PREPRINT - Accepted at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2023.

Task Assignment, Scheduling, and Motion Planning for Automated Warehouses for Million Product Workloads

Christopher Leet[1], Chanwook Oh[1], Michele Lora[1,2], Sven Koenig[1], Pierluigi Nuzzo[1]

***Abstract*— We address the Warehouse Servicing Problem (WSP) in automated warehouses, which use teams of mobile robots to move products from shelves to packaging stations. Given a list of products, the WSP amounts to finding a motion plan which brings every product on the list from a shelf to a packaging station within a given time limit. The WSP consists of four subproblems, namely, deciding where to source and deposit a product (task formulation), who should transport each product (task assignment) and when (scheduling) and how (motion planning). These problems are NP-Hard individually and made more challenging by their interdependence. The difficulty of the WSP is compounded by the scale of automated warehouses, which use teams of hundreds of agents to transport thousands of products. In this paper, we present Contract-based Cyclic Motion Planning (CCMP), a novel contract-based methodology for solving the WSP at scale. CCMP decomposes a warehouse into a set of traffic system components. By assigning each component a contract which describes the traffic flows it can support, CCMP can generate a traffic flow which satisfies a given WSP instance. CCMP then uses a novel motion planner to transform this traffic flow into a motion plan for a team of robots. Evaluation shows that CCMP can solve WSP instances taken from real industrial scenarios with up to 1 million products while outperforming other methodologies for solving the WSP by up to 2.9×.**

I. Introduction

An *automated warehouse* uses a team of mobile robots to transfer products from its shelves to its packaging stations. Over the last decade, automated warehouses have become widely used in industrial logistics and e-commerce. Today, companies such as Amazon use teams of hundreds of robots to transport products across large warehouse complexes [1]. To orchestrate an automated warehouse, a warehouse operator must solve the *Warehouse Servicing Problem* (WSP). In the WSP, we are given a *warehouse layout* and a list of products termed a *workload* and asked to find a plan for a team of robots which brings every product on the list to a packaging station within a given timeframe. The WSP consists of four interdependent subproblems:

1) *Task Formulation.* What shelf should each product be sourced from, and what station should it be taken to?
2) *Task Assignment.* What tasks should a robot perform?
3) *Scheduling.* When should a robot perform its tasks?
4) *Motion Planning.* How should a robot execute its tasks?

The WSP is challenging because of the interdependence of its subproblems and because of the scale of automated warehouses. Task formulation, task assignment, and scheduling are discrete space problems, while motion planning is a continuous space problem. The interdependence of these subproblems requires techniques from dissimilar domains to be combined. Task assignment and scheduling are NP-Hard while motion planning is PSPACE-Hard [2]. Thus, solving these problems at scale is difficult. Their interdependence only increases this challenge.

Due to these challenges, existing methodologies that perform task assignment, scheduling, and motion planning concurrently can only scale beyond tens of robots and tens of tasks at the cost of conservative, discrete-space abstractions of the continuous dynamics [3], [4]. Automated warehouse operators, however, routinely use teams of hundreds of robots to service workloads with 100,000s of products [1]. The question: *"Is it possible to solve the WSP at scale?"* is thus both open and highly relevant.

We answer this question in the affirmative by introducing a new methodology for solving the WSP: *Contract-based Cyclic Motion Planning* (CCMP). CCMP uses a traffic system to structure the high-level movement of traffic in a warehouse. The structure provided by a traffic system allows CCMP to construct a motion plan based on a *traffic cycle set*. A *traffic cycle* associates a cycle of roads in a traffic system with a set of robots. These robots circle the traffic cycle, picking up products from its *source road* and depositing them at its *destination road*. A cyclic approach is appropriate for the WSP since it consists of a large number of similar tasks.

CCMP computes a traffic cycle set using *Assume-Guarantee (A/G) contracts* [5], [6], [7]. The constraints that the traffic system framework places on the rate that robots can enter and leave each junction are compiled into A/G contracts. The rate that each product must be deposited at the warehouse's stations is also compiled into an A/G contract. A logical solver is used to find a traffic flow that satisfies these contracts. This traffic flow is then decomposed into a set of traffic cycles. CCMP then finds a motion plan for each robot which moves it around its traffic cycle.

Prior work [4] has applied contract-based planning to a variant of the WSP where a warehouse is modeled as a grid and robots as idealized agents which move between grid cells using discrete actions. CCMP extends this methodology to robots with realistic kinematics with the following novelties:

1) A formal framework for augmenting an automated warehouse modeled in continuous space with a traffic system.
2) A system of A/G contracts capturing traffic patterns that this traffic system can support.

[1]*University of Southern California, Los Angeles, California, USA.* {cjleet|chanwooo|skoenig|nuzzo}@usc.edu

[2]*University of Verona, Verona, Italy* – michele.lora@univr.it

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 Okawa Research Grant, and Siemens under the USC Center for Autonomy and Artificial Intelligence. The 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.

3) A motion planner which can convert a traffic cycle set synthesized using these contracts into a motion plan.

The evaluation shows that CCMP can solve instances of the WSP taken from real industrial scenarios whose workload contains 1 million products. CCMP outperforms existing methodologies by up to 2.9×.

II. Related Work

This paper presents the first methodology that concurrently performs task formulation, task assignment, scheduling, and motion planning in automated warehouses. Prior work, however, has studied sub-cases of this problem.

Motion planning for teams of robots has been extensively studied. Prior work can be grouped into three classes: fully centralized planners, such as prioritized planners [8], decentralized planners, such as collision-avoidance-based planners [9], and partially decentralized planners, such as Probabilistic Roadmap Planning [10].

Many variants of the multi-robot task assignment and scheduling problem have also been studied. We limit our focus to the single task (ST) robots, single robot (SR) tasks, time extended allocation (TA) problem since it closely relates to our work. Approaches to the ST-SR-TA problem include auction-based methods [11], where tasks are auctioned to robots through a bidding process, optimization-based methods [12], which formulate the SR-ST-TA problem as a mixed-integer linear programming problem, and trait-based methods [3], which encodes task requirements in terms of traits. Planners which perform motion planning and task assignment jointly include reactive motion planning [13], symbolic planning [12] and tree-based planning [3]. None of these approaches, however, incorporates task formulation or has been scaled to a million tasks.

Multi-Agent Path Finding (MAPF) is a highly related field. MAPF planners model a workspace such as a warehouse as a grid graph and robots as idealized agents which move between vertices with discrete actions. Extensive work has been done on MAPF [14]. Simple MAPF variants include lifelong MAPF [15], where an agent must visit a sequence of goal vertices. A highly scalable contract-based methodology for solving the WSP within a path planning framework, Contract-based Cyclic Path Planning CCPP [4], has been proposed. Converting a plan for idealized robots into a plan for realistic robots, however, degrades the quality of the plan substantially. One highly related MAPF problem, the Multi-Agent Pickup and Delivery (MAPD) problem has also been studied [16]. Solved variants include lifelong MAPD [17], where a task is not revealed until its release time, and deadline-aware MAPD [18], where each task has a deadline. None of these planners, however, have been shown to scale far beyond 100 agents.

III. Problem Formulation

Warehouse. A *warehouse* $W := (\mathcal{W}, \boldsymbol{S}, \boldsymbol{B}, \boldsymbol{\rho}, \boldsymbol{\lambda})$ is represented as a 5-tuple containing the following elements:

1) *Floorplan* $\mathcal{W} \subset \mathbb{R}^2$. The open space in warehouse W.
2) *Shelves* $\boldsymbol{S} := \langle S_1, S_2, \ldots \rangle \subset (\mathbb{R}^2 - \mathcal{W})^{|\boldsymbol{S}|}$. The ith shelf in warehouse W occupies the space S_i. A shelf is an obstruction, and so, for all $S_i \in \boldsymbol{S}$, $S_i \subset \mathbb{R}^2 - \mathcal{W}$. Two shelves may not overlap. The open space that the products in shelf S_i can be accessed from is termed shelf S_i's *shelf access space* and denoted $\text{ACCESS}(S_i) \subset \mathcal{W}$.
3) *Stations* $\boldsymbol{B} := \langle B_1, B_2, \ldots \rangle \subset \mathcal{W}^{|\boldsymbol{B}|}$. The ith station in warehouse W occupies the space B_i. A station is navigable, and so, for all $B_i \in \boldsymbol{B}$, $B_i \subset \mathcal{W}$. Two stations may not overlap.
4) *Products* $\boldsymbol{\rho} := \langle \rho_1, \rho_2, \ldots \rangle$. A list of the products in warehouse W.
5) *Location Matrix* $\boldsymbol{\lambda}$. A $|\boldsymbol{\rho}| \times |\boldsymbol{S}|$ matrix where $\lambda_{k,l} \in \mathbb{N}_0$ is the number of units of product ρ_k in shelf S_l.

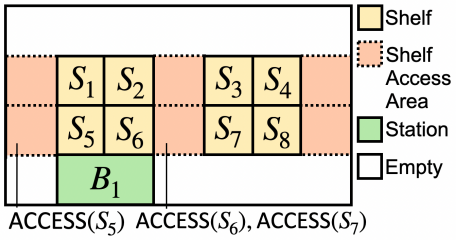

Fig. 1. An example warehouse.

Example. Fig. 1 shows a warehouse with 8 shelves $\boldsymbol{S} = \langle S_1, \ldots S_8 \rangle$ and 1 station $\boldsymbol{B} = \langle B_1 \rangle$. Shelves are accessed from the east and the west. The shelf access areas of (a) shelf S_2 and S_3 and (b) shelf S_6 and S_7 overlap.

Robots. Products are moved through an automated warehouse by a team of mobile *robots* $\boldsymbol{r} := \langle r_1, r_2, \ldots \rangle$. A robot r_i is modeled as a rigid disk of radius b which moves in the plane. The *configuration space* $\mathcal{C}$ of a robot is thus $\mathbb{R}^2 \times \mathbb{S}$, where $\mathbb{S}$ is the set of angles. A *configuration* of robot r_i is defined as $q_i := (x_i, y_i, \theta_i)$. The position (x_i, y_i) of a robot r_i uses the center of its disk as a reference point. The motion of a robot r_i is modeled as obeying first-order differential constraints. A robot r_i thus has state $\boldsymbol{z}_i := (q_i, \dot{q}_i)$ where $\dot{q}_i$ is the derivative of the robot's configuration q_i. The state space of a robot is $\mathcal{Z} := \mathbb{R}^2 \times \mathbb{S} \times \mathbb{R}^3$.

A robot r_i is actuated with the *action vector* $\boldsymbol{u}_i \in \mathcal{U} \subset \mathbb{R}^\alpha$. The *state transition equation* $\dot{\boldsymbol{z}}_i = \tilde{f}(\boldsymbol{z}_i, \boldsymbol{u}_i)$ describes how applying an action vector $\boldsymbol{u}_i \in \mathcal{U}$ to a robot r_i in state $\boldsymbol{z}_i$ changes the state of robot r_i. Robots are homogeneous, that is, all robots have the same *action space* $\mathcal{U}$ and *state transition function* $\tilde{f}$. The dynamics of a robot are position invariant and load invariant, that is, the state transition function of a robot r_i neither depends on the position of the robot nor the product it is carrying.

Plan. A length T *plan* $(\boldsymbol{s}, \tilde{\boldsymbol{u}})$ for a team of robots is a pair of vectors where $s_i \in \boldsymbol{s}$ is the *start state* assigned to robot r_i and $\tilde{u}_i \in \tilde{\boldsymbol{u}}$ such that $\tilde{u}_i : [1, T] \to \mathcal{U}$ is the *action trajectory* assigned to robot r_i. The action vector applied to robot r_i at time $t \in [1, T]$ is denoted $\tilde{u}_i(t)$. The *state trajectory* that a length T plan moves robot r_i along is denoted $\tilde{z}_i : [1, T] \to \mathcal{Z}$. The state and configuration of robot r_i at a particular time $t \in [1, T]$ are denoted $\tilde{z}_i(t)$ and $q_i(t) := (x_i(t), y_i(t), \theta_i(t))$ respectively. A length T plan is *safe* if and only if:

(1) a robot never collides with the environment, that is, a robot's disk always occupies free space:

$$\forall\, t \in [0, T],\ \forall\, r_i \in \boldsymbol{r},$$
$$\{(x', y') : ||\langle x' - x_i(t), y' - y_i(t) \rangle|| \leq b\} \subset \mathcal{W}.$$

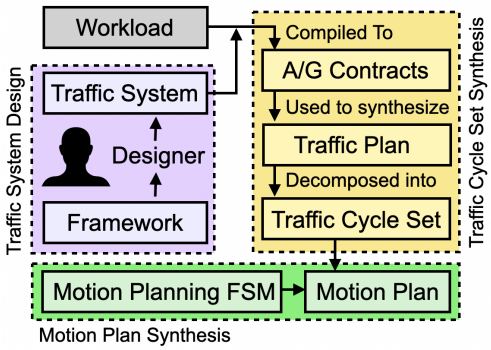

Fig. 2. High-level workflow.

(2) robots never collide with each other. Two robots collide if their centers are distance b or less apart.

$$\forall\ t \in [0, T],\ \neg\exists\ r_i, r_j \in \boldsymbol{r}: \\ ||(x_i(t), y_i(t)) - (x_j(t), y_j(t))|| \leq b.$$

Warehouse Servicing Problem. A *workload* $\boldsymbol{w} := \langle w_1, \ldots, w_n \rangle$ is a vector where w_k indicates the units of product ρ_k that must be brought to a station. A robot brings a product ρ_k to a station by (a) moving to the shelf access area of a shelf containing product ρ_k, (b) picking up a unit of product ρ_k, (c) moving to a station, and (d) depositing the product. A robot can only carry one unit of product at a time. Picking up a product ρ_k from a shelf S_l decrements the units $\lambda_{k,l}$ of product ρ_k available at S_l. A length T plan *services* workload $\boldsymbol{w}$ if and only if it is safe and it brings w_k units of each product $\rho_k \in \boldsymbol{\rho}$ to the warehouse's stations.

Problem 3.1 (Warehouse Servicing Problem): Given a warehouse W, a workload $\boldsymbol{w}$, and a time limit T, find a plan of length T or less with an arbitrary number of robots which services workload $\boldsymbol{w}$.

IV. Overview

We synthesize a motion plan for a given WSP instance using the workflow shown in Fig. 2. This workflow has three stages: traffic system design, traffic cycle set synthesis, and motion plan synthesis.

Traffic System Design. To use CCMP, warehouse operators must construct traffic systems for their warehouse. A traffic system consists of *junctions*, connected by a set of line segments called *roads*. The center of a robot must always be on a road or a junction.

A road behaves similarly to a one-way road in a city. Robots enter a road at its inlet junction and move without backtracking to its outlet junction. A junction behaves similarly to an all-way junction in a city. Each junction is associated with a lock. A robot can only enter a junction when it holds that junction's lock. Locks are awarded to the robot that has been waiting for the longest, breaking ties arbitrarily. Figure 3 shows a traffic system for the example warehouse. Junctions are depicted as black dots, and roads are depicted as black arrows. A road's arrowhead indicates the direction that robots must move.

CCMP's traffic system framework formally states rules that an operator must obey when designing a traffic system layout and that a robot must obey when moving through a traffic system. These rules prevent collisions, ensure that a robot can access any shelf or station in the warehouse from any point in the traffic system, and ensure that no robot has to wait indefinitely to access a junction or road.

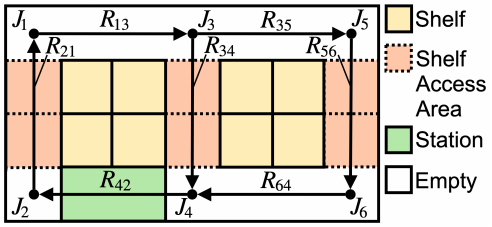

Fig. 3. A traffic system for the example warehouse.

Traffic Cycle Set Synthesis. CCMP solves a WSP instance by constructing a *traffic cycle set* $\Sigma := \{\sigma_1, \sigma_2, \ldots\}$ which satisfies that instance. A *traffic cycle*:

$$\sigma_i := (\text{ROBOTS}(\sigma_i), \text{ROADS}(\sigma_i), \text{SRC}(\sigma_i), \text{DST}(\sigma_i), \rho(\sigma_i))$$

associates a set of b robots $\text{ROBOTS}(\sigma_i)$ with a cycle of b roads $\text{ROADS}(\sigma_i)$. One of these roads is designated as traffic cycle's *source road* $\text{SRC}(\sigma_i)$ and another as its *destination road* $\text{DST}(\sigma_i)$. A traffic cycle's robots circle its cycle of roads, picking up units of product $\rho(\sigma_i)$ from its source road and depositing them at its destination road.

A traffic cycle set is associated with a cycle time t_c. Every t_c timesteps, each robot advances one road. Thus a traffic cycle set delivers a unit of product $\rho(\sigma_i)$ from its source road's shelves to its destination road's stations once every t_c timesteps. A traffic cycle set's cycle time must be long enough for every robot to advance on the road while possibly picking up or depositing a product. An overly long cycle time, however, degrades solution quality. The optimal cycle time length depends on the length of a road and the kinodynamics of the robots.

A traffic cycle set is computed as follows. First, CCMP computes a *traffic plan*. A traffic plan lists the number of robots carrying each product $\rho_i \in \boldsymbol{\rho}$ that enter and leave each junction and road each cycle period. A traffic plan is computed using A/G contracts. The constraints the traffic system framework sets on the number of robots that can enter and leave each road and junction each cycle period are compiled into contracts. The number of units of each product that must be deposited at the warehouse's stations each cycle period to satisfy the WSP is also compiled into a contract. A traffic plan which satisfies these contracts is computed and then decomposed into cycles.

Motion Plan Synthesis. A motion plan is synthesized for each robot using a finite state machine which moves it around its traffic cycle at the rate of one road per cycle period, picking up and depositing products where appropriate.

V. Traffic System Design

The traffic system framework formally defines a traffic system, describes constraints that its layout must follow, and specifies rules that a robot in a traffic system must obey.

Traffic System. A warehouse *traffic system* $(\boldsymbol{J}, \boldsymbol{R})$ is a set $\boldsymbol{J} := \{J_1, J_2, \ldots\}$ of points called *junctions* connected by a set $\boldsymbol{R}$ of line segments called *roads*. Each road starts at an *inlet junction* and ends at an *outlet junction*. Junctions are labeled numerically: $\boldsymbol{J} := \{J_1, J_2, \ldots\}$. Roads are labeled

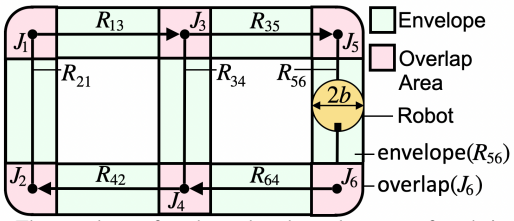

Fig. 4. The envelope of each road and overlap area of each junction in the example traffic system.

with their inlet junction and outlet junction. The road R_{ij} has inlet junction J_i and outlet junction J_j. Robots are confined to the traffic system. At any time t, the center $(x_i(t), y_i(t))$ of a robot r_i must lie on a junction or road:

$$\forall\ r_i \in \boldsymbol{r},\ \forall\ t \in [0,T],\ (x_i(t), y_i(t)) \in \bigcup_{J_j \in \boldsymbol{J}} J_j \cup \bigcup_{R_{kl} \in \boldsymbol{R}} R_{kl}.$$

Layout Constraints. Traffic system layout constraints ensure that a robot cannot collide with the environment or a robot on a different road and can access every shelf and station. Collisions with the environment are prevented by constraining the position of a road by its *envelope*. The envelope $\textsc{envelope}(R_{ij})$ of a road R_{ij} is the set of points that a robot traversing the road passes through. Since a robot is a disc of radius b, the envelope of a road R_{ij} contains every point distance b or less from a point on the road:

$$\textsc{envelope}(R_{ij}) := \bigcup_{(x,y) \in R_{ij}} \{(x', y') : ||\langle x' - x, y' - y \rangle|| \leq b\}.$$

The envelope of a road must only contain open space:

$$\forall\ R_{ij} \in \boldsymbol{R},\ \textsc{envelope}(R_{ij}) \subset \mathcal{W}.$$

The envelopes of two roads which do not share a junction may not overlap. Let the *overlap area* $\textsc{overlap}(J_i)$ of junction J_i be the set of points contained by two or more roads with an endpoint at junction J_i. If $\bigcup H$ is the union of the elements in the set H, we have:

$$\textsc{overlap}(J_i) := \bigcup \{\textsc{envelope}(R_{jk}) \cap \textsc{envelope}(R_{lm}) : R_{ji}, R_{ik} \in \textsc{rIn}(J_i) \cup \textsc{rOut}(J_i)\}.$$

Fig. 4 illustrates the envelope of each road and the overlap area of each junction in the example traffic system when it is populated by disc-shaped robots with radius b. Envelopes are depicted in blue and overlap areas in red.

To prevent robots from colliding in a junction overlap area, each junction is associated with a lock. To enter a junction's overlap area, a robot must hold the junction's lock. Only one robot may hold a junction's lock at a time. To avoid robots waiting to access a junction indefinitely, a junction's lock is awarded on a first come first served basis. Collectively, these constraints ensure that a robot cannot collide with the environment or a robot on a different road.

A robot can access a station B_j from any point on a road which intersects the station and is not in a junction overlap area. There must therefore be an intersection point outside of a junction overlap area between any station B_j in a warehouse and some road R_i in a traffic system:

$$\forall\ B_j \in \boldsymbol{B}, \exists\ R_i \in \boldsymbol{R},\ (B_i \cap R_j) \not\subseteq \bigcup_{J_k \in \boldsymbol{J}} \textsc{overlap}(J_k).$$

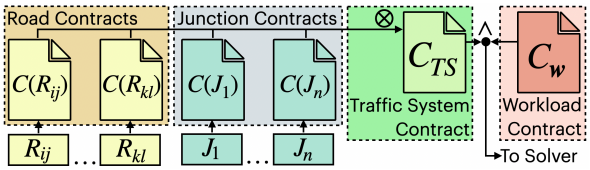

Fig. 5. Synthesizing a traffic plan using contracts.

An analogous rule holds true for each shelf access area. A traffic system must provide a way to reach any road or junction from any other road or junction. A robot can thus travel to any shelf or station in the traffic system.

Robot Motion Constraints. The framework's robot motion constraints prevent robots on the same road from colliding. They state that a robot must remain a safe distance, d_{sep}, behind the robot in front. Let the capacity $\textsc{capacity}(R_{ij})$ of road R_{ij} be the maximum number of agents that can occupy road R_{ij} when: (a) all robots are at least distance d_{sep} from their neighbors and (b) no robot intersects a junction overlap area. If the road is in n traffic cycles, at most $2n$ robots may occupy the road during each cycle period. To prevent collisions, a road may not be in more than $\lfloor \textsc{capacity}(R_{ij})/2 \rfloor$ traffic cycles. All roads must have a capacity of at least 2.

VI. Traffic Cycle Set Synthesis

A traffic cycle set is computed by generating a traffic plan and then decomposing it into a set of traffic cycles.

Traffic Plan. A traffic plan $(\boldsymbol{n^{in}}, \boldsymbol{n^{out}})$ specifies the number of robots which enter and leave each junction and road carrying each product during each cycle period. Let n^{in}_{ijk} and n^{out}_{ijk} be the number of robots that enter and leave road R_{ij} carrying product ρ_k each cycle period, respectively. Robots which are not carrying a product are modeled as carrying the *null product* ρ_0. It follows that the number of robots which enter and leave junction J_i carrying product ρ_k each cycle period is:

$$\sum_{R_{ji} \in \textsc{rIn}(J_i)} n^{out}_{jik} \text{ and } \sum_{R_{ij} \in \textsc{rOut}(J_i)} n^{in}_{ijk}.$$

Traffic Plan Synthesis. A traffic plan is synthesized using A/G contracts. A road R_{ij} assumes that the number of robots entering it each cycle period is constrained. A road R_{ij} guarantees that the number of robots exiting it each cycle period is constrained. These constraints are compiled into an A/G contract $C(R_{ij})$ termed a *road contract* (Fig. 5, yellow). A junction J_i also makes assumptions and provides guarantees about the number of robots entering and leaving it each cycle period. These assumptions and guarantees are compiled into a *junction contract* $C(J_i)$ (Fig. 5, blue). Each road and junction contract is composed into a *traffic system contract* C_{TS} (Fig. 5, green) which constrains the types of traffic plan that a traffic system can support, i.e.,

$$C_{TS} := \Big[\bigotimes_{J_i \in \boldsymbol{J}} C(J_i)\Big] \otimes \Big[\bigotimes_{R_{ij} \in \boldsymbol{R}} C(R_{ij})\Big].$$

The number of units of each product ρ_k that the team of robots must deposit at a station during each cycle period in order to service a given WSP instance is compiled into a

workload contract $C_{\boldsymbol{w}}$ (Fig. 5, red). A traffic plan which satisfies the conjunction of the traffic system contract and workload contract is synthesized. If no such traffic plan exists, the given WSP instance cannot be solved by CCMP.

Junction Contract. Let $t_x(J_i)$ be the maximum time that it takes a robot to cross a junction. Recall that t_c is the length of a cycle period. A junction contract assumes that at most $t_c/t_x(J_i)$ robots enter a junction J_i each cycle period:

$$\sum_{R_{ji} \in \text{RIN}(J_i)} n^{out}_{jik} \leq \frac{t_c}{t_x(J_i)}.$$

A junction contract guarantees that the same number of robots carrying a product ρ_k enter and leave a junction J_i each cycle period:

$$\forall\ \rho_k \in \boldsymbol{\rho},\ \sum_{R_{ji} \in \text{RIN}(J_i)} n^{out}_{jik} = \sum_{R_{ij} \in \text{ROUT}(J_i)} n^{in}_{jik}.$$

Road Contract. Road R_{ij}'s contract assumes that at most $\lfloor \text{CAPACITY}(R_{ij})/2 \rfloor$ robots enter the road each cycle period:

$$\sum_{\rho_k \in \boldsymbol{\rho}} n^{in}_{ijk} \leq \left\lfloor \frac{\text{CAPACITY}(R_{ij})}{2} \right\rfloor.$$

Road R_{ij}'s contract has the following guarantees. Let n_c be the maximum number of cycle periods in time T:

$$n_c := \frac{T}{t_c}.$$

Let s_{ijk} be the units of product ρ_k *sourced* from road R_{ij}'s shelves each cycle period. Let road R_{ij} contain Λ_{ijk} units of product ρ_k at time $t = 0$. No more than Λ_{ijk}/n_c units of product ρ_k can be sourced from road R_{ij} each cycle period:

$$s_{ijk} \leq \frac{\Lambda_{ijk}}{n_c}.$$

A product can only be picked up by an unburdened robot. The total number of products sourced from road R_{ij} each cycle period must be less than the number of unburdened robots entering road R_{ij} each cycle period:

$$\sum_{\rho_k \in \boldsymbol{\rho}} s_{ijk} \leq n^{in}_{ij0}.$$

No more than n^{out}_{ijk} units of product ρ_k can be *deposited* at road R_{ij}'s stations each cycle period. If d_{ijk} is the units of product ρ_k deposited at road R_{ij} each cycle period, then:

$$d_{ijk} \leq n^{out}_{ijk}.$$

A robot cannot appear or disappear. Thus, in a cycle period, the number of robots leaving road R_{ij} carrying product ρ_k is equal to:

1) the number of robots entering R_{ij} carrying product ρ_k
2) *plus* the number of robots picking up product ρ_k in R_{ij}
3) *minus* the number of robots depositing product ρ_k in R_{ij}

$$\forall \rho_k \in \boldsymbol{\rho},\ n^{out}_{ijk} = n^{in}_{ijk} + s_{ijk} - d_{ijk}.$$

An analogous equation can be written which relates the number of unburdened robots entering and leaving road R_{ij}:

$$n^{out}_{ij0} = n^{in}_{ij0} - \sum_{\rho_k \in \boldsymbol{\rho}} s_{ijk} + \sum_{\rho_k \in \boldsymbol{\rho}} d_{ijk}.$$

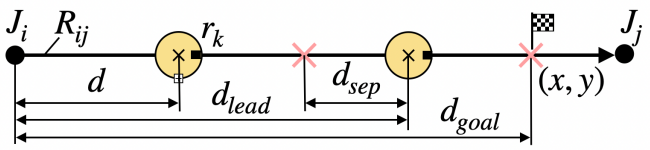

Fig. 6. The variables used by the motion planning FSM.

Workload Contract. A workload contract $C_{\boldsymbol{w}}$ has no assumptions. It guarantees that the units of product ρ_k deposited at a station each cycle period is greater than w_k/n_c. Recall that w_k is the demand for product ρ_k and n_c is the number of cycle periods executable in time T:

$$\forall\ \rho_k \in \boldsymbol{\rho},\ \sum_{R_{ij} \in \boldsymbol{R}} d_{ijk} \geq \frac{w_k}{n_c}.$$

Synthesis Implementation. The above contracts are used to generate a formula in propositional logic augmented with arithmetic constraints over the non-negative integers, which is solved using a satisfiability modulo theory (SMT) solver to produce a traffic plan for a given WSP instance.

Traffic Cycle Set Synthesis. Let a *road path* $p := \langle R_{ij}, R_{jk}, \ldots \rangle$ be a sequence of roads such that the outlet junction of each road is the inlet junction of its successor. By construction, a traffic plan $(\boldsymbol{n^{in}}, \boldsymbol{n^{out}})$ has properties:

Property 6.1: There is a set of road paths P_k for each product $\rho_k \in \boldsymbol{\rho}$ such that n^{in}_{ijk} and n^{out}_{ijk} paths in P_k enter and leave the road R_{ij}.

Property 6.2: There is a set of road paths P_0 such that n^{in}_{ij0} and n^{out}_{ij0} paths in P_0 enter and leave the road R_{ij}.

These properties imply that there is a bijection $\tilde{B} : P_0 \to \bigcup_{\rho_k \in \boldsymbol{\rho}} P_k$ for any traffic plan such that if road path $p \in P_0$ is mapped to road path $p' \in \bigcup_{\rho_k \in \boldsymbol{\rho}} P_k$, the first road in path p is the same as the last road in path p' and *vice versa*. Each cycle in a traffic cycle set Σ is synthesized from a traffic plan by concatenating each pair of paths $p \mapsto p' \in \tilde{B}$:

$$\Sigma := \{pp' : p \mapsto p' \in \tilde{B}\}.$$

VII. Motion Plan Synthesis

Motion planning computes an action trajectory for each robot which causes the robot to circle its traffic cycle at the rate of one road per cycle period, picking up products at its source road and depositing them at its destination road. An action trajectory is computed by using a finite state machine (FSM) to generate *instructions*. An instruction is a function which takes a robot's state and a small number of additional parameters and returns a short *instruction action trajectory*. A robot executes its instructions' action trajectories in sequence. Thus, a robot's action trajectory is formed by concatenating its instruction action trajectories.

A. Instructions

An instruction can either be a *primitive instruction* or a *derived instruction*. A primitive instruction has a self-contained definition. A derived instruction computes a set of parameters and then passes these parameters to a primitive instruction. An instruction assumes that a robot r_k is initially stationary, that is, has $\dot{q}_k = \mathbf{0}$, and guarantees that a robot will be stationary after it is executed. As a result, instructions can be combined in any order.

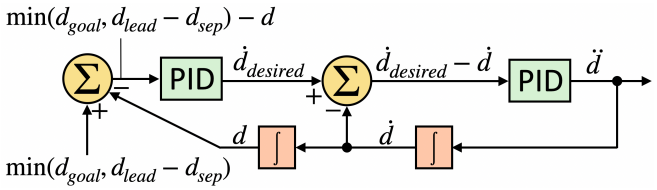

Fig. 7. A motion controller implementing the MV instruction.

Primitive Instructions. There are three primitive instructions. The instruction:

1) WAIT(t) instructs a robot to wait for t seconds.
2) ORIENT(R_{ij}) instructs a robot to rotate on the spot until it is oriented in the same direction as road R_{ij}.
3) MV$(R_{ij}, (x, y))$ instructs a robot to move along road R_{ij} to the point (x, y) without colliding with the robot in front. It assumes that the point (x, y) is on road R_{ij} and is closer to road R_{ij}'s outlet junction than the robot.

There are many ways to implement the instruction MV$(R_{ij}, (x, y))$. One possible implementation is as follows. Let the distance from road R_{ij}'s inlet junction J_i to:

1) robot r_k be d.
2) robot r_k's goal position (x, y) be d_{goal}.
3) the position of the robot on road R_{ij} in front of robot r_k be d_{lead}. If no such robot exists, $d_{lead} = \infty$.

Let d_{sep} be the distance that robot r_k should maintain between itself and the robot in front for safety. Fig. 6 illustrates these variables. The instruction MV$(R_{ij}, (x, y))$ can be implemented using the motion controller shown in Fig. 7. The left PID block finds the speed $\dot{d}_{desired}$ that robot r_k should move along road R_{ij} at to reduce the error signal

$$\min(d_{goal}, d_{lead} - d_{sep}) - d$$

to 0 as quickly as possible. If the robot ahead of robot r_k is distance d_{sep} or more beyond the point (x, y), the PID controller sets d to d_{goal}, moving robot r_k to the point (x, y). Otherwise, the PID controller sets d to $d_{lead} - d_{sep}$, holding robot r_k distance d_{sep} behind the robot in front. The right PID block finds an acceleration $\ddot{d}$, which minimizes the difference between robot r_k's desired and actual speed.

Derived Instructions. The following instructions are derived from the primitive instructions. The instruction:

1) MVTOSHELF(R_{ij}, ρ_l) moves a robot on the road R_{ij} to the nearest point where road R_{ij} intersects the access area of a shelf containing product ρ_l.
2) MVTOSTATION(R_{ij}) moves a robot on road R_{ij} to the nearest point where road R_{ij} intersects a station.
3) WAITFORLOCK(J_i) makes a robot wait until it has acquired the lock J_i. Recall that each junction is associated with a lock, and that a robot may not enter the junction's overlap area until it acquires this lock.

B. Motion Planning FSM

The sequence of instructions that a robot r_k executes is generated by the state machine shown in Fig. 8. Each FSM state is depicted in yellow. The condition and operations associated with an FSM transition are depicted in red and blue, respectively. Default transitions, transitions taken by a robot when it does not satisfy any other transition's condition, have the condition $*$.

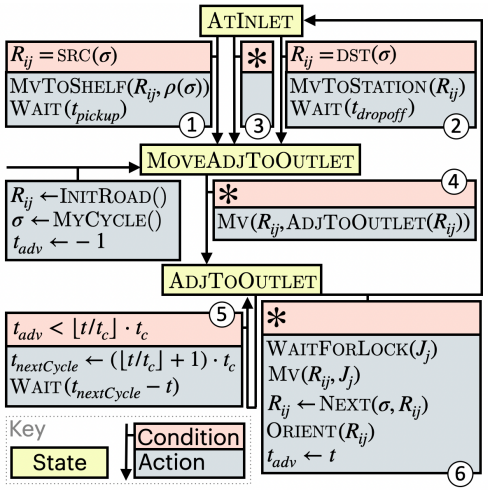

Fig. 8. The motion planning FSM.

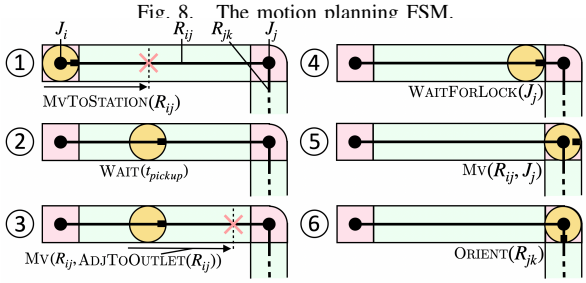

Fig. 9. The envelope of each road and overlap area of each junction in the example traffic system.

Let $\sigma := (\text{ROBOTS}(\sigma), \text{ROADS}(\sigma), \text{SRC}(\sigma), \text{DST}(\sigma), \rho(\sigma))$ be the traffic cycle that a robot r_k is part of, that is, $r_k \in \text{ROBOTS}(\sigma)$. Recall that a robot in traffic cycle σ transports product $\rho(\sigma)$ from the shelves in the source road SRC(σ) to the stations in the destination road DST(σ) via the cycle of roads ROADS(σ). Let R_{ij} be the road that robot r_k is traversing. When robot r_k is at road R_{ij}'s inlet junction J_i, robot r_k is in the state ATINLET. If road R_{ij} is:

1) robot r_k's source road, robot r_k moves to the nearest shelf containing the product $\rho(\sigma)$ and waits for its arm to pick up a unit of this product (Fig. 8 ①).
2) robot r_k's destination road, robot r_k moves to the nearest shelf containing the product $\rho(\sigma)$ and waits for its arm to put down a unit of this product (Fig. 8 ②).
3) neither, robot r_k takes no action (Fig. 8 ③).

Robot r_k then transitions to the state MVADJTOOUTLET. We say that robot r_k is adjacent to the outlet junction J_j of road R_{ij} if it is as close to junction J_j as it is possible to be without entering the junction's overlap area. A robot adjacent to the outlet junction of a road is shown in Fig. 9 ④. Let ADJTOOUTLET(R_{ij}) be the point on road R_{ij} that a robot is at when it is adjacent to road R_{ij}'s outlet junction:

$$\text{ADJTOOUTLET}(R_{ij}) := \underset{(x,y)\in(R_{ij}\setminus\text{OVERLAP}(J_j))}{\arg\min} ||(x, y) - J_j||.$$

In the state MVADJTOOUTLET, robot r_k moves adjacent to road R_{ij}'s outlet and then transitions to the state ADJTOOUTLET (Fig. 8 ④).

A robot may only advance one road in its traffic cycle each cycle period. Let t be the current time and t_{adv} be the time

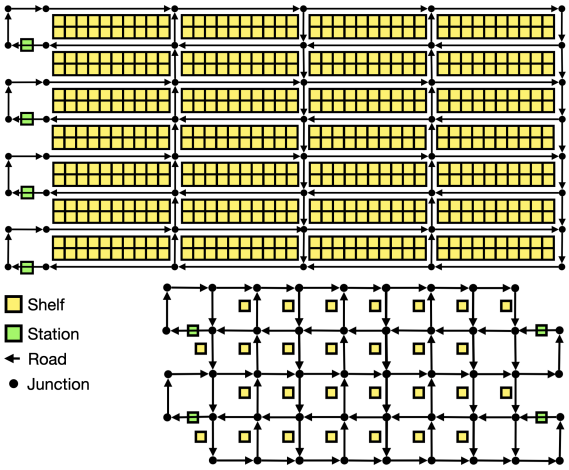

Fig. 10. (top) The automated warehouse map WAREHOUSE1. (bottom) The sorting center map SORTINGCENTER.

that robot r_k last advanced. If robot r_k has advanced after the start of the current cycle period (at time $\lfloor t/t_c \rfloor \cdot t_c$), robot r_k waits until the next cycle period starts (at time $t_{nextCycle} := (\lfloor t/t_c \rfloor + 1) \cdot t_c$) (Fig. 8 ⑤). Otherwise, robot r_k waits to acquire the lock to road R_{ij}'s outlet junction J_j. When this lock has been acquired, robot r_k moves to junction J_j, sets R_{ij} to the next road in its road cycle and orients itself in the direction of this road. Robot r_k then updates the time t_{adv} it advanced last and returns to the state ATINLET (Fig. 8 ⑥).

Example. Fig. 9 shows a robot executing the FSM on its destination road. The robot starts off at the road's inlet junction. First, the robot moves to the nearest station ① and then waits long enough for its arm to deposit the product it is carrying ②. Next, the robot moves adjacent to the road's outlet junction ③ and waits to acquire the junction's lock ④. When the robot acquires the junction's lock, it moves to the junction ⑤ and orients itself in the direction of the next road in its traffic cycle ⑥.

VIII. EVALUATION

Our evaluation compares CCMP's scalability and solution quality to the state of the art on real industrial scenarios.

Implementation. CCMP is implemented as an automatic toolchain. Junction contracts, road contracts, and the workload contract are compiled and composed in Python 3.11, and a traffic plan satisfying these contracts is synthesized using Gurobi [19]. This traffic plan is converted into a traffic cycle set and the traffic cycle set into a motion plan using modules written in Python 3.11.

Scenarios. We evaluate CCMP on two real industrial scenarios: a Kiva (now Amazon Robotics) automated warehouse [20] and a package sorting center [21].

Automated Warehouse. Evaluations are conducted on two warehouse maps [20]: WAREHOUSE1, a map with 280 shelves and 4 stations, and WAREHOUSE2, a map with 240 shelves and 10 stations. The map WAREHOUSE1 is depicted in Fig. 10 (top).

Sorting Center. Evaluations are also conducted on a sorting center scenario. A sorting center sorts packages by destination. A sorting center contains chutes and bins of packages. Each chute leads to a shipping container bound for a unique destination. A robot sorts a package by ferrying it from a bin to the chute associated with its destination. Bins are typically assumed to contain an unlimited number of packages. The goal is to fill the shipping containers before they are scheduled to leave the warehouse.

We model this problem as a WSP as follows. Let each bin be modeled as a station. Let the ith chute be modeled as a shelf containing unlimited units of product ρ_i. Let n_i be the number of packages that must be brought to the ith chute. A WSP instance is generated where the demand for each product $\rho_i \in \boldsymbol{\rho}$ is n_i. Solving this WSP instance produces a motion plan which brings n_i units of product ρ_i from the ith chute to the bins of products. Swapping the locations where robots pick up and drop off products generates the desired solution. We evaluate CCMP on the map SORTINGCENTER, depicted in Fig. 10 (bottom). It has 28 chutes and 4 bins.

Robot Model. A robot is modeled as a tricycle robot whose wheels have a maximum speed of 1 m/s and a maximum acceleration of 1 m/s^2.

Benchmarks. We evaluate CCMP against three benchmark planners taken from the literature: Contract-based Cyclic Path Planning (CCPP) [4], a contract-based planner, iterated Explicit Estimation Conflict Based Search (iterated EECBS) [14], a bounded-suboptimal search based planner and Rolling Horizon Collision Resolution (RHCR) [15], another bounded-suboptimal search based planner. Since neither iterated EECBS nor RHCR can perform task formulation, these planners were asked to find a motion plan where each robot visits the same sequence of shelves and stations as it did in CCMP's plan. CCPP, iterated EECBS and RHCR model a warehouse's floorplan as a grid graph and robots as idealized agents which move between vertices using move instructions. The sequences of instructions that these planners generate are converted into a motion plan using the MV(), ORIENT() and WAIT() instructions described in Section VII-A. An attempt to benchmark CCMP against ORCA [9], a collision-avoidance-based motion planner, failed because ORCA cannot handle the one-robot-wide corridors found in the automated warehouse maps.

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

Experiments. CCMP and the benchmark planner were run on WSP instances whose workload contained 10^2, 10^3, 10^4, 10^5 and 10^6 products on each of the three benchmark maps. Each planner was asked to find a motion plan which took less than $3.6\times$ the number of products in the workload minutes, that is, $3.6 \cdot \sum_{\rho_k \in \boldsymbol{\rho}} w_k$ minutes to execute. Each planner was given 2 minutes to begin to move the robots. The planner was then expected to move the robots in real time until execution finished.

Results. Table I lists the WSP instances that each planner was and was not able to solve. Table II lists the length of the motion plan generated by each benchmark planner on selected WSP instances as a multiple of the length of CCMP's motion plan. Iterated EECBS and RHCR, the bounded-suboptimal search-based planners, could slightly outperform CCMP because a traffic system did not constrain their motion plans. The space of solutions to the WSP

TABLE I

THE WSP INSTANCES THAT CCMP AND THE BENCHMARK PLANNERS WERE AND WERE NOT ABLE TO SOLVE.

	Planner	Products in Workload				
		10^2	10^3	10^4	10^5	10^6
AW1	CCMP	✓	✓	✓	✓	✓
	CCPP	✓	✓	✓	✓	✓
	EECBS	✓	✗	✗	✗	✗
	RHCR	✓	✗	✗	✗	✗
AW2	CCMP	✓	✓	✓	✓	✓
	CCPP	✓	✓	✓	✓	✓
	EECBS	✓	✗	✗	✗	✗
	RHCR	✓	✗	✗	✗	✗
SORTC	CCMP	✓	✓	✓	✓	✓
	CCPP	✓	✓	✓	✓	✓
	EECBS	✓	✓	✗	✗	✗
	RHCR	✓	✓	✗	✗	✗

TABLE II

THE LENGTH OF THE PLAN GENERATED BY EACH OF THE BENCHMARK PLANNERS AS A MULTIPLE OF THAT OF CCMP.

Scenario	Performance		
	CCPP	EECBS	RHCR
AW1 (10^2 products)	2.8×	0.85×	0.87×
AW1 (10^6 products)	2.5×	N/A	N/A
AW1 (10^2 products)	1.7×	0.91×	0.92×
AW2 (10^6 products)	2.4×	N/A	N/A
SORTC (10^2 products)	2.5×	0.95×	0.83×
SORTC (10^6 products)	2.9×	N/A	N/A

that these planners have to search grows exponentially with workload size, however, preventing these solvers from scaling beyond 10^2 products on the automated warehouse maps. CCPP, conversely, scales as well as CCMP because it also uses contract-based planning. CCPP, however, plans for idealized agents moving on a grid. Converting CCPP's plan for idealized robots into a plan for realistic robots substantially degrades CCPP's solution quality. Converting CCPP's plan for idealized robots to a plan for realistic robots produces a relatively low-quality solution. CCPP's plan consists of a sequence of discrete movements between neighboring grid cells. Converting this discrete space plan to continuous space causes robots to start and stop unnecessarily. Additionally, since all robots have to finish movement associated with one time step before movement associated with the next time step can begin, robots may have to wait for other robots to finish turning before starting to move again. These factors lead CCMP to outperform CCPP by 2.9x.

IX. CONCLUSIONS

We introduced CCMP, a methodology to solve the WSP at scale. CCMP models the warehouse as a traffic system made of multiple components, each specified by a contract. It then exploits an ILP solver to generate traffic flows that satisfy the contracts and a novel motion planner to convert them into plans for a team of robots. CCMP was implemented a Python toolchain which leveraged the Gurobi logical solver. Evaluated on real industrial scenarios with up to 1 million products, CCMP outperformed comparable methodologies by up to 2.9×.

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] S. M. LaValle, *Planning Algorithms*. Cambridge U. Press, 2006.

[3] G. Neville, A. Messing, H. C. Ravichandar, S. A. Hutchinson, and S. Chernova, "An interleaved approach to trait-based task allocation and scheduling," in *The IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE Press, 2021, pp. 1507–1514.

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

[5] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, "Contracts for system design," *Foundations and Trends® in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.

[6] 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.

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

[8] J. P. van den Berg and M. H. Overmars, "Prioritized Motion Planning for Multiple Robots," *The International Conference on Intelligence Robots and Systems*, pp. 430–435, 2005.

[9] J. van den Berg, S. Guy, M. Lin, and D. Manocha, "Reciprocal n-Body Collision Avoidance," *Robotics Research. Springer Tracts in Advanced Robotics*, vol. 70, pp. 3–19, 2011.

[10] L. E. Kavaraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[11] S. Giordani, M. Lujak, and F. Martinelli, "A Distributed Multi-Agent Production Planning and Scheduling Framework for Mobile Robots," *Computers and Industrial Engineering*, vol. 64, no. 1, p. 19–30, 2013.

[12] G. A. Korsah, B. Kannan, B. Browning, A. Stentz, and M. B. Dias, "xBots: an Approach to Generating and Executing Optimal Multi-robot Plans with Cross-Schedule Dependencies," *The International Conference on Robotics and Automation*, p. 115–122, 2012.

[13] A. Javier, J. A. DeCastro, R. Vasumathi, D. Rus, and H. Kress-Gazit, "Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles," *Auton. Robots*, vol. 42, no. 4, p. 801–824, 2018.

[14] J. Li, W. Ruml, and S. Koenig, "EECBS: A bounded-suboptimal search for multi-agent path finding," *The AAAI Conference on Artificial Intelligence*, vol. 35, pp. 12 353–12 362, 2021.

[15] J. Li, A. Tinka, S. Kiesel, J. W. Durham, S. T. K. Kumar, and S. Koenig, "Lifelong Multi-Agent Path Finding in Large-Scale Warehouses," in *The AAAI Conference on Artificial Intelligence*, 2021, pp. 11 272–11 281.

[16] M. Liu, H. Ma, J. Li, and S. Koenig, "Task and Path Planning for Multi-Agent Pickup and Delivery." *The Joint Conference on Autonomous Agents and Multi-Agent Systems*, pp. 11 560–11 565, 2019.

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

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

[19] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

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

[21] 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.