number of copies of a token $z_j \in Z$ that a process p_i consumes and produces are denoted $\text{NUMIN}(p_i, z_j)$ and $\text{NUMOUT}(p_i, z_j)$.

A process that does not consume any tokens is a source process. Source processes represent operations which retrieve raw materials. A process which does not produce any tokens is a sink process. Sink processes represent operations which remove finished products or waste. A manufacturing procedure's the set of processes is denoted $P := \{p_1, p_2, \ldots\}$. Exactly one of these processes $\text{OUTP}(P)$ must be an output process, a sink process which exports finished products.

Manufacturing Procedure. A manufacturing procedure (Z, P) is a set of tokens and a set of processes that consume and produce those tokens.

Example. Fig. 1. (a) shows a manufacturing procedure for toy cars. It has the tokens $Z := \{z_p, z_f, z_a, z_w, z_c\}$, the source processes p_1 and p_4, and the sink process p_6, which is also an output process. The tokens that a process consumes and produces are shown below it. Process p_5 consumes a chassis, 4 wheel and 2 axle tokens and produces a car token.

B. Smart Factory

Machines. A smart factory has a set of machines $M := \{m_1, m_2, \ldots\}$. A machine m_i can run a subset $\mathcal{P}(m_i) \subseteq P$ of a manufacturing procedure's processes. Time is discretized. A process p_j takes a machine m_i $\text{RUNTIME}(m_i, p_j)$ timesteps to run. Machines have input and output buffers. To initiate a process p_j, a machine must consume the multiset of tokens $\text{INTK}(p_j)$ from its input buffer. When a machine finishes process p_j, it deposits the multiset of tokens $\text{OUTTK}(p_j)$ into its output buffer. Source and sink machines are special types of machines. Only source machines can run source processes. They represent bins of raw materials. Only sink machines can run sink processes. They represent chutes for waste and finished products.

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.

Layout. We model the layout of a smart factory as a grid of square cells. Each cell has a set of entry cells and a set of exit cells, which must be vertically or horizontally adjacent. An agent can only enter a cell from its entry cells and leave a cell to its exit cells. Our model has three types of cells:

- *Road Cells.* A road cell has one entry and one exit cell.
- *Junction Cells.* A junction cell has at least one entry and exit cell. These cells must be road cells.
- *Non-Traversable Cells.* Agents may not enter non-traversable cells. Any cell which contains an obstacle like a machine chassis must be a non-traversable cell.

Let cell c_i be the ith road or junction cell in a layout. Let $C := \{c_1, c_2, \ldots\}$ be the set of all road and junction cells in a layout. We represent the connections between the cells in a layout with a directed graph called a layout graph $G_L := (C, E_L)$. Each vertex in the layout graph is a road or junction cell. There is an arc $(c_i, c_j) \in E_L$ iff cell c_i is one of cell c_j's entry cells. Note that if cell c_i is one of cell c_j's entry cells, then cell c_j must be one of cell c_i's exit cells.

To be valid, a layout must have the following properties:

1) Its layout graph must be strongly connected. If a layout graph is not strongly connected, agents may not be able to carry tokens between certain pairs of machines.
2) It must contain at least one junction cell. In this paper, we exclude the trivial case where a factory's machines are connected by a single loop of road cells.

Any machine's input buffer and output buffer is associated with an input cell and an output cell. An agent can only deposit tokens into an input buffer on its input cell and remove tokens from an output buffer on its output cell.

Example. Fig. 1. (b) depicts an example smart factory. Regular road cells are colored purple and junction cells green. Road cells which are also input and output cells are colored blue and yellow. An impassable cell is gray if it contains a machine's chassis and white if it empty. A machine's border with its input and output cells is indicated with a dotted line. For example, the cells $(5, 1)$ and $(0, 2)$ are machine m_5's input and output cells. Table I describes this smart factory's machines. It has two source machines, m_1 and m_5, and one sink machine, m_6.

Traffic System. We group the road and junction cells in a layout into roads and junctions. A road R_i is a path of $\text{LEN}(R_i)$ road cells which connects two junction cells. We denote the set of roads in a layout $\mathcal{R} := \{R_1, R_2, \ldots\}$. A junction J_i is single junction cell. Each junction connects two or more roads. We denote the set of junctions in a layout $\mathcal{J} := \{J_1, J_2, \ldots\}$. A layout's roads and junctions collectively form a traffic system $(\mathcal{R}, \mathcal{J})$. Fig. 1. (c) shows the example smart factory's traffic system. It has 9 junctions $\mathcal{J} := \{J_1, \ldots, J_9\}$ and 12 roads $\mathcal{R} := \{R_1, \ldots, R_{12}\}$.

A junction J_i has a set of entry and exit roads $\text{ENTRY}(J_i)$ and $\text{EXIT}(J_i)$. Any road which leads to and away from J_i is one of its entry and exit roads. For example, in the example smart factory, junction J_8 has the entry roads $\text{ENTRY}(J_8) = \{R_{11}\}$ and the exit roads $\text{EXIT}(J_8) = \{R_9, R_{11}\}$.

C. Agents

A team of n agents $A := \{a_1, \ldots, a_n\}$ carries tokens between machines. At the start of each timestep t, each agent $a_i \in A$ occupies a road or junction cell. We denote this cell $\pi(a_i, t) \in C$. Each timestep, an agent $a_i \in A$ must either wait at its current cell or move to one of its current cell's exit cells. Two agents may not occupy the same cell or traverse the same edge in the layout graph on the same timestep.

Agents can carry a single token at a time. We term the token that an agent a_i is carrying its cargo. If an agent is not carrying a token, we say that it is carrying the null token z_0. We denote the cargo that agent a_i is carrying on timestep t $\sigma(a_i, t) \in Z \cup \{z_0\}$. The state $(\pi(a_i, t), \sigma(a_i, t))$ of agent a_i on timestep t is its location and cargo.

If an agent carrying a non-null token is on a machine's input cell at the end of a timestep t, it may deposit its token into the machine's input buffer. If an agent carrying a null token is on a machine's output cell at the end of a timestep t, it may pick up a token from the machine's output buffer.

D. Cell Based Embedding

A cell based embedding specifies how a manufacturing procedure is implemented by a smart factory. It is a 3-tuple $(\boldsymbol{\alpha}, \boldsymbol{\lambda}, \mathcal{G})$ with the following elements:

Assignment Matrix. An assignment matrix $\boldsymbol{\alpha}$ is a $|M| \times |P|$ matrix. The field $\alpha(m_i, p_j)$ contains a binary variable which indicates if machine m_i has been assigned process p_j. A machine may only be assigned one process.

Rate Matrix. A rate matrix λ is also a $|M| \times |P|$ matrix. The field $\lambda(m_i, p_j)$ indicates the rate that machine m_i runs process p_j. A machine m_i can only run process p_j at a non-zero rate if it is assigned process p_j.

Cell Based Transport Plan. A cell based transport plan specifies how agents carry tokens through a smart factory. A smart factory often has to run a manufacturing procedure for an indefinite length of time. We would thus like to construct a transport plan generator $\mathcal{G}$, a function which takes a smart factory's state $\phi(t)$ at any timestep t and computes its state $\phi(t+1)$ at timestep $t+1$. As long as our generator can run in real time, we can run a manufacturing procedure endlessly.

The state $\phi(t)$ of a smart factory at timestep t is a 4-tuple $(\boldsymbol{\mathcal{I}}(t), \boldsymbol{\mathcal{O}}(t), \boldsymbol{\pi}(t), \boldsymbol{\sigma}(t))$ with the following components:

Buffer Contents Vectors. The input and output buffer contents vectors $\boldsymbol{\mathcal{I}}(t)$ and $\boldsymbol{\mathcal{O}}(t)$ are length $|M|$ vectors. The fields $\mathcal{I}(m_i, t)$ and $\mathcal{O}(m_i, t)$ specify the contents of a machine m_i's input and output buffers on timestep t. If machine m_i does not have an input buffer, $\mathcal{I}(m_i, t) = \emptyset$. If it does not have an output buffer, $\mathcal{O}(m_i, t) = \emptyset$.

Agent State Vectors. The position and cargo vector $\boldsymbol{\pi}(t)$ and $\boldsymbol{\sigma}(t)$ are length $|A|$ vectors. The fields $\boldsymbol{\pi}(a_i, t)$ and $\boldsymbol{\sigma}(a_i, t)$ specify agent a_i's position and cargo on timestep t.

E. Smart Factory Embedding Problem

The throughput $\theta(\boldsymbol{\alpha}, \boldsymbol{\lambda}, \mathcal{G}) := \sum_{m_i \in M} \lambda(m_i, \text{OUTP}(P))$ of an embedding $(\boldsymbol{\alpha}, \boldsymbol{\lambda}, \mathcal{G})$ is the total rate at which its machines run its output process. In the smart factory embedding problem, we are given a manufacturing procedure (Z, P), and a smart factory $(M, C, A, \mathcal{J}, \mathcal{R})$ and asked to find a maximum throughput embedding. We term the tuple $(Z, P, M, C, A, \mathcal{J}, \mathcal{R})$ a SFE instance and denote it $inst$.

III. Related Work

The Multi-Agent Path Finding (MAPF) problem is the problem of moving a team of agents from start locations to goal locations without collisions. MAPF is an key part of SFE. MAPF has been solved via a range of methods such as SAT solving [7] and answer set programming [8].

Many other optimization problems involve MAPF. One problem which incorporates MAPF and is related to the SFE problem is the Collective Construction (CC) problem. In the CC problem, a team of agents is tasked with arranging building blocks into a structure. The agents are the same size as the blocks, forcing them to climb the structure to position the block. The CC problem is formalized as a combinatorial optimization problem and solved for a single agent using dynamic programming [9]. This approach is generalized to multiple agents in [10]. Empirically, however, the solutions that it generates do not achieve much parallelism. The reinforcement learning approach proposed in [11] finds solutions with more parallelism. The CC problem is solved optimally using both constraint set programming and a MILP in [12].

Several systems for coordinating agents in a smart factory have been proposed. In [4], the authors use a traffic system to traffic system to generate paths for agents in a manufacturing facility. In [5], the authors assign tasks to agents and plan their paths using a distributed petri net. Neither system, however, assigns processes to machines. ACES [6] solves the SFE problem using a MILP. It is unclear, however, whether it can scale to SFE instances with more ~ 25 machines.

IV. Overview

TS-ACES solves the SFE by constructing a traffic system based embedding and then converting it into a cell based embedding. In this section, we define a traffic system based embedding and then describe TS-ACES's workflow.

Traffic System Based Embedding. A traffic system based embedding is based on roads, not cells. It aggregates timesteps

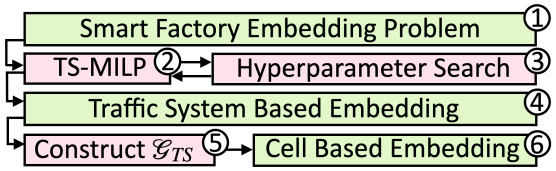

Fig. 2. TS-ACES's workflow.

into epochs. It moves agents one road per epoch. At the start of an epoch, each agent is in a queue at the head of a road. During an epoch, each agent passes through the junction J_i at the end of its road and joins a queue on one of J_i's exit roads. The number of timesteps in an epoch is denoted T_e.

A manufacturing procedure often has to be run for an indefinite length of time. We therefore construct an embedding with a cyclic transport plan [13], a transport plan that can be looped repeatedly. A cyclic transport plan is associated with a number of epochs N_e. It loops after N_e epochs.

A traffic system based embedding can be written as an 6-tuple: $(\boldsymbol{\alpha}, \boldsymbol{\lambda}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{O}}, \boldsymbol{pk}, \boldsymbol{dp})$. Its assignment and rate matrices $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$ are written using the same matrices as a cell based embedding. Its transport plan is expressed as a 4 tuple $(\boldsymbol{inB}, \boldsymbol{outB}, \boldsymbol{pk}, \boldsymbol{dp})$ with the following new components:

Inbound and Outbound Traffic Tensors. The inbound and outbound traffic tensors $\boldsymbol{inB}$ and $\boldsymbol{outB}$ are $|\mathcal{R}| \times N_e + 1 \times |Z|$ tensors. The fields $inB(R_i, T, z_j)$ and $outB(R_i, T, z_j)$ specify the number of agents carrying token z_j that enter and leave road R_i during epoch T.

Pick Up and Deposit Tensors. The pickup and deposit tensors $\boldsymbol{pk}$ and $\boldsymbol{dp}$ are $|M| \times N_e + 1 \times |A|$ tensors. Let R_{out} and R_{in} be the roads that contain machine m_i's output and input cells. The field $pk(m_i, T, z_j)$ specifies the number of agents entering road R_{out} during epoch T that pick up a copy of token z_j from m_i's output buffer. The field $dp(m_i, T, z_j)$ specifies the number of agents entering R_{in} during epoch T that pick up a copy of z_j from m_i's input buffer.

Since our embedding loops after N_e epochs, the values of its tensors for the epoch $T = 0$ and $T = N_e$ are the same.

TS-ACES*'s Workflow.* The workflow of TS-ACES is shown in Fig. 2. TS-ACES uses a MILP termed the TS-MILP ② to generate a traffic system based embedding ④ for a SFE instance ①. We describe the TS-MILP in Section V. The TS-MILP's solution quality and runtime depend heavily on its hyperparameters, the number of epochs N_e and the length of an epoch T_e in the embedding that it is asked to find. TS-ACES uses a hyper-parameter search algorithm ③ to find a pair of hyperparameters which produce a good solution. We describe this algorithm in Section VI.

Once TS-ACES has finished searching for a traffic system based transport plan, it uses that it to construct a cell based transport plan generator $\mathcal{G}_{TS}$. $\mathcal{G}_{TS}$ implements TS-ACES's transport plan. For example, $\mathcal{G}_{TS}$ moves $inB(R_i, T, z_j)$ agents carrying token z_j into road R_i on any epoch $T' = T\%N_e$. It takes $O(|A|)$ time for $\mathcal{G}_{TS}$ to compute the next state in its transport plan. We describe $\mathcal{G}_{TS}$ in Section VII.

V. Generating a Traffic System Based Embedding

In this section, we specify the TS-MILP, the MILP used to construct a traffic system based embedding.

Objective. The TS-MILP maximizes the throughput of its embedding that it generates:

$$\max \sum_{m_i \in M} \lambda(m_i, \text{OUTP}(P))$$

The TS-MILP has 5 categories of constraints.

Machine Configuration Constraints. These constraints specify how the machines in a smart factory can be configured.

Constraint 1. A machine can only be assigned one process.

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

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

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

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

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

Constraint 4. A machine $m_i \in M$ can only run the process $p_j \in P$ at a non-zero rate if it is assigned p_j.

$$\forall\ m_i \in M,\ \forall\ p_j \in P,\ \lambda(m_i, p_j) - \alpha(m_i, p_j) \leq 0.$$

Buffer Conservation Constraints. These constraints ensure that tokens don't appear in and disappear from buffers.

Constraint 5. Each cycle, the number of copies of a token z_j that a machine m_i produces and the number of copies of z_j that agents pick up from its output buffer must be the same.

$$\forall\ m_i \in M - M_{sink},\ \forall\ z_j \in Z,\ \sum_{T=0}^{N_e-1} pk(m_i, T, z_j) = \sum_{p_k \in P} \lambda(m_i, p_k) \cdot \text{NUMOUT}(p_k, z_j) \cdot N_e \cdot T_e.$$

Constraint 6. Each cycle, the number of copies of a token z_j that a machine m_i consumes and the number of copies of z_j that agents deposit in its input buffer must be the same.

$$\forall\ m_i \in M - M_{src},\ \forall\ z_j \in Z,\ \sum_{T=0}^{N_e-1} dp(m_i, T, z_j) = \sum_{p_k \in P} \lambda(m_i, p_k) \cdot \text{NUMIN}(p_k, z_j) \cdot N_e \cdot T_e.$$

Traffic System Conservation Constraints. These constraints ensure that agents and tokens don't appear or disappear. Let $\text{INPUTON}(R_i) \subseteq M$ and $\text{OUTPUTON}(R_i) \subseteq M$ be the sets of machines whose input and output cells are on road R_i.

Constraint 7. The number of agents carrying a non-null token $z_j \in Z$ leaving the road R_i during epoch $T + 1\ \%\ N_e$ is the number of agents carrying token z_j entering road R_i during epoch T, *less* the number of these agents which deposit their

copy of z_j before leaving R_i, *plus* the number of agents which enter R_i during epoch T that pick up a copy of z_j.

$$\forall\ R_i \in \mathcal{R},\ \forall\ z_j \in Z,\ \forall\ T \in [0..N_e - 1],$$
$$outB(R_i, T + 1\ \%\ N_e, z_j) =$$
$$inB(R_i, T, z_j) - \sum_{m_k \in \text{INPUTON}(R_i)} dp(m_k, T, z_j) + \sum_{m_k \in \text{OUTPUTON}(R_i)} pk(m_k, T, z_j).$$

Constraint 8. Similarly, the number of agents carrying a null token leaving the road R_i during epoch $T + 1\ \%\ N_e$ is the number of agents carrying token z_0 entering R_i during epoch T, *less* the number of these agents which pick up a token before leaving R_i, *plus* the number of agents entering R_i during epoch T which deposit a token.

$$\forall\ R_i \in \mathcal{R},\ \forall\ T \in [0..N_e - 1],\ outB(R_i, T + 1\ \%\ N_e, z_0) =$$
$$inB(R_i, T, z_0) - \sum_{m_k, z_j \in \text{OUTPUTON}(R_i) \times Z} pk(m_k, T, z_j) + \sum_{m_k, z_j \in \text{INPUTON}(R_i) \times Z} dp(m_k, T, z_j).$$

Constraint 9. Every agent which enters a junction $J_i \in \mathcal{J}$ during epoch T must leave junction J_i during epoch T.

$$\forall\ J_i \in \mathcal{J},\ \forall\ T \in [0..N_e - 1],\ \forall\ z_j \in Z \cup \{z_0\},$$
$$\sum_{R_k \in \text{EXIT}(J_i)} inB(R_k, T, z_j) = \sum_{R_k \in \text{ENTRY}(J_i)} outB(R_k, T, z_j).$$

Agent Behavior and Team Size Constraints. After an agent changes its cargo, it must move to a new road before changing its cargo again. The TS-MILP does not know how a road's input and output cells are ordered. If a road R_i's output cells come after its input cells, an agent on road R_i will not be able to deposit and then pick up a token (and vice versa). Consequently, the TS-MILP can only rely on agents being able to change their token once per road.

Constraint 10. The number of agents that deposit a copy of token z_j on road R_i during epoch T must be less than the number of agents that enter R_i carrying z_j during epoch T.

$$\forall\ R_i \in \mathcal{R},\ \forall\ z_j \in Z,\ \forall\ T \in [0..N_e - 1],$$
$$inB(R_i, T, z_j) \leq dp(R_i, T, z_j).$$

Constraint 11. The number of agents that pick up token z_j on road R_i during epoch T must be less than the number of agents that enter R_i carrying the null token during epoch T.

$$\forall\ R_i \in \mathcal{R},\ \forall\ z_j \in Z,\ \forall\ T \in [0..N_e - 1],$$
$$inB(R_i, T, z_j) \leq dp(R_i, T, z_j).$$

Constraint 12. The TS-MILP cannot produce a solution which uses more agents than the $|A|$ agents available.

$$\sum_{R_i \in \mathcal{R}} \sum_{z_j \in Z \cup \{z_0\}} outB(R_i, 0, z_j) \leq |A|.$$

Road Capacity Constraints. These constraints limit the number of agents which pass through each road every epoch. Let $totInB(R_i, T) := \sum_{z_j \in Z \cup \{z_0\}} inB(R_i, T, z_j)$ and $totOutB(R_i, T) := \sum_{z_j \in Z \cup \{z_0\}} outB(R_i, T, z_j)$ be the number of agents that enter and leave road R_i on epoch T.

Algorithm 1 TSPLANNER($inst, timer$)

```
sol*, Ne*, Te* ← NULL, NULL, NULL
Ne ← 1
while timer has not expired do
    sol, Te ← PLANFORNUMEPOCHS(inst, timer)
    if sol ≠ NULL ∧ θ(sol) > θ(sol*) then
        sol*, Ne*, Te* ← sol, Ne, Te
    Ne ← Ne + 1
return sol*, Ne*, Te*
```

Constraint 13. Our ILP cannot specify when a road R_i's outbound agents will leave or its inbound agents arrive during any given epoch. Its inbound agents may all arrive before any of its outbound agents leave. Every agent inbound to and outbound from a road R_i during any epoch T must therefore be able to fit on road R_i at the same time.

$$\forall\ R_i \in \mathcal{R}, \forall\ T \in [0..N_e - 1],$$
$$totInB(R_i, T, z_j) + totOutB(R_i, T, z_j) \leq \text{LEN}(R_i).$$

Constraint 14. Each epoch, every agent must move from the queue at the end of one road to the queue at the end of a different road. If there are a lot of agents on a junction's entry roads, it will take them a long time to reach these queues on its exit roads. We ensure that this process doesn't take longer than one epoch by limiting the number of agents which can be on a junction's entry roads at the start of any epoch.

Theorem 1: An agent in a queue on one of a junction $J_i \in \mathcal{J}$'s entry roads at the start of epoch T takes at most:

$$\sum_{R_j \in \text{ENTRY}(J_i)} totOutB(R_j, T) + \max_{R_j \in \text{EXIT}(J_i)} [\text{LEN}(R_j) - totInB(R_j, T)] + 1$$

timesteps to reach a queue on any one of J_i's exit roads.

Proof. Let a_k be the last agent to enter junction J_i on epoch T. Agent a_k has to wait $\sum_{R_j \in \text{ENTRY}(J_i)} totOutB(R_j, T) - 1$ timesteps for the other agents on J_i's entry roads to pass through J_i. Entering J_i takes a_k an additional timestep.

Agent a_k then has to reach the queue at the end of one of J_i's exit roads. Let $R_j \in \text{EXIT}(J_i)$ be the exit road whose queue takes a_k the longest to reach. Agent a_k will be the last agent on road R_j to reach its queue. Thus, when a_k reaches the queue, it will be $totInB(R_i, T) - 1$ agents long. Agents on a road move one cell per timestep until the reach its queue. It therefore takes a_k $\text{LEN}(R_j) - totInB(R_j, T)] + 1$ timesteps to reach road R_j's queue. □

By Theorem 1, we can ensure that all agents reach the queue at the end of their road with the constraint:

$$\forall\ J_i \in \mathcal{J},\ \forall\ R_j \in \text{EXIT}(J_i),\ \forall\ T \in [0..N_e - 1],\ T_e \geq$$
$$\sum_{R_k \in \text{ENTRY}(J_i)} totOutB(R_k, T) + \text{LEN}(R_j) - totInB(R_j, T) + 1.$$

VI. Hyperparameter Search

In this section, we introduce an algorithm which finds good values for the number of epochs N_e and the length of an epoch T_e in the embedding that the TS-MILP constructs.

Number of Epochs. The space of solutions to the TS-MILP with N_e epochs can be very different to the space of solutions

Algorithm 2 PLANFORNUMEPOCHS($inst, N_e, timer$)

$sol^*, T_e^* \leftarrow$ NULL, NULL
$T_e \leftarrow \max_{R_i \in \mathcal{R}}$ LEN$(R_i) + \delta$
$runsSinceBestSol \leftarrow 0$
while $timer$ has not expired and $runsSinceBestSol < \gamma$ **do**
  $sol \leftarrow$ RUNMILP$(inst, N_e, T_e, timer)$
  if $sol \neq$ NULL $\wedge\ \theta(sol) > \theta(sol^*)$ **then**
    $sol^*, T_e^* \leftarrow sol, T_e$
    $runsSinceBestSol \leftarrow 0$
  else
    $runsSinceBestSol \leftarrow runsSinceBestSol + 1$
  $T_e \leftarrow T_e + \delta$

to the TS-MILP with N_e+1 epochs. We would therefore like our solver to try to solve the TS-MILP with as many different values of N_e as possible. We accomplish this using the function TSPLANNER. TSPLANNER is shown in Algorithm 1.

TSPLANNER. Let $timer$ be a timer which tracks the time allocated to TSPLANNER. Let PLANFORNUMEPOCHS($inst$, $N_e, timer$) be a function which returns an optimized solution sol to the TS-MILP for the SFE instance $inst$ with N_e epochs and its epoch length T_e. If the timer expires while this function is running, it returns its current best solution if one exists and (NULL, NULL) otherwise.

The number of variables in the TS-MILP increases with the number of epochs in its transport plan. Solving the TS-MILP with a small number of epochs is thus usually faster than solving it with a large number of epochs. TSPLANNER therefore begins by passing the TS-MILP a small value of N_e. It then solves the TS-MILP with progressively larger values of N_e until the $timer$ runs out, whereupon it returns sol^*, the highest throughput solution that it has found.

Epoch Length. Most epoch lengths do not produce good solutions. If our solver's epoch length is too small, only a couple of agents can pass through a junction every epoch. If its epoch length is too large, agents spend most of their time waiting in a queue for the next epoch to start. We pick epoch lengths heuristically using the function PLANFORNUMEPOCHS, shown in Algorithm 2.

PLANFORNUMEPOCHS. Let RUNMILP($inst, N_e, T_e$, $timer$) be a function which uses the TS-MILP to solve the TS-SFE instance $inst$ with N_e epochs and the epoch length T_e. If the $timer$ expires before it has finished, it returns its current best solution if one exists and NULL otherwise.

PLANFORNUMEPOCHS starts by solving the TS-MILP with an epoch length T_e intended to be smaller than optimal. It then solves the TS-MILP for progressively larger values of T_e. When T_e grows larger than optimal, the throughput of RUNMILP's solutions will stop improving. Let sol^* be RUNMILP's best solution. If γ consecutive epoch length increases pass without sol^* improving, the function terminates.

We increase RUNMILP's epoch length by δ every run. Small values of δ improve PLANFORNUMEPOCHS's solution quality but increase its runtime, large values do the opposite.

TS-MILP needs an epoch length of $\max_{R_i \in \mathcal{R}}$ LEN$(R_i)+1$ or more to allow agents to traverse every road in the traffic system. PLANFORNUMEPOCHS initially runs RUNMILP with an epoch length δ timesteps longer than this minimum.

Algorithm 3 $\mathcal{G}_{TS}(\phi(t), t)$

1: **global** $(Z, P, M, A, \mathcal{J}, \mathcal{R})$
2: **global** $(\boldsymbol{\alpha}, \boldsymbol{\lambda}, \boldsymbol{inB}, \boldsymbol{outB}, \boldsymbol{pk}, \boldsymbol{dp}), N_e, T_e$
3: **global** $canChgTk, enT$
4: **global** $\boldsymbol{inB'}, \boldsymbol{outB'}, \boldsymbol{pk'}, \boldsymbol{dp'}$
5: **global** $(\boldsymbol{\mathcal{I}}(t), \boldsymbol{\mathcal{O}}(t), \boldsymbol{\pi}(t), \boldsymbol{\sigma}(t)) \leftarrow \phi(t)$
6: **global** $\boldsymbol{at}(t) \leftarrow$ GENOCCUPANCYVEC$(\boldsymbol{\pi}(t))$
7: **initialize** $\boldsymbol{\mathcal{I}}(t+1), \boldsymbol{\mathcal{O}}(t+1), \boldsymbol{\sigma}(t+1), \boldsymbol{at}(t+1)$
8: $T \leftarrow \lfloor t/T_e \rfloor$
9: **if** $t \ \% \ T_e = 0$ **then**
10: ADJUSTSTATEFORNEWEPOCH()
11: **for** $J_h \in \mathcal{J}$ **do**
12: $a_i \leftarrow at($CELL$(J_j), t-1)$
13: **if** $a_i \neq$ NULL **then**
14: MOVEAGENTONJUNCTION(J_h, a_i, t, T)
15: **for** $R_h \in \mathcal{R}$ **do**
16: MOVEAGENTSONROAD(R_h, t, T)
17: DEPOSITTOKEN(M, t, T)
18: PICKUPTOKEN(M, t)
19: KEEPTOKEN(t)
20: $\boldsymbol{\pi}(t+1) \leftarrow$ GENPOSITIONVEC$(\boldsymbol{at}(t+1))$
21: **return** $(\boldsymbol{\mathcal{I}}(t+1), \boldsymbol{\mathcal{O}}(t+1), \boldsymbol{\pi}(t+1), \boldsymbol{\sigma}(t+1))$
22: **function** MOVEAGENTONJUNCTION(J_h, a_i, t, T)
23: $z_j \leftarrow \sigma(a_i, t)$
24: $R_k \leftarrow$ RNDELE$(\{R_l \in$ EXIT$(J_h) : inB(R_l, T, z_j) > 0\})$
25: $at($TAIL$(R_k), t+1) \leftarrow a_i$
26: $inB'(R_k, T, z_j) \leftarrow inB'(R_k, T, z_j) - 1$
27: $canChgTk \leftarrow canChgTk \cup \{a_i\}$
28: $enT(a_i) \leftarrow T$

VII. GENERATING A CELL BASED EMBEDDING

TS-ACES uses the traffic system based embedding generated by the TS-MILP to construct the cell based transport plan generator $\mathcal{G}_{TS}$. $\mathcal{G}_{TS}$ takes the smart factory's state $\phi(t)$ at timestep t and generates its state $\phi(t+1)$ at timestep $t+1$. $\mathcal{G}_{TS}$ is shown in Algorithm 3. $\mathcal{G}_{TS}$ begins by loading the following global state variables:

1) The SFE instance that TS-ACES is solving (Line 1) and its traffic system based embedding (Line 2).
2) The set $canChgTk$, which contains every agent which is allowed to change tokens on timestep t (Line 3).
3) The length $|A|$ entry epoch vector enT. The field $enT(a_i)$ stores the epoch during which agent a_i entered its current road (Line 3).
4) The maps $\boldsymbol{inB'}, \boldsymbol{outB'}, \boldsymbol{pk'}$ and $\boldsymbol{dp'}$ (Line 4). The fields $inB'(R_i, T, z_j)$ and $outB'(R_i, T, z_j)$ store the number of additional agents carrying token z_j that $\mathcal{G}_{TS}$ need to move into and out of road R_k during epoch T. The fields $pk'(m_h, T, z_j)$ and $dp'(m_h, T, z_j)$ store the number of additional copies of z_j that need to be picked up from and deposited into m_h's buffers by agents that entered their current road during epoch T.

$\mathcal{G}_{TS}$ then loads the smart factory's state (Line 5).

GENOCCUPANCYVEC. $\mathcal{G}_{TS}$ stores the positions of its agents in an occupancy vector $\boldsymbol{at}(t)$, a length $|C|$ vector (Line 6). The field $at(c_j, t)$ stores the agent on cell c_j on timestep t. If no agent is on c_j on timestep t, $at(c_j, t) =$ NULL.

$\mathcal{G}_{TS}$ then initializes the vectors storing the smart factory's state at timestep $t+1$ (Line 7). Their fields are set to NULL.

ADJUSTSTATEFORNEWEPOCH. If timestep t is the start of

Algorithm 4 MOVEAGENTSONROAD(R_h, t, T)

1: **for** $c_j \in$ REVERSEPATH(R_h) **do**
2: $a_i \leftarrow at(c_j, t)$
3: **if** $a_i \neq$ NULL **then**
4: **if** EXIT$(c_j) =$ HEAD$(R_h) \wedge enT(a_i) = T$ **then**
5: $at(c_j, t+1) \leftarrow a_i$
6: **else if** $at($EXIT$(c_j), t+1) =$ NULL **then**
7: $at($EXIT$(c_j), t+1) \leftarrow a_i$
8: **else**
9: $at(c_j, t+1) \leftarrow a_i$

Algorithm 5 DEPOSITTOKEN(M, t)

1: **for** $m_h \in M$ s.t. $\neg$ISSOURCEM(m_h) **do**
2: $a_i \leftarrow at($INPUTCELL$(m_h), t+1)$
3: **if** $a_i \neq$ NULL $\wedge\ \sigma(a_i, t) \neq z_0 \wedge a_i \in canChgTk$ **then**
4: $z_j \leftarrow \sigma(a_i, t)$
5: **if** $dp'(m_h, enT(a_i), z_j) > 0$ **then**
6: $\sigma(a_i, t+1) \leftarrow z_0$
7: $\mathcal{I}(m_h, t+1) \leftarrow \mathcal{I}(m_h, t+1) \cup \{\{z_j\}\}$
8: $canChgTk \leftarrow canChgTk \setminus \{a_i\}$
9: $dp'(m_h, enT(a_i), z_j) \leftarrow dp'(m_h, enT(a_i), z_j) - 1$

a new epoch T (Line 9), $\mathcal{G}_{TS}$ adjusts its state variables (Line 10) with the function ADJUSTSTATEFORNEWEPOCH. It adds a field $inB'(R_i, T, z_j)$ to inB' for every (road, token) combination. This field is set to $inB(R_i, T\%N_e, z_j)$ since no agents carrying token z_j have moved into road R_i on epoch T. It removes each field from the epoch $T-2$. It performs analogous operations on the maps $\boldsymbol{outB'}$, $\boldsymbol{pk'}$ and $\boldsymbol{dp}$.

MOVEAGENTONJUNCTION. Next, $\mathcal{G}_{TS}$ moves each agent a_i on a junction J_h's cell CELL(J_h) into one of its exit roads with the function MOVEAGENTONJUNCTION (Lines 11-14, 22-28). Let the token z_j be agent a_i's cargo (Line 23). $\mathcal{G}_{TS}$ identifies a road R_k that needs an additional agent carrying z_j (Line 24). It moves a_i to the tail TAIL(R_k) of R_k (Line 25). It adds a_i to the set $canChgTk$ (Line 27) since it is on a new road and updates $\boldsymbol{inB'}$ (Line 26) and enT (Line 28).

MOVEAGENTSONROAD. $\mathcal{G}_{TS}$ then moves each agent a_i on a road R_h with the function MOVEAGENTSONROAD (Algorithm 4). Let cell c_j be the cell that a_i is on (Line 2). Let EXIT(c_j) be cell c_j's exit cell if cell c_j is a road cell. If agent a_i is at the head HEAD(R_h) of road R_h and it entered R_h on the current epoch, it waits on c_j for the next epoch to begin (Lines 4-5). Otherwise, if the cell EXIT(c_j) is currently unoccupied on timestep $t+1$, a_i moves onto EXIT(c_j) (Lines 6-7). If it is, a_i waits on c_j (Line 9).

Let the function REVERSEPATH(R_h) return a list of the cells in road R_h starting at its head and ending at its tail. We process the agents on a road R_h in reverse order of their distance from its head (Line 1). Consequently, whenever we process an agent a_i on a cell c_j on road R_h, we know if an agent in front of a_i on R_h needs to wait at the cell EXIT(c_j).

DEPOSITTOKEN. $\mathcal{G}_{TS}$ then checks if each agent a_i on a machine m_h's input cell INPUTCELL(m_h) carrying a non-null token z_j should deposit its token using the function DEPOSITTOKEN (Algorithm 5). If (a) agent a_i is allowed to deposit its token (Line 3) and (b) machine m_h needs an additional agent which entered its current road on epoch $enT(a_i)$ to deposit a copy of token z_j (Line 5), agent ai's token is placed in m_h's input buffer (Lines 6-7). The variables $canChgTk$ and $\boldsymbol{dp'}$ are then updated (Lines 8-9).

PICKUPTOKEN. $\mathcal{G}_{TS}$ instructs agents to pick up tokens using the function PICKUPTOKEN (Algorithm 3, Line 18). This function operates analogously to DEPOSITTOKEN.

KEEPTOKEN. If an agent neither picks up nor deposits a token, the token that it is holding does not change (Line 19).

Finally, $\mathcal{G}_{TS}$ converts the occupancy vector $\boldsymbol{at}(t+1)$ back into a position vector $\boldsymbol{\pi}(t+1)$ (Line 20).

INITIALIZESF. TS-ACES determines the smart factory's state on timestep $t=0$ using the function INITIALIZESF. Each agent is initialized in a queue at the head of a road. There are $outB(R_i, 0, z_j)$ agents in the queue at the head of road R_i carrying the token $z_j \in Z \cup \{z_0\}$. Agents do not pick up or deposit tokens until they have passed through an intersection for the first time. Each machine's input buffer is initialized with every token that it consumes in a single cycle of epochs. Since every one of these tokens will be replaced by the start of the next cycle, a machine will never run out of tokens. We initialize each machine's output buffer with every token that it emits in a single cycle for similar reasons.

VIII. ANALYSIS

A solution to the SFE problem is complete if, given enough time, it can solve any SFE instance.

Theorem 2: TS-ACES is complete.

Proof. By construction, any traffic system based embedding can be converted into a cell based embedding. TS-ACES is therefore complete if it can construct a traffic system based embedding for any SFE instance. A smart factory's layout graph is strongly connected. There is thus a traffic system based embedding sol for any SFE instance whose transport plan uses a single agent. We will show that TS-ACES will find sol if no better embedding exists.

Let ζ be the finite sequence of roads that sol's agent traverses every cycle. The TS-MILP is always run with a large enough epoch length to allow an agent on any of a junction's input roads to move to any of its output roads as long as there are not other agents on its input roads. Therefore, the TS-MILP can construct an embedding with a single agent that traverses the sequence of roads ζ if it is run with $|\zeta|$ epochs. If TSPLANNER is given enough time, the TS-MILP will be run with $|\zeta|$ epochs. □

IX. EVALUATIONS

Our evaluations investigate the following questions:

- How do TS-ACES's solution quality and runtime compare to those of ACES [6], a state-of-the-art SFE solver?
- Can the TS-MILP and $\mathcal{G}_{TS}$ scale to factories with more than a hundred machines based on realistic scenarios?

Implementation. TS-ACES is written in Python 3.11. The TS-MILP is expressed using the Gurobipy library and solved using the Gurobi MILP solver [17].

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

Scenario Name	Machines	ACES Thr.	TS-ACES Thr.	ACES Runt.	TS-MILP Runt.	$\mathcal{G}_{TS}$ Runt.	TS-ACES Agents Used
Contact Lens Small [14]	24	2	1.14	60	0.31	0.004	102
Contact Lens Large [14]	107	2.14	4.71	60	1.92	0.01	430
Drug Synthesis Small [15]	18	0.2	0.095	60	11.6	0.003	44
Drug Synthesis Large [15]	108	N/A	0.5	60	60	0.02	222
Hard Candy Small [16]	8	0.25	0.14	60	0.16	0.008	28
Hard Candy Large [16]	104	0.8	1.86	60	60	0.02	368

TABLE II

EVALUATION RESULTS

Methodology. TS-ACES is benchmarked on 6 scenarios taken from the food and pharmaceutical manufacturing industry. Companies in these industries frequently want to manufacture different versions of a product. A drug maker, for instances, frequently wants to manufacture different sizes of pill. As a result, these industries benefit from flexible manufacturing. Table II shows the number of machines in each scenario. Each of the contact lens, drug synthesis and hard candy scenarios have 6, 8 and 8 processes. The solvers were allowed to use at most 1000 agents in their solutions. Roads contained at most one input or output cell. Each solver was given a maximum runtime of 60 seconds. TS-ACES was terminated if it failed to improve on its best solution after increasing N_e or T_e $\gamma = 2$ consecutive times. TS-ACES increased T_e by $\delta = 1$ each run.

Results. Table II shows ACES's and TS-ACES's throughput, ACES's, the TS-MILP's and $\mathcal{G}_{TS}$'s runtime, and the number of agents that TS-ACES used on each of our 6 scenarios. Runtimes are measured in seconds. ACES outperforms TS-ACES on each of its small scenarios, since its MILP does not have the traffic system based limitations that the TS-MILP does. ACES, however, generates less than half of TS-ACES's throughput on Contact Lens Large and Hard Candy Large and could not solve Drug Synthesis Large within 60 seconds. The TS-MILP can generate solutions for SFE instances with over 100 machines in 60 seconds. Despite the TS-MILP producing solutions that use 430, 222 and 368 agents, $\mathcal{G}_{TS}$ 0.01, 0.02 and 0.02 seconds to run (averaged over 100 runs). TS-ACES is able to scale to very large SFE instances based on real industrial scenarios.

X. CONCLUSION

In this paper, we addressed the Smart Factory Embedding (SFE) problem with TS-ACES, the Traffic System based Anytime Cyclic Embedding Solver. We analyze TS-ACES and show that it is complete. In future work, we hope to scale TS-ACES further. We also hope to find ways to automatically generate smart factory traffic system layouts.

ACKNOWLEDGEMENT

The research at the University of California, Irvine and University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 2434916, 2346058, 2321786, 2121028, and 1935712 as well as gifts from Amazon Robotics. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

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

[3] 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/

[4] 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.

[5] 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.

[6] C. Leet, A. Sciortino, and S. Koenig, "Jointly Assigning Processes to Machines and Generating Plans for Autonomous Mobile Robots in a Smart Factory," 2025. [Online]. Available: https://arxiv.org/abs/2502.21101

[7] J. Yu and S. LaValle, "Multi-agent Path Planning and Network Flow," *Algorithmic Foundations of Robotics X*, pp. 157–173, 2013.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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