Co-Design of Topology, Scheduling, and Path Planning in Automated Warehouses

Christopher Leet[1], Chanwook Oh[1], Michele Lora[1,2], Sven Koenig[1], Pierluigi Nuzzo[1]
[1]*University of Southern California, Los Angeles, California, USA*
[2]*University of Verona, Verona, Italy*
{cjleet|chanwooo|loramich|skoenig|nuzzo}@usc.edu

***Abstract*—We address the *warehouse servicing problem* (WSP) in automated warehouses, which use teams of mobile agents to bring products from shelves to packing stations. Given a list of products, the WSP amounts to finding a plan for a team of agents which brings every product on the list to a station within a given timeframe. The WSP consists of four subproblems, concerning *what* tasks to perform (task formulation), *who* will perform them (task allocation), and *when* (scheduling) and *how* (path planning) to perform them. These subproblems are NP-hard individually and are made more challenging by their interdependence. The difficulty of the WSP is compounded by the scale of automated warehouses, which frequently use teams of hundreds of agents. In this paper, we present a methodology that can solve the WSP at such scales. We introduce a novel, contract-based design framework which decomposes an automated warehouse into traffic system components. By assigning each of these components a contract describing the traffic flows it can support, we can synthesize a traffic flow satisfying a given WSP instance. Component-wise search-based path planning is then used to transform this traffic flow into a plan for discrete agents in a modular way. Evaluation shows that this methodology can solve WSP instances on real automated warehouses.**

I. Introduction

An *automated warehouse* is a warehouse which uses a team of mobile agents to move products from its shelves to its packing stations. Over the past decade, automated warehouses have become increasingly important to industrial logistics and e-commerce. Today, companies such as Amazon routinely deploy teams of hundreds of agents to manage large warehouse complexes [1]. An automated warehouse must solve the *warehouse servicing problem* (WSP) to operate. In the WSP, we are given a warehouse and a list of products termed a *workload*, and asked to find a plan for a team of agents which brings every product on the list to one of the warehouse's stations within a given time limit. The WSP consists of four interdependent sub-problems:

1) *Task Formulation.* Which shelf and station should a product be taken and brought to?
2) *Task Assignment.* Which tasks should an agent perform?
3) *Scheduling.* When should an agent perform its tasks?
4) *Path Planning.* What path should an agent take through the warehouse to perform each of its tasks?

This research was supported in part by the National Science Foundation (NSF) under Awards 1846524 and 2139982, the Office of Naval Research (ONR) under Award N00014-20-1-2258, the Defense Advanced Research Projects Agency (DARPA) under Award HR00112010003, the 2022 Okawa Research Grant. This project has also received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 894237.

Task assignment, scheduling, and path planning are NP-Hard [2]. Interdependence only increases the challenge of these sub-problems. As a result, existing methodologies for performing task assignment, scheduling, and path planning concurrently [3] do not scale beyond tens of agents. Automated warehouses, however, often have hundreds of agents [1]. Methodologies for each solving subcase of the WSP which omit one or more of these sub-problems have also been proposed [4], but it is unclear whether any can be extended to the full WSP. The question: *"is it possible to solve the WSP at scale?"* is thus open and highly relevant.

We answer the question in the affirmative by developing a novel traffic-system-based methodology for the WSP. In a traffic-system-based warehouse, each shelf and station is linked by a network of roads termed a *traffic system*. The movement of agents through a traffic system is restricted by the rules of the traffic system. Well chosen rules prune the space of potential solutions to the WSP dramatically while preserving efficient solutions. Almost all automated warehouses today use traffic systems to plan for their agents.

In this paper, we present the first formal framework for designing a warehouse traffic system. This framework provides a designer with a library of traffic system *components* and rules for composing these components into a traffic system. There are three types of components: *shelving rows*, which provide access to shelves, *station queues*, which provide access to stations, and *transports*, which connect other components. The traffic flows that a component can support are captured by an assume-guarantee contract termed a *component contract*.

This compositional formalization of a traffic system allows for a compositional formulation of the planning problem. In this formulation, a plan is composed out of a set of *agent cycles*. An agent cycle is a cycle of traffic system components containing a *target shelving row* and *target station queue*. The agents in an agent cycle loop through this cycle of components, carrying products from the target shelving row to the target station queue. We synthesize a plan for a WSP instance by finding a set of agent cycles which solves this instance and satisfies the constraints posed by a given traffic system.

Based on this compositional view of a plan, our methodology for synthesizing a plan for a WSP instance proceeds as follows. The traffic flow required to service the instance's workload within the instance's time limit is captured by an assume-guarantee contract termed a *workload contract*. A traffic flow is found which satisfies the conjunction of this workload contract with the composition of the traffic system's

978-3-9819263-7-8/DATE23/© 2023 EDAA

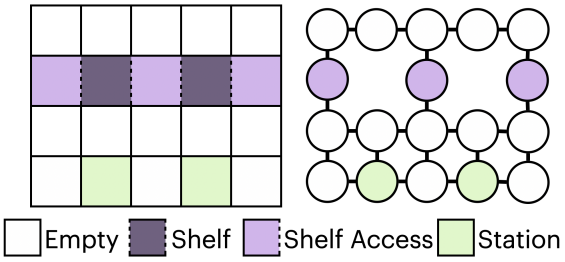

Fig. 1. (left) An example warehouse and (right) its floorplan graph.

component contracts. This traffic flow both solves the WSP instance and can be supported by the traffic system.

This traffic flow is then mapped to a set of agent cycles, which is converted to a plan in a modular fashion. Each timestep, a component moves each agent that it contains toward the next component in its agent cycle. Evaluations show that our methodology can solve WSP instances with hundreds of agents and thousands of tasks on real warehouse layouts in under a minute.

II. Background

A. Prior Work

While this paper is the first to formalize and solve the WSP problem, a related problem termed the Multi-Agent Pickup and Delivery (MAPD) problem has been studied [3]. In MAPD, we are given a set of tasks characterized by a pickup vertex v_i, a delivery vertex v_j, and a release time t, and asked to execute each task by moving an agent from v_i to v_j after time t. Studied variants include lifelong MAPD [5], where a task is not revealed until its release time, and deadline aware MAPD [6], where each task has a deadline. These solvers are not directly applicable to the WSP problem, however, because they do not perform task formulation and have not been shown to scale beyond 50-75 agents.

B. Assume-Guarantee Contracts

We provide an overview of the Assume-Guarantee (A/G) contract framework by starting with the notion of a component. A component M is an element of a design, which can be connected with other components to form larger systems. An A/G contract $\tilde{C} := (V, \tilde{A}, \tilde{G})$ is a triple where V is a set of component variables, $\tilde{A}$ is a set of *assumptions*, that is, the set of behaviors that a component M expects from the environment, and $\tilde{G}$ is the *guarantees*, that is, the set of behaviors promised by the component M if the assumptions hold. A/G contracts can be combined via the *composition* ($\otimes$) or *conjunction* ($\wedge$) operators. Let $\tilde{C}_1$ and $\tilde{C}_2$ be contracts describing the components M_1 and M_2. Taking the composition of $\tilde{C}_1$ and $\tilde{C}_2$ produces a contract which describes the system formed by composing the components M_1 and M_2. Taking the conjunction of $\tilde{C}_1$ and $\tilde{C}_2$ produces a contract which combines the requirements of the two contracts. Further details on the A/G contract framework can be found in the literature [7], [8].

III. Problem Formulation

An automated *warehouse* $W := (G, S, R, \boldsymbol{\rho}, \boldsymbol{\Lambda})$ is represented as a 5-tuple containing the following elements:

- *Floorplan Graph* $G := (V, E)$. An undirected graph representing the warehouse's floorplan where each vertex $v_i \in V$ represents a one-agent-wide cell in the floorplan. There is an edge $(v_i, v_j) \in E$ if and only if (iff) an agent can move from v_i to v_j without traversing another cell.
- *Shelf Access Vertices* $S \subset V$. The vertices in V which an agent can access a shelf from.
- *Station Vertices* $R \subset V$. The vertices in V which workers can access an agent from.
- *Product Vector* $\boldsymbol{\rho} := \langle \rho_1, \ldots, \rho_n \rangle$. The products that warehouse W contains.
- *Location Matrix* $\boldsymbol{\Lambda}$. A $|\boldsymbol{\rho}| \times |S|$ matrix where $\boldsymbol{\Lambda}_{k,l} \in \mathbb{N}$ is the number of units of product ρ_k accessible from shelf access vertex v_l.

Fig. 1 (left) shows a warehouse with two shelves and two stations. Shelves are accessed from the east and west. Fig. 1 (right) shows the floorplan graph $G := (V, E)$ of this warehouse. If $v_{x,y}$ is the vertex in V representing the cell at coordinates (x, y), this warehouse has shelf access vertices $S = \{v_{0,2}, v_{2,2}, v_{4,2}\}$ and stations $R = \{v_{1,0}, v_{3,0}\}$. If the shelves at $(1, 2)$ and $(3, 2)$ contain 10 units of product ρ_1 and ρ_2, respectively, this warehouse has product vector $\boldsymbol{\rho} := \langle \rho_1, \rho_2 \rangle$ and location matrix:

$$\boldsymbol{\Lambda} = \begin{bmatrix} 10 & 10 & 0 \\ 0 & 10 & 10 \end{bmatrix}.$$

Products are carried by a team of mobile *agents* $\boldsymbol{A} := \langle a_1, a_2, \ldots \rangle$. Time in a warehouse is discretized. At each timestep t, an agent a_i has state $(\pi_{i,t}, \phi_{i,t}) \in V \times \{\rho_0\} \cup \boldsymbol{\rho}$, where $\pi_{i,t}$ and $\phi_{i,t}$ are the vertex that agent a_i occupies and the product $\rho_k \in \boldsymbol{\rho}$ that it holds at time t respectively. If agent a_i is not holding a product at time t, $\phi_{i,t} = \rho_0$.

A T *timestep plan* $(\boldsymbol{\pi}, \boldsymbol{\phi})$ for a team of c agents is a pair of $c \times T$ matrices such that $(\pi_{i,t}, \phi_{i,t})$ is the state of agent $a_i \in \boldsymbol{A}$ at time $t \in [1, T]$. A T-timestep plan is *feasible* iff:

(1) An agent a_i moves by 0 or 1 vertices per timestep, that is, the vertex $\pi_{i,t+1}$ that a_i occupies at step $t+1$ is the same as or adjacent to the vertex $\pi_{i,t}$ that a_i occupies at step t:

$$\forall\ t \in [1, T],\ \forall\ a_i \in \boldsymbol{A},\ \pi_{i,t+1} \in \{\pi_{i,t}\} \cup Adj(\pi_{i,t}).$$

(2) Two agents do not collide, that is, two agents $a_i, a_j \in \boldsymbol{A}$ do not occupy the same vertex or traverse the same edge in opposite directions at the same timestep:

$$\forall\ t \in [1, T],\ \forall\ a_i, a_j \in \boldsymbol{A},$$
$$\pi_{i,t} \neq \pi_{j,t} \wedge \neg(\pi_{i,t+1} = \pi_{j,t} \wedge \pi_{i,t} = \pi_{j,t+1}).$$

(3) An agent can only pick up a product ρ_k at a shelf access vertex containing ρ_k and put down a product at a station. Let $\textsc{productsAt}(v)$ be the set of products accessible at vertex v (if $v \notin S$, $\textsc{productsAt}(v) = \emptyset$); then,

$$\forall\ t \in [1, T],\ \forall\ a_i \in \boldsymbol{A},$$
$$\phi_{i,t+1} \in \begin{cases} \{\rho_0\} \cup \textsc{productsAt}(\pi_{i,t}) & \phi_{i,t} = \rho_0 \\ \{\rho_0, \phi_{i,t}\} & \pi_{i,t} \in R \\ \{\phi_{i,t}\} & \text{otherwise.} \end{cases}$$

A *workload* is a vector $\boldsymbol{w} := \langle w_1, \ldots, w_n \rangle$ where w_k is the number of units of product ρ_k which must be transferred

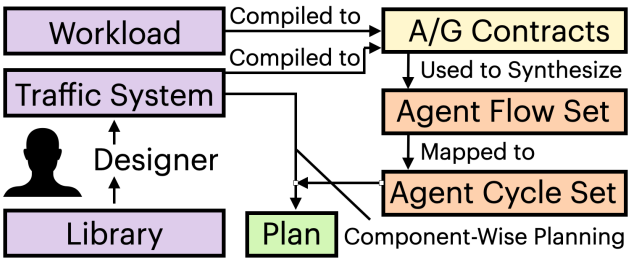

Fig. 2. The high-level workflow of the methodology.

to a station. We say that a T timestep plan *services* workload $\boldsymbol{w}$ iff it is feasible and it transfers w_k units of each product $\rho_k \in \boldsymbol{\rho}$ to the warehouse's stations R.

Problem 3.1 (Warehouse Servicing Problem): Given a warehouse W, a workload $\boldsymbol{w}$, and a timestep limit T, find a T timestep plan containing an arbitrary number of agents which services workload $\boldsymbol{w}$.

IV. Co-Design Methodology

Figure 2 shows the high-level workflow of the methodology. The methodology provides a designer with a framework for designing a traffic system (Subsection IV-A). This framework consists of a set of rules for grouping the vertices in a floorplan graph into traffic system components. The high-level movement of agents through a traffic system is represented by an agent cycle set (Subsection IV-B). An agent cycle set is converted into a plan in a modular fashion (Subsection IV-C).

The methodology finds an agent cycle set for a traffic system which services a workload $\boldsymbol{w}$ within T timesteps as follows. The rate at which each component can accept and emit agents is compiled into an A/G contract. The properties that the pattern of traffic in the traffic system must have to service a workload $\boldsymbol{w}$ within T timesteps are also compiled into an A/G contract (Subsection IV-D). An agent flow set satisfying the conjunction of these contracts is found (Subsection IV-D) and mapped to an agent cycle set (Subsection IV-E).

A. Traffic System Design Framework

An operator can construct a *traffic system* for a warehouse by dividing the vertices in its floorplan graph into disjoint simple paths called traffic system *components*. A component C_i behaves similarly to a one way road. Agents enter a component C_i at its head $\textsc{head}(C_i)$ and exit it from its tail $\textsc{tail}(C_i)$. A component may not contain both shelf access vertices and station access vertices. A component is termed a:

1) *shelving row* if it contains shelf access vertices;
2) *station queue* if it contains station vertices;
3) *transport* if it contains neither.

Every station and shelf access vertex must be contained by a component. Other vertices, however, need not be. Vertices which are not part of any component are termed *unused vertices* since they will not be traversed by any agent.

A component C_i has 1 or 2 *inlet components* $\textsc{inlets}(C_i)$ and 1 or 2 *outlet components* $\textsc{outlets}(C_i)$. Agents must enter C_i from one of its inlets and exit C_i from one of its outlets. There must be an edge in the floorplan graph between the head of a component and the tail of each of its inlets and the tail of a component and the head of each of its outlets. A component with 1 vertex cannot be the inlet or outlet of another component with 1 vertex.

The connections between the components of a traffic system are represented by a directed graph termed a *traffic system* graph $G_s := (V_s, E_s)$. Each vertex in V_s is a component of the traffic system. There is an arc (C_i, C_j) in V_s iff component C_i is one of component C_j's inlets. A traffic system graph must be strongly connected, that is, there must be a path between any two components in a traffic system graph. Since every shelf access and station access vertex is in a component, it follows that there is a way for an agent to move from any shelf access vertex to any station access vertex and *vice versa*.

B. Agent Cycle Set

We synthesize a plan for a traffic system G_s which satisfies a workload $\boldsymbol{w}$ within T timesteps from an *agent cycle set* Σ. An *agent cycle* is a set of b agents associated with a cycle of b components in the traffic system graph G_s. These components must include a *target shelving row* and a *target station queue*. The agents in an agent cycle transfer products from the *target shelving row* to the *target station queue*.

An agent cycle set has a *cycle time* t_c. At timestep $t = 1$, each agent in an agent cycle is positioned on a unique component. Every t_c timesteps, each agent in an agent cycle advances one component. Thus, an agent cycle delivers one product from its target shelving row to its target station queue every t_c timesteps. In Subsection IV-C we show how we find a plan that realizes the agent movement specified by an agent cycle set. (Due to space constraints, we do not specify the exact timesteps at which an agent cycle picks up and drops off products at its target shelving row and target station queue.) In Subsection IV-D, we show how we find an agent cycle set that services a workload $\boldsymbol{w}$ within T timesteps.

C. Realizing an Agent Cycle Set

Let the time interval $[1, T]$ be divided into the $\lfloor T/t_c \rfloor$ *cycle periods* $[1, t_c]$, $[t_c + 1, 2t_c]$, etc. Let $|C_i|$ be the number of vertices in a component C_i and m be the length of the longest component in a traffic system, i.e.,

$$m := \max(|C_i| : C_i \in V_s).$$

The realization algorithm has the following property:

Property 4.1: Our realization algorithm can realize an agent cycle set Σ if it has cycle time $2m$ and there is no component C_i contained by more than $\max(1, \lfloor |C_i|/2 \rfloor)$ agent cycles.

Proof Sketch. The realization algorithm moves an agent at least once every other timestep until the agent advances to the next component in its agent cycle. It therefore takes at most $2m$ timesteps for the realization algorithm to advance any agent in any agent cycle by one component. The realization algorithm assumes, however, that a component C_i can send an agent to its outlets at least once every other timestep. This condition is violated if a component's outlets fill up with agents, preventing them from accepting additional agents. To prevent a component from filling up, the realization algorithm prevents agents from advancing multiple components in a single cycle period. Since each component C_i is in at most $\max(1, \lfloor |C_i|/2 \rfloor)$ agent cycles, no more

Algorithm 1 COMPONENTTIMESTEP(C_i, $t+1$, $\pi_{1,t}$, $\pi_{2,t}$, ...)

1: $t_s \leftarrow$ CYCLEPERIODSTARTT(t)
2: $a_j \leftarrow$ HEAD(AGENTS(C_i, t))
3: **if** $\pi_{j,t} =$ HEAD(C_i) $\wedge$ ADVANCET(a_j) $< t_s$ **then**
4: $\quad k \leftarrow$ CYCLEI(a_j) $+ 1 \mod |$CYCLE(a_j)$|$
5: $\quad$ **if** ACCEPTING(C_i, CYCLE(a_j)[k]) **then**
6: $\quad\quad \pi_{j,t+1} \leftarrow$ TAIL(CYCLE(a_j)[k])
7: $\quad\quad$ CYCLEI(a_j) $\leftarrow k$
8: $\quad\quad$ ADVANCET(a_j) $\leftarrow t+1$
9: **for** $a_j \in$ AGENTS(C_i, t) **do**
10: $\quad v \leftarrow$ NEXT($C_i, \pi_{j,1}$)
11: $\quad$ **if** $v \neq \perp \wedge \neg\exists\ a_k \in$ AGENTS(C_i, t) : $\pi_{k,t} = v$ **then**
12: $\quad\quad \pi_{j,t+1} \leftarrow v$
13: $\quad$ **else**
14: $\quad\quad \pi_{j,t+1} \leftarrow \pi_{j,t}$

than $2 \cdot \max(1, \lfloor |C_i|/2 \rfloor)$ unique agents will occupy any component C_i every cycle period. Since $2 \cdot \lfloor |C_i|/2 \rfloor \leq |C_i|$, a component cannot fill up unless it has a single vertex. If no two single-vertex components are adjacent and each single-vertex component discharges its agent on the first timestep of a cycle period, no component will ever fill up.

Realization Algorithm Initialization. At timestep $t = 1$, we place an agent associated with an agent cycle on an arbitrary vertex of each component that the agent cycle passes through.

Realization Algorithm Timestep. We realize the location of the agents in a traffic system at time $t+1$ from their locations at time t by calling the function COMPONENTTIMESTEP($C_i, t, \pi_{1,t}, \pi_{2,t}, \ldots$), Algorithm 1, on every component C_i in the traffic system.

Let t_s be the time at which the current cycle period starts (Line 1), AGENTS(C_i, t) be a list of the agents in C_i at time t ordered by their distance from HEAD(C_i), and a_j be the agent at the head of this list (Line 2). If a_j is at the head of C_i and ADVANCET(a_j), the timestep when a_j advanced to C_i, was before t_s, we check if a_j can advance to the next component in its agent cycle CYCLE(a_j) (Line 3). A component with a single vertex advances its agent at the start of the cycle period. Otherwise, a component can advance an agent every other timestep in a cycle period.

We compute the index k of the next component in CYCLE(a_j) from the index CYCLEI(a_j) of the current component in CYCLE(a_j) (Line 4). If the component CYCLE(a_j)[k] is accepting agents from C_i, we move agent a_j to this component's tail (Lines 6-8). Next, we move agents within C_i. Let NEXT(C_i, u) be the vertex in C_i following vertex u, if one exists, and $\perp$ otherwise. We move each agent $a_j \in$ AGENTS(C_i, t) to vertex NEXT($C_i, \pi_{j,1}$) if it exists and is not occupied by another agent (Lines 9-14).

D. Synthesizing an Agent Flow Set

Let an *agent flow* $f_{i,j,k} \in \{0\} \cup \mathbb{N}$ be the number of agents that move from component C_i to component C_j carrying product ρ_k every t_c timestep period in a plan. Let an *agent flow set* F be the set containing all of a plan's agent flows:

$$F := \{f_{i,j,k} : (C_i, C_j) \in E_s \wedge \rho_k \in \boldsymbol{\rho}\}.$$

An agent flow set completely describes how agents move between the components in a traffic system each cycle period.

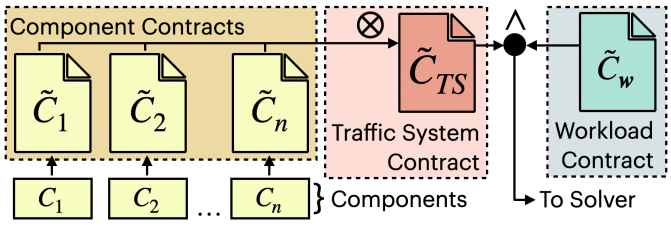

Fig. 3. Synthesizing agent flows using contracts.

We synthesize an agent flow set which services workload $\boldsymbol{w}$ within T timesteps as follows.

A component C_i assumes that the agent flows entering it have certain properties and guarantees that the agent flows leaving it have certain properties. These assumptions and guarantees are compiled into an A/G contract $\tilde{C}_i := (\tilde{A}_i, \tilde{G}_i)$, termed a *component contract* (Fig. 3, yellow). Component contracts are composed into a *traffic system contract* $\tilde{C}_{TS}$ (Fig. 3, red) describing the agent flow sets that the traffic system allows, i.e.,

$$\tilde{C}_{TS} := \bigotimes_{C_i \in V_s} \tilde{C}_i.$$

The properties that a traffic system's agent flow set must have to service workload $\boldsymbol{w}$ within T timesteps are compiled into an A/G contract termed a *workload contract* $\tilde{C}_{\boldsymbol{w}}$ (Fig. 3, blue). We attempt to synthesize an agent flow set which satisfies the conjunction of the traffic system contract and workload contract $\tilde{C}_{\boldsymbol{w}}$ (Fig. 3). If no such agent flow set exists, a plan which services workload $\boldsymbol{w}$ within T timesteps cannot be synthesized using our methodology.

Component Contract Assumptions. As discussed in Subsection IV-C, at most $\max(1, \lfloor |C_i|/2 \rfloor)$ agents may enter any component $C_i \in V_s$ in any cycle period, i.e.,

$$\forall\ C_i \in V_s, \sum_{C_j \in \text{INLETS}(C_i)} \sum_{\rho_k \in \boldsymbol{\rho}} f_{j,i,k} \leq \max\left(1, \left\lfloor \frac{|C_i|}{2} \right\rfloor\right).$$

Since a flow $f_{i,j,k}$ is defined as a non-negative integer, the minimum flow entering any component $C_i \in V_s$ is 0.

Component Contract Guarantees. Let $f^{out}_{i,k} \in \{0\} \cup \mathbb{N}$ be the number of units of product ρ_k transferred from an agent to a station in component C_i each cycle period. If component C_i does not contain a station, $f^{out}_{i,k} = 0$. If component C_i contains stations, $f^{out}_{i,k}$ is between 0 and the number of agents entering component C_i carrying ρ_k each cycle period, that is,

$$f^{out}_{i,k} \in \begin{cases} \{0\} & |C_i \cap R| = 0 \\ [0, \sum_{C_j \in \text{INLETS}(C_i)} f_{j,i,k}] & \text{otherwise} \end{cases}.$$

Let $f^{in}_{i,k} \in \{0\} \cup \mathbb{N}$ be the number of units of product ρ_k transferred from a shelf in component C_i each cycle period. If component C_i does not contain shelves, $f^{in}_{i,k} = 0$. If component C_i contains shelves, $f^{in}_{i,k}$ is limited by the number of units of product ρ_k that component C_i contains. Let UNITSAT(C_i, ρ_k) be the total number of units of product ρ_k available at C_i:

$$\text{UNITSAT}(C_i, \rho_k) := \sum_{v_j \in C_i \cap S} \Lambda_{k,j}.$$

Let $q_c := \lfloor t_c/T \rfloor$ be the number of cycle periods executable in T timesteps. In q_c cycle periods, a component can transfer at most $\text{UNITSAT}(C_i, \rho_k)/q_c$ units of product ρ_k each cycle period. It follows that:

$$f^{in}_{i,k} \in \begin{cases} \{0\} & |C_i \cap S| = 0 \\ [0, \text{UNITSAT}(C_i, \rho_k)/q] & \text{otherwise} \end{cases}.$$

A product can only be transferred to an unburdened agent. Thus, the total number of products transferred to agents in C_i each cycle period is limited by the total number of unburdened agents entering C_i each cycle period.

$$\sum_{\rho_k \in \boldsymbol{\rho}} f^{in}_{i,k} \leq \sum_{C_j \in \text{INLETS}(C_i)} f_{j,i,0}.$$

Agents cannot appear or disappear. Thus, the total flow of agents carrying product ρ_k out of component C_i is:

1) the total flow of agents carrying product ρ_k into C_i,
2) *plus* the total number of unburdened agents given a unit of product ρ_k from a shelf in C_i each cycle period,
3) *minus* the total number of agents which transfer a unit of product ρ_k to a station in C_i each cycle period. Overall,

$$\forall\ C_i, \rho_k \in V_s \times \boldsymbol{\rho},\ \sum_{C_j \in \text{OUTLETS}(C_i)} f_{i,j,k} = \sum_{C_j \in \text{INLETS}(C_i)} f_{j,i,k} + f^{in}_{i,k} - f^{out}_{i,k}.$$

An analogous expression can be written relating the total flow of unburdened agents leaving and entering a component C_i:

$$\forall\ C_i \in V_s,\ \sum_{C_j \in \text{OUTLETS}(C_i)} f_{i,j,0} = \sum_{C_j \in \text{INLETS}(C_i)} f_{j,i,0} + \sum_{\rho_k \in \boldsymbol{\rho}} f^{in}_{i,k} - \sum_{\rho_k \in \boldsymbol{\rho}} f^{out}_{i,k}.$$

Workload Contract. A workload contract $\tilde{C}_{\boldsymbol{w}}$ makes no assumptions. It guarantees that the total number of units of product ρ_k transferred to the warehouse's stations each cycle period is greater than w_k/q_c, where w_k is the demand for ρ_k in workload $\boldsymbol{w}$ and q_c is the number of cycle periods q_c executable in T timesteps, i.e.,

$$\forall\ \rho_k \in \boldsymbol{\rho},\ \sum_{C_i \in V_s} f^{out}_{i,k} \geq \frac{w_k}{q_c}.$$

The above contracts (and associated constraints) are used to generate a formula in propositional logic augmented with arithmetic constraints over the reals, which is solved using a satisfiability modulo theory (SMT) solver to produce the flow rate through every component for every product.

E. Mapping the Agent Flow Set to an Agent Cycle Set

By construction, an agent flow set F has the properties:

Property 4.2: There is a set of paths P_k on G_s for each product ρ_k such that:

1) There are exactly $f^{in}_{i,k}$ paths in P_k beginning at each component $C_i \in V_s$.
2) There are exactly $f^{out}_{i,k}$ paths in P_k ending at each component $C_i \in V_s$.
3) There are exactly $f_{i,j,k}$ paths that contain the edge (C_i, C_j) for each edge $(C_i, C_j) \in E_s$.

Property 4.3: There is a set of paths P_0 on G_s such that:

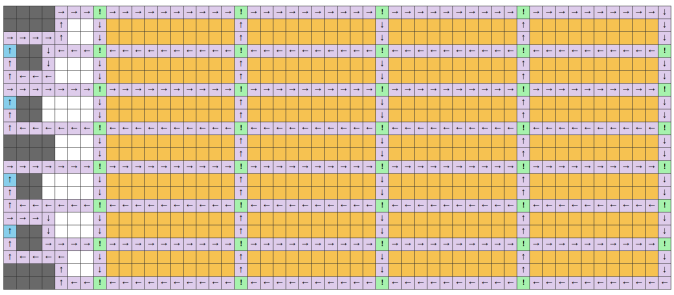

Fig. 4. Fulfillment Center Map.

1) There are exactly $\sum_{\rho_k \in \boldsymbol{\rho}} f^{out}_{i,k}$ paths in P_0 beginning at each component $C_i \in V_s$.
2) There are exactly $\sum_{\rho_k \in \boldsymbol{\rho}} f^{in}_{i,k}$ paths in P_0 ending at each component $C_i \in V_s$.
3) There are exactly $f_{i,j,0}$ paths that contain the edge (C_i, C_j) for each edge $(C_i, C_j) \in E_s$.

Properties 4.2 and 4.3 imply that there is a bijection $B_F : P_0 \to \bigcup_{\rho_k \in \boldsymbol{\rho}} P_k$ for any agent flow set F such that if path $p \in P_0$ is mapped to path $p' \in \bigcup_{\rho_k \in \boldsymbol{\rho}} P_k$, then the head of path p is the tail of path p' and *vice versa*:

$$B_F(p) = p' \Rightarrow \text{HEAD}(p) = \text{TAIL}(p') \wedge \text{HEAD}(p') = \text{TAIL}(p).$$

An agent cycle set Σ is formed from an agent flow set F by turning each pair of paths in the bijection B_F into a cycle.

V. Evaluations

The methodology is implemented as an automatic toolchain. The component and workload contracts are compiled and composed using the CHASE framework [9]. An agent flow set satisfying these contracts is then found using Z3 [10]. Other toolchain components are implemented in Python 3.11. The methodology is evaluated on two real industrial scenarios taken from the literature: a Kiva (now Amazon Robotics) automated warehouse [11] and a package sorting center [12].

Automated Warehouse. A fulfillment center map is characterized by blocks of shelves in its center and stations on its perimeter. The proposed approach is evaluated on two fulfillment center maps, a real map borrowed from our reference scenario [11] with 1071 cells, 560 shelves, 4 stations, and 55 unique products and a synthetic map based on the reference scenario [11] with 793 cells, 240 shelves, 1 station, and 120 products. The real fulfillment center map is depicted in Fig. 4. Shelves are depicted as yellow cells, stations as blue shelves, obstacles as grey cells and empty space as white cells. Single cell components are depicted as green cells. Multi-cell components are depicted as paths of purple cells where each cell has an arrow pointing to the next cell in the component.

Sorting Center. The methodology is also evaluated on a variant of the WSP which takes place in a sorting center [12]. A sorting center sorts packages by destination. A sorting center map is characterized by uniformly placed chutes in its center and bins of unsorted packages on its perimeter. Each chute leads to a shipping counter bound for a unique destination. An agent sorts a package by ferrying it from a bin to the chute associated with its destination. Typically, a bin is modeled as

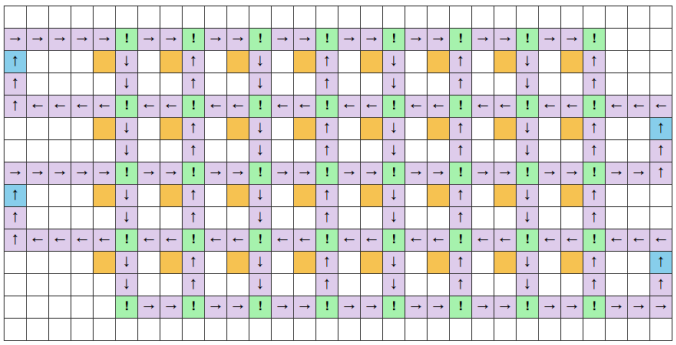

Fig. 5. Sorting Center Map.

TABLE I
BENCHMARKING THE METHODOLOGY ON 9 WSP INSTANCES.

Map	Unique Products	Units Moved	Runtime (s)
Sorting Center	36	160	8.054
	36	320	8.343
	36	480	14.437
Fulfillment (real)	55	550	6.939
	55	825	7.001
	55	1100	8.014
Fulfillment (syn.)	120	1200	65.880
	120	1320	65.886
	120	1440	67.825

having an unlimited number of packages. The goal is to fill each shipping container before it is scheduled to depart.

This problem is modeled as an WSP instance as follows. Let the ith chute be modeled as a shelf containing an arbitrary amount of the product ρ_i. Let each bin of unsorted products be modeled as a station. Let n_i be the number of packages that must be brought to the ith chute. An instance of the WSP is generated where the demand for product ρ_i is n_i. Solving this WSP instance produces an agent cycle set where n_i units of product ρ_i are brought from the ith chute to the bins of unsorted products. Swapping the locations where agents pick up and drop off products generates the desired solution.

The sorting center evaluation is conducted on a map based on [12]. This map contains 406 cells, 32 chutes, and 4 bins. It is depicted in Fig. 5.

Experimental Hardware. Each evaluation was performed on an 2.6 GHz Intel(R) Core i7-10705H CPU with 32 GB of RAM in a Ubuntu 20.04 VM run on Windows 11.

Results. The three WSP instances were generated on each map. For each WSP instance, Table I lists:

1) the number of unique products placed in the warehouse.
2) the total units of product moved to a station.
3) the time required to generate an agent flow set (the time required to convert an agent flow set into a plan is small).

The length T of a plan for each WSP instance was limited to 3,600 timesteps. Solver runtime was limited to 1 hour.

The proposed methodology was able to solve an WSP instance where more than 1400 products had to be moved to a station in just over a minute. We benchmarked the methodology on this instance against Iterated EECBS [4], a state-of-the-art search-based lifelong path planner as follows. Iterated EECBS was given the start position of each agent in our solution. It was asked to find a plan where each agent visited the same sequence of shelves and stations as it did in our solution. Iterated EECBS failed to terminate after an hour.

The runtime of traditional multi-agent path planners is exponential to the size of their team of agents, the number of locations that an agent has to visit, and the average distance between the locations that an agent has to visit. Our proposed methodology, by contrast, is exponential in the number of components in the traffic system (finding a set of agent flows is reducible to the Integer Linear Programming problem). As a result, our methodology outscales state-of-the-art path planners. Additionally, our methodology is relatively insensitive to the number of products in a WSP instance. On both the sorting center and fulfillment center, doubling workload size increased runtime by less than 10%.

VI. CONCLUSION

In this paper we introduce the first methodology for solving the warehouse servicing problem. This methodology provides a designer with a formal framework, based on assume-guarantee contracts, for designing a warehouse traffic system. We show that this methodology can solve WSP instances with more than 1,000 products in just over a minute. In future work, we hope to find an optimal solution to the WSP. In particular, we intend to examine ways to iteratively refine our satisficing solution into an optimal solution.

REFERENCES

[1] E. Ackerman, "Amazon Uses 800 Robots to Run This Warehouse," https://spectrum.ieee.org/amazon-introduces-two-new-warehouse-robots, IEEE Spectrum, 2021, accessed: 16-May-2022.

[2] J. D. Ullman, "NP-Complete Scheduling Problems," *Journal of Computer and System Sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[3] M. Liu, H. Ma, J. Li, and S. Koenig, "Task and Path Planning for Multi-Agent Pickup and Delivery." in *Proc. of the International Joint Conference on Autonomous Agents and Multiagent Systems Search*, 2019, pp. 11 560–11 565.

[4] J. Li, W. Ruml, and S. Koenig, "EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding," *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 12 353–12 362, 2021.

[5] H. Ma, W. Honig, T. K. Satish, N. Ayanian, and S. Koenig, "Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery." in *Proc. of the AAAI Conference on Artificial Intelligence*, 2019, p. 7651–7658.

[6] X. Wu, Y. Liu, X. Tang, W. Cau, F. Bai, G. Khonstantine, and G. Zhao, "Multi-Agent Pickup and Delivery with Task Deadlines." in *Proc. of the International Symposium on Combinatorial Search*, 2021, p. 206–208.

[7] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, K. G. Larsen *et al.*, "Contracts for System Design," *Foundations and Trends in Electronic Design Automation*, vol. 12, pp. 124–400, 2018.

[8] P. Nuzzo, A. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa, "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems," *Proc. IEEE*, vol. 103, no. 11, Nov. 2015.

[9] P. Nuzzo, M. Lora, Y. A. Feldman, and A. L. Sangiovanni-Vincentelli, "CHASE: Contract-based Requirement Engineering for Cyber-Physical System Design," in *Proc. of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 839–844.

[10] L. d. Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. of the International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[11] P. R. Wurman, R. D'Andrea, and M. Mountz, "Co-ordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses," in *Proc. of the AAAI Conference on Artificial Intelligence*, 2007, p. 1752–1760.

[12] Q. Wan, C. Gu, S. Sun, M. Chen, H. Huang, and X. Jia, "Lifelong Multi-Agent Path Finding in a Dynamic Environment," in *The International Conference on Control, Automation, Robotics and Vision*, 2018, p. 875–882.