Distributed Multi-robot Equitable Partitioning Algorithm for Allocation in Warehouse Picking Scenarios

Giovanni D'urso[1], Armin Sadeghi[2], Chanyeol Yoo[1], Stephen L. Smith[2], Robert Fitch[1]

***Abstract*— Growth in e-commerce means that warehouses need to fulfil more orders in less time. Order fulfilment dominates operational cost (55% to 70%), hence improvements can have substantial economic impacts. Warehouses can increase order picking throughput by using methods that account for the stochastic nature of real-time online order arrival. This paper introduces an improvement over traditional zone picking strategies by *partitioning* the warehouse into *zones of equal work* that account for spatio-temporal demand arrival. We then prescribe a service policy for the team of robots or human workers with fixed item-storage capacity to service the demands of a given zone. The policy and partitioning are designed to optimize steady state performance. Our method is not specific to a particular warehouse configuration and scales to large warehouses with many robots. We validate our algorithms' performance on simulated warehouse environments and show favourable performance compared to existing equitable partitioning methods and naive order to picker allocation. We show through simulation that a team of 5 robots with a 5-item capacity collects 10-30% more items per day than the compared methods.**

I. Introduction

Globally integrated logistics are fundamental to a modern society. Recent world events have had negative impacts on the global supply chain, demonstrating the importance of automating logistics and its essential role in our daily lives. Maintaining these systems is a key component in building a more resilient future as our society grows.

A central operation in warehouse logistics is *order picking*, where items are collected from their warehouse storage locations and then packaged to fulfil customer orders. Order picking is an on-demand process that can benefit greatly from optimisation.

Improving order picking efficiency is also important in terms of cost savings. Because manual order picking accounts for 55% to 70% of warehouse operation costs [1], a modest improvement in efficiency can yield considerable economic benefits. Manual *picker-to-part* order picking systems are the most common in practice (up to 80% adoption) [2], but automation technologies, including multi-robot systems, are also available and used.

In this paper, we consider the problem of designing efficient planning algorithms to optimise order picking in warehouses. Our intention is to design algorithms that are applicable to both robotic and manual operations. The rate at which orders arrive is not necessarily regular, so planning algorithms must account for this *dynamic* aspect. Warehouse operations often require a *team* of people or robots (agents), so planning algorithms must also be capable of utilising the cooperation of multiple robots to achieve high throughput. Finding optimal algorithmic solutions, unfortunately, is infeasible for all but the smallest of scenarios because component subproblems such as routing, batching, and allocation are NP-hard [3].

We propose a method to address dynamic warehouse picking that works by partitioning the environment into areas of *equal work*. According to an estimation of expected work, these partitions are equitable, which accounts for an irregular rate of order arrival. These partitions facilitate the allocation of work to each robot/agent: When an order of several items arrives, items are considered individually, and work is allocated to the partition that contains the corresponding item. These item demands are then serviced by robots/agents which follow a given policy. This approach of picking orders at the level of constituent items requires a collating process at the depot where items are dropped off, but this is a worthwhile tradeoff since picking is the dominant process in terms of time, complexity, and cost. Our formulation scales well with the number of robots/agents because it avoids the joint optimisation problems involved in batching and allocation, and due to efficiencies gained through the distributed design of the partitioning algorithm.

More formally, we create an initial set of Voronoi partitions [4] using *kmeans++* [5] seeding and then use *Lloyds algorithm* [6] to compute an initial spatially balanced/equally weighted power diagram [7]. Subsequently, we iteratively update the weights in the power diagram using a local optimisation algorithm that balances regions according to a heuristic measure of *work* required to service a partition. The measure of work is the expected time it will take to visit a set of items to pick up and then return to the depot to drop off. This metric is computed by sampling orders from a spatio-temporal model of order arrival. Once the partitions have been created, a robot/agent is allocated to each and it follows a policy to service a queue of demands that are allocated to its partition. A benefit of the partitioning method is that our algorithm has the *anytime* property; each iteration produces a feasible candidate partitioning. Additionally, its fast computation speed means that it can be used as a design tool to evaluate the performance of various system characterisations.

This research is supported by an Australian Government Research Training Program (RTP) Scholarship and a Jumbunna Postgraduate Research Scholarship.

[1]Authors are with the University of Technology Sydney, Sydney, Australia. giovanni.durso@student.uts.edu.au, {Chanyeol.Yoo,Robert.Fitch}@uts.edu.au

[2]Authors are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. {a6sadegh,stephen.smith}@uwaterloo.ca

Typically, warehouse operations seek to maximise the utilisation of robots/agents. Therefore, we evaluate our method operating in a near heavy-load regime (i.e., the order arrival rate is approximately equal to the service rate). The policy and design of our algorithm behave accordingly and exhibit the best performance under these conditions.

This paper offers several contributions. First, we propose an equitable partitioning and batch servicing policy-based approach to solve the dynamic warehouse order-picking problem. Second, we define a cost metric that accounts for the obstacles and depot returns in warehouse order picking, along with approximations to compute it in polynomial time for an arbitrary distribution that describes orders. Finally, we provide extensive simulated results showing improved performance compared to typical order picking and a more generic equitable partitioning method.

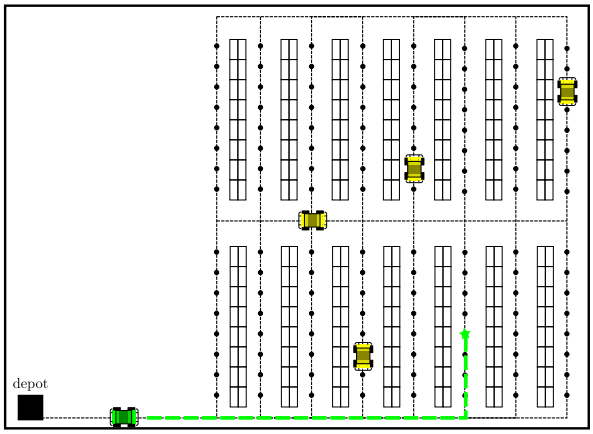


Fig. 1. Example illustration of a warehouse with robots performing order picking. The black circles are the item locations that robots must visit to pick up an item. The depot shown as a black square is where robots must return to drop off their collected items.

II. Related work

This section outlines the available relevant literature on the multi-agent dynamic warehouse picking problem. It has been shown that order-picking performance can be improved by batching orders into tours. Batches for fully known static problems can be found with *Ant colony optimisation* (ACO) [8], linear programming with column generation [9] or learning-based methods [10]. Similarly, dynamic order batching can be handled by and updating existing plans with new orders [11]. However, this does not scale well because of its computational complexity.

The warehouse-order-picking problem is similar to the vehicle routing problem (VRP) [12]. Traditionally, VRP problems consist of calculating an optimal set of routes for a team of vehicles that service all demands. Unfortunately, finding an optimal VRP solution to a problem is NP-Hard [13]. However, many heuristics are available, and it is still field applicable. Order-picking has been modelled as a VRP with time windows by [14], and [15] while also considering delivery vehicle costs. However, their solutions are computationally complex and do not handle dynamic demands.

Static VRP solutions assume that all information is available and does not change. Many problems have jobs that arrive throughout the day; costs can be stochastic. These problems are known as *dynamic* VRPs [16]. Usually, demands are distributed by a spatio-temporal process, and the solution is to find a policy that minimises the average time a demand waits to be serviced or maximising the throughput of the system. Research into this class of problems includes spatially-distributed surveillance policies for UAVs [17], and information-gathering problems [18]. DVRPs have been applied in warehouse operational research by splicing orders into existing allocations [19] and handling the dynamic aspects by introducing dummy orders based on the forecasted dynamic demand [20]. However, these methods' performance degrades in highly dynamic scenarios.

Solving DVRPs and formulating a high-quality policy becomes more difficult as the size of the team grows. However, some methods partition the workspace space into less complicated standalone sub-problems. An example of this technique is finding approximately equitable partitions for convex regions using a distributed partitioning policy [21]. Subsequently, they solve the dynamic vehicle routing problem by applying their partitioning policy and adaptive queueing [22]. Similar work on performing decentralised Voronoi coverage in non-convex environments is also shown [23]. Finally, the multi-agent spatial load balancing problem in non-convex environments is introduced in [23, 24]

However, the non-warehouse-focused research does not consider the effects of routing around shelves, returning to the depot to drop off items and warehouse environment spatial distributions of items in warehouses. Routing around shelves can break obstacle-free space assumptions and require non-euclidean distance metrics. In addition, depot returns introduce a variable cost based on the partition location relative to the depot and the agent's position. Finally, some methods rely on statistical properties about the spatial distributions of demands that may not be valid in a warehouse environment, which can have very skewed order composition distributions. Therefore, there is a lack of practical and scalable methods to solve the warehouse-order-picking problem with highly dynamic orders and high agent utilisation.

There are solutions to DVRP problem variants, such as DVRP with capacity, multi-trip DVRP, DVRP in non-convex environments, and DVRP with empirically derived distributions. However, no solution covers all these variations jointly, which is essential to practically solving the warehouse picking problem. This work addresses the specifics of the multi-agent dynamic warehouse order picking problem in a practical scalable way.

III. Problem Definition

Suppose that we have a two-dimensional warehouse $\mathcal{W} \in \mathbb{R}^2$ with a known layout of the shelves and obstacles $\mathcal{O} \subset \mathcal{W}$ as depicted in Fig. 1. The warehouse stores a known set of items $\mathcal{I} = \{I_0, I_1, \ldots\}$ and their pickup locations $\mathcal{X}_I =$

$\{\mathbf{x}_0, \mathbf{x}_1, \cdots\}$ are known in advance. We assume that the stored quantity for each item is practically infinite, and there exists only one location to pick up an item $I \in \mathcal{I}$ (i.e., item I_k can only be picked up at $\mathbf{x}_k$). All items in $\mathcal{I}$ have the same weight and size. The items are then dropped off at *depot* $D \in \mathcal{W} \backslash \mathcal{O}$ to be serviced.

We define an *order* as a set of item-quantity pairs to be serviced. The order received at time t is denoted as $\mathbf{o}_t = (I_k^t, N_k^t) \in \mathbb{O}$, where $I_k \in \mathcal{I}$ represents the item and the corresponding location, $N_k \in \mathbb{Z}^+$ is the quantity of the item in the order and $\mathbb{O}$ denotes a set of all possible orders (i.e., item-quantity pairs). The timed discrete sequence of orders received is denoted as $\mathbf{O} = \mathbf{o}_{t_0}\mathbf{o}_{t_1}\cdots$. We assume that the interarrival time of orders and the item-quantity pair within orders are both independent and identically distributed. As commonly practiced [25, 26], arrival times of orders are modelled as a continuous-time Poisson process $\{\mathcal{P}(t)\}_{t\in\mathbb{R}^+}$. The contents of those orders (i.e., item-quantity pairs) are drawn according to item selection distribution ϕ and quantity distribution ψ. Note that the item and quantity distributions can be arbitrary and are often derived based on a historic dataset of orders. An order is considered serviced when all requested items are delivered to the depot D.

Items in orders are added to a first-in-first-out (FIFO) queue $\mathbf{Q}$ defined as a discrete sequence of items, denoted as $\mathbf{Q}(t) = (I^k \in \mathcal{I})_{k\in\mathbb{Z}_{\geq 0}}$, where k indicates the indexes of items (i.e., the very first item added to the queue is denoted as I^0). The queue describes the set of items yet serviced at time t. Therefore, the number of items serviced, denoted as $N(t)$, is $k-1$ where k is the superscript of the next item to be serviced in the queue.

To service the orders received, we deploy m homogeneous robots/agents $\mathbf{R} = \{\mathbf{r}_0, \cdots, \mathbf{r}_m\}$ (henceforth called robots for clarity) where the number does not change. Robot $\mathbf{r}_i \in \mathbf{R}$ is capable of carrying R_c items at a time and moves at a constant speed R_t. The time it takes to pick up and drop off items are R_p and R_d, respectively. A robot $\mathbf{r}_i \in \mathbf{R}$ is assigned a multiset of items from the queue Q called a *batch* B_i where the number is less than or equal to R_c. Note that a batch may contain duplicate items. Once robot $\mathbf{r}_i$ delivers all items in the batch to the depot D, it receives a new batch. Given a batch B_i for robot $\mathbf{r}_i \in \mathbf{R}$, a policy $\sigma_i : B_i \rightarrow (\mathcal{W}\backslash\mathcal{O})$ is used to complete the batch, where σ_i dictates a sequence of positions in environment $\mathcal{W}$; the set of policies for all robots $\mathbf{R}$ is denoted as $\Sigma = (\sigma_1, \ldots, \sigma_m)$. The sub-route between two adjacent positions is assumed to be the collision-free shortest path, which can be easily found using path planning algorithms such as *probabilistic roadmap* (PRM) [27].

We define the warehouse picking problem as the question of how to allocate arriving orders $\mathcal{O}$ and the corresponding items to robots in order to maximise the number of items serviced.

We formally define the warehouse picking problem as follows:

Problem 1 (Dynamic warehouse order picking problem)**.** *Given a set of items $\mathcal{I}$ and their locations $\mathcal{X}$ in a warehouse $\mathcal{W}$ with obstacles and shelves $\mathcal{O}$, and the spatio-temporal process $\mathcal{P}$ that describes the order and the time of arrival, find a set of optimal policies $\Sigma^* = \{\sigma_0^*, \cdots, \sigma_m^*\}$ for service robots $\mathbf{R}$ with capacity R_c that maximises the expected throughput of servicing items over a long time horizon, such that*

$$\Sigma^* = \arg\max_{\Sigma} \lim_{t\to\infty} \mathbb{E}\left[\frac{N(t)}{t}\right], \tag{1}$$

where robots $\mathbf{R}$ receives new batches of items from the queue Q.

IV. Distributed Equitable Partitioning

Partitioning of the environment is a well-known approach for solving DVRP problems with infinite capacity [22], and limited capacity [16]; we extend it to work in warehouse order-picking environments. To solve the dynamic warehouse order picking problem, we consider a class of policies that partition the environment $\mathcal{W}$ into a set of m regions $P_1, \ldots, P_m \in \mathcal{M}$, where the pickup locations of items within each partition are assigned to a robot. The key idea is that the maximum expected throughput of servicing items can be achieved by minimising the expected wait time of the orders. This minimisation can be done by servicing batches of orders in each partition at around the amount of time it takes for a new batch to arrive. This arrival rate is dictated by the spatio-temporal process $\mathcal{P}$. The service rate is controlled by the partitions and the policy that robots follow. A robot $\mathbf{r}_i \in \mathbf{R}$ assigned to a partition P that is characterised by a subset of items $\mathcal{I}_i \subset \mathcal{I}$ in the warehouse and the corresponding policy $\sigma_i(B_i)$ that dictates the order of pick ups.

The distributed warehouse partitioning problem is decomposed into three sub-problems. We first formally define a service policy and associated cost metric for each robot that evaluates the expected service time for a batch. We then present a sampling-based evaluation of the cost metric given an arbitrary spatio-temporal process. We argue that our framework can work with any type of spatial distribution that describes which items are within an order. Finally, we use the sampling-based cost metric to iteratively find a set of equitable partitions that give good performance for the defined service policies for robots.

Finding the expected service time for a batch requires defining the robots' policy to do the servicing. The choice of policy can affect the partitions that need to be created.

Our first step is to compute the expected cost of servicing demands using this policy and then subsequently find the partitions that give the best performance.

A. Cost Metric for Partitions

Our servicing policy is to allocate a robot to each partition. Subsequently, each robot is responsible for servicing demands that arrive to vertices inside their partition. If there are no demands to service in partition i, then the robot i should wait at the point that minimises the travel distance to each pick location in its partition. This can be found by

computing the centroid of the set P_i. If there are demands to service in the queue of the robot, the robot waits until a batch of size R_c of demands have arrived. These batches are then added to a queue and served according to a first-in-first out (FIFO) discipline. To service a batch, the robot computes a shortest tour consisting of the demands in the batch and the depot and then follows the tour to service the demands. Using standard Kendall notation [28], this system is classified as a G/G/1 queue due to a general arrival time distribution, a general service time distribution, and each queue is serviced by a single robot.

We denote the arrival rate in the set P_i as λ_i, where $\lambda_i = \sum_{\mathbf{x}_u \in P_i} \lambda_u$. Recall that the Poisson process is defined by the Poisson distribution. The Erlang distribution is complementary to the Poisson distribution, and it counts the amount of time until the occurrence of a fixed number of events. Hence, observe that the inter-arrival times for batches of size R_c are randomly distributed according to an Erlang distribution with mean R_c/λ_i and variance R_c/λ_i^2. Let $\mathbb{E}[L_i]$ be the expected tour length to service a batch of size R_c in partition i and $R_c R_p$ be the total time in each tour for scanning and picking-up the items. Then the expected time to service a demand arriving in partition i is

$$T_i = \frac{1}{2} R_c/\lambda_i + \frac{1}{2}(\mathbb{E}[L_i] + R_c R_p) + Q_i, \tag{2}$$

where Q_i is the expected wait time in the queue of robot i. The queue of each robot is a G/G/1 queue with arrival rate λ_i/R_c. In the heavy load regime where the batches arrive frequently, the time to form a batch in each robot's queue is negligible compared to the expected time to service the demands in the queue, i.e. servicing time dominates waiting time. Now, let c_a^2 denote the variance of arrival times to a queue of a robot and c_s^2 denote the variance of service times. Then an approximation for the expected wait time of a batch in a G/G/1 queue is provided by Kingman's formula [29], i.e.,

$$Q_i \approx \frac{\lambda_i(\mathbb{E}[L_i] + R_c R_p)}{R_c - \lambda_i(\mathbb{E}[L_i] + R_c R_p)} \left(\frac{R_c/\lambda_i^2 + c_s^2}{2} \right) (\mathbb{E}[L_i] + R_c R_p). \tag{3}$$

Note that fulfilling an order may consist of picking from different sets in the partition. Therefore, for any given partition $P_1, \ldots, P_m$ of the environment, the number of items picked by robot i with the highest waiting time Q_i is a bottleneck on the total number of orders fulfilled in the warehouse. Therefore, the goal is to find a partition of the environment which minimizes the maximum expected wait time of the demands across all partitions, i.e.,

$$\min_{P_1,\ldots,P_m} \max_{i \in [m]} T_i \approx \min_{P_1,\ldots,P_m} \max_{i \in [m]} \frac{1}{2} (\mathbb{E}[L_i] + R_c R_p) + Q_i. \tag{4}$$

Observe that for a given partition of the warehouse $P_1, \ldots, P_m$, in the heavy load regime, the expected wait time for a demand is a monotonically increasing function of $\lambda_i(\mathbb{E}[L_i] + R_c R_p)$. Therefore, the problem of minimising the maximum expected wait time is equivalent to

$$\min_{P_1,\ldots,P_m} \max_{i \in [m]} \lambda_i(\mathbb{E}[L_i] + R_c R_p). \tag{5}$$

From (4) it can be seen that as the size of the set P_i increases, the time to form a batch decreases and the expected tour lengths increase. This combined with the desire to operate with as few robots as possible motivates finding partitions that work near heavy load, i.e., $0.5 \leq \lambda R_p + \frac{2\lambda \bar{r}}{m R_c} \leq 1$ where $\bar{r}$ is the expected distance of items to depot [16]. It is shown that under heavy loading regimes the unbiased Travelling Salesman Problem (TSP) policy is optimal for servicing demands [22, 30]. This motivates computing partitions that are designed to balance the length of the tour and are equitable to optimise (4).

B. Equitable Workload Partitioning

Our approach for computing regions of equal workload consists of initialising a set of partitions and then using an iterative local optimisation method to adjust these partitions until they are equitable according to our cost metric.

Recall that the warehouse stores a known set of items $\mathcal{I} = \{I_0, \cdots\}$ and their pickup locations $\mathcal{X}_I = \{\mathbf{x}_0, \cdots\}$ and we want to partition the warehouse and the corresponding pickup locations into P partitions $\mathcal{I}_i \subset \mathcal{I}$

We create partitions by partitioning the warehouse using a generalised Voronoi diagram called a *power diagram*. A power diagram consists of *power cells* that are defined by a generator location and a weight parameter. The cell contains all points that have the lowest *power distance* (weighted distance) between the point and a generator. The power cells are used to compute the partitioning by allocating each point to the power cell that has the lowest power. Subsequently, this allows the controlling partition generation based on a tunable weight parameter and can also capture the travel distance between points while considering traversing around obstacles.

More formally, let each partition contain a power cell defined as $C_i = (g_i, w_i)$ where g_i is the generator, and w_i is the weight for partition i. Let ω_i be the centroid of the partition C_i that minimises the distance to each pickup location. To allocate a pickup location $\mathbf{x}$ to the power cell C_i that describes the partition i, we use a modified version of the *power of a point equation* [7],

$$\mathrm{P}(\mathbf{x}, C_i) = p_{\mathbf{x}}^2 - w_i^2. \tag{6}$$

We define $p_{\mathbf{x}}^2$ as the collision-free shortest path cost in the warehouse $\mathcal{W}$ for traversing from the centroid ω_i of power cell C_i to pickup location $\mathbf{x}$.

Calculating the cost of time to service a batch for an arbitrary distribution of items in orders can be analytically difficult or overly specific. Therefore, we use sampling to compute an approximation of the time it takes to service a batch. For a given finite length horizon H, let S be the set of items that are contained in a set of sample orders drawn from a simulation of the spatio-temporal Poisson process $\mathcal{P}$. Using

the modified power of a point equation (6) the sampled item demands can be allocated to the partitions by the following

$$P_i^s = \{s \in S | \mathrm{P}(s, C_i) < \mathrm{P}(s, C_j), \forall C_j \in C - \{C_i\}\}. \quad (7)$$

Using these partitions and allocated samples, we can compute an approximation of the batch service time. The cost of servicing a batch consists of the expected tour length $\mathbb{E}[L_i]$ to service a batch of size R_c, the time it takes to return to the depot, and how long it takes for a batch to form.

To compute this cost, we use approximations: the time for batches to form is approximated as the number of batches formed over the finite horizon H, the depot returns are assumed to be from the centroid of the cell ω_i, and the expected tour lengths are found by computing the expected cost of TSPs over random batches selected from the set of samples allocated to a partition. Formally, the expected tour length is equal to

$$\mathbb{E}[L_i] = \mathbb{E}(\mathrm{TSP}(P_i^s, \omega_i, R_c) + V(\omega_i)). \quad (8)$$

Let $TSP(P_i^s, \omega_i, R_c)$ be an approximate polynomial time solution for the TSP that begins and ends at ω_i while making a tour over R_c samples randomly selected from P_i^s. The function V returns the round trip cost of returning to the depot from the power cell's waiting point ω_i. Then using that expected tour length and the other components, the cost to service a region over the horizon is

$$Z_i = \frac{|P_i^s|}{R_c} \left(R_c R_p + \mathbb{E}[L_i]\right). \quad (9)$$

C. Iterative Optimisation

This cost is then used in an iterative local optimisation method to adjust the weights of the partitions based on their cost relative to their neighbours to produce the set of equitable partitions with respect to (9). Our partitioning algorithm produces a fixed number of partitions, and we allocate one robot per partition, so it is initialised by creating a partition for each of the m robots. With no loss of generality, we chose to initialise the partitions as approximately spatially equal since, in practice, it works well. The initial partitions are found using the Lloyds algorithm with the kmeans++ seeding algorithm to create the initial partitions P that are approximately equal in size. A power cell is created for each partition and is initialised with equal weight. These initial weights are the starting point for optimisation.

The initial partitions are roughly equitably sized. However, they are not balanced according to the expected cost of servicing a partition (9). To balance the costs of servicing these partitions, we iteratively optimise neighbourhoods by adjusting their relative power cell weights. To perform the optimisation, let W be a tuple containing the weight of each power cell $W = (w_1, \ldots, w_m)$. Then, the cost of servicing all the regions is

$$H_v(W) = \sum_{i=1}^{m} Z_i. \quad (10)$$

The difference in workload between partitions is computed as

$$RC_i = \sum_{j \in N_i} \frac{1}{2\gamma_{ij}} \left(\frac{1}{Z_j} - \frac{1}{Z_i}\right) / H_v(W), \quad (11)$$

where $\gamma_{ij} = ||g_j - g_i||$. N_i is the set of neighbouring partitions for partition i. This local difference is used to adjust the weights according to

$$w_i' = RC_i \cdot \alpha + w_i \quad (12)$$

where α is a scalar selected to control the step size of each iteration. The new weights are used to update the weight tuple W that defines the new iteration of power cells:

$$W = (w_1', \ldots, w_m'). \quad (13)$$

The set of weights is updated simultaneously then the new partition allocations (7) are computed. This method results in changing the weights by their portion of the total cost and their cost relative to their neighbours. This results in partitions that are more expensive than their neighbours and make up more of the total cost getting a negative weight which shrinks their partition size. Similarly, lower-cost partitions get a positive weight and increase their partition size.

This process of local weight updating ((10)-(13)) continues until the allotted calculation time elapses or the system stabilises on a local minimum.

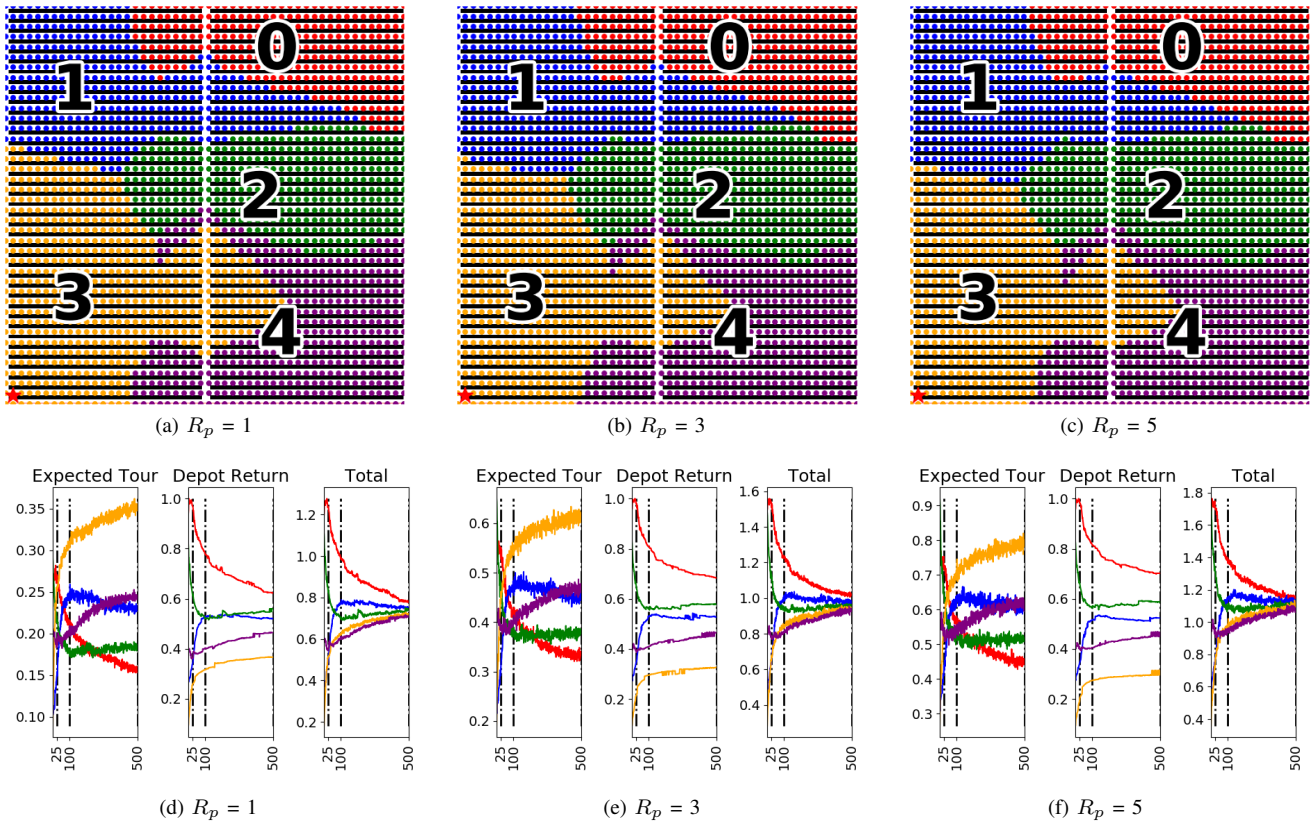
V. Discussion

The overall problem is computationally hard, mainly due to multiple agents and order uncertainty in space and time. In order to address the inherent challenge, we presented a number of heuristics and approximations. For example, we randomly sample from the process $\mathcal{P}$ rather than finding an analytical solution to simulate demand. Therefore, our sampling-based approach applies to any warehouse setting with complex and arbitrary representations of order estimates. On top of that, this approach is known to work well in practice [24].

Solving a TSP optimally is NP-Hard, so using it as a part of the partitioning cost seems detrimental. However, for conventional rectangle warehouse layouts, an optimal TSP can be computed in linear time [31]. For more generic warehouses, there still exist good quality polynomial-time approximations [32].

Since the partitioning algorithm produces valid partitions each iteration, it is *anytime* and can be stopped for the current best solution or left to run for iteratively improved solutions. This is possible because the computational complexity for computing an iteration is polynomial and can be run for a finite number of iterations, all of which give a solution.

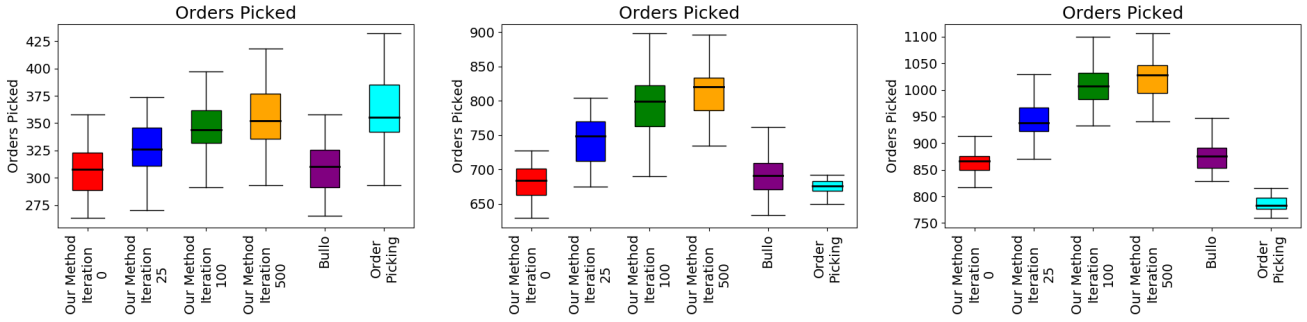
Our method scales with the problem size and number of robots due to computing the optimisation over local neighbourhoods; it only needs to consider a subset of neighbouring partitions for each partition instead of all pairwise combinations. In addition, we avoid the batching and allocation problems with our policy choice. Online each robot only needs to consider jobs inside their partition and follow the



(a) R_p = 1 (b) R_p = 3 (c) R_p = 5

(d) R_p = 1 (e) R_p = 3 (f) R_p = 5

Fig. 2. (a, b, c) Equitable partitions produced by our method for five robots with capacities $R_p = \{1, 2, 3\}$; capacity affects the partitions as capacity increases the importance of depot returns decreases. Partition 3 (in yellow) becomes smaller at higher capacity, while Partition 0 becomes larger. (d,e,f) Normalised costs of each partition over iterations. Tour costs for a single partition may increase to reduce the depot return costs for another partition, resulting in lower overall totals.



(a) $R_p = 1$ (b) $R_p = 3$ (c) $R_p = 5$

Fig. 3. Performance comparison between our method at different iterations and Bullo [22] and order picking for five robots with varying capacity $R_p = \{1, 2, 3\}$, where order picking is a baseline greedy method without partitions. The improvement as we increase the capacity R_P is shown in (a, b, c).

policy to service them. The decomposition into a series of single robot problems avoids needing to plan in the joint space.

VI. Results

This section outlines the setup for the simulation environment that we use to validate our algorithms' performance and use these empirical results to validate our algorithm.

Our simulation environment consists of 5-robots (*pickers*) operating in a hundred-meter by hundred-meter picking area within a two-block warehouse (rows are divided into two blocks) with a two-meter spacing between rows and 1200 pickup locations. The geometry of our simulated warehouse is based on a typical fulfilment warehouse owned by a logistics company that performs manual picker-to-part operations.

For our simulations, we model the item quantity ψ as a normal distribution with a mean of 5 and a variance of 2 and the item spatial order density function ϕ, which determines the likelihood for each item to be included in the order as a uniform distribution ϕ = U(0, 1). The order arrival rate is set to a heavy load regime (i.e., the order arrival rate is approximately equal to the service rate).

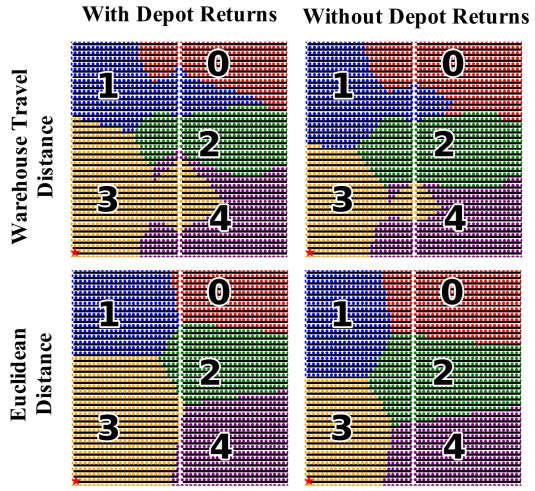


Fig. 4. Partitions created for five robots with return depot D (red asterisk) showing how our method (upper left corner) differs from existing methods. The results show that considering depot return in the cost metric in (9) induces larger partitions near the depot. Similarly, the layout of the warehouse affects the partition since it affects the practical travel distance.

The robots in our simulation have a varying amount of capacity R_p depending on the simulation. The robot travels at a constant speed of 1 meter per second. We assume that it takes a constant five seconds per item to pick up R_p and drop off R_d. For this work, we assume that all items are similarly sized with a capacity cost of 1. Finally, we assume that all robots are homogeneous.

The experimental results are averages of 60 simulation trials (approximately two months of operation) that are 10-hours long in the same environment with different sets of orders drawn from the Poisson process for each day. Each algorithm is tested for each day using the same set of demands drawn from the Poisson process to ensure that the only factor changing is the algorithm the robot team uses to service the dynamically arriving unknown demands.

We observe the relative impact of our approach by comparing it against two methods: single-order picking and a similar spatially equitable partitioning policy. Single-order picking is a policy that naively allocates whole orders to a robot in a first-come, first-serve style without considering future orders or sharing work between robots. A robot services an order by following a route from the depot to each item location and back to the depot. Once all items are picked, the robot waits for a new order at the depot. The spatially equitable partitioning policy we compare against is from prior work [22] (To avoid confusion, we refer to it as the "bullo" method), which creates partitions that are equitable with respect to the spatial density of items ϕ. Once these partitions are found, they follow the same servicing policy as our algorithm, so the main point of comparison is the different partitions.

A. Impact of Expected Tour Metric on Produced Partitions

To understand how our expected tour cost metric (8) influences the partitions created, we examine the effect of changing two of its main components. First, the expected TSP tour length in the warehouse graph to service demands and the cost to return to the depot to drop off the collected items. The chosen distance metric strongly affects the cost of the expected tours. We examine the effect of these different metrics on the cost component by replacing it with the obstacle-free Euclidean distance when computing the expected tour lengths. Similarly, we investigate the depot return effects by setting the depot return costs to 0. In Fig. 4, we show the different partitions that are generated when these two parts of the cost metric are changed. Disabling the TSP distance metric component produces differently shaped partitions. These partitions are shaped according to the Euclidean distance (ignoring obstacles) from the power generator instead of using the routes around the warehouse. Turning off the depot returns results in partitions that are sized evenly throughout the warehouse. When depot returns are used, the partitions that are closer to the depot are larger, which implies that the closer to depot partitions can cover more item locations for the same amount of work. These results help to explain the performance differences between our method and the equitable partitions produced by the "bullo" method. Their partitions are equivalent to the disabled TSP and disabled depot return case (shown bottom right); these are distinctly different to the partitions generated by and used in our method (shown top left).

B. Partitioning Results

To examine the quality of our *equal work* partitioning, we tested our algorithm in simulation, then observed the costs for each partition after each iteration. The results of 500 iterations of the algorithm for various capacities shown in Fig. 2. The first row shows that the capacity of the robots changes the partitions that the algorithm produces; as capacity increases, the importance of depot returns decreases. This effect can be seen by observing that partition labelled 3 becomes smaller at a higher capacity, and similarly, partition 0 becomes slightly larger. The second row shows that initially, the amount of work needed to service each partition is quite unbalanced, with the most expensive partition taking nearly six times the amount of work to serve as the least expensive according to our cost metric. After 500 rounds, our algorithm reduces the gap to within 5-10%; these results are consistent with most of our tested scenarios. Our algorithm produces near-balanced results that, when used in practise, are capable of high-quality improvements over the unbalanced initial partition configuration. This result demonstrates that local optimisation balances the workload between partitions and that the anytime aspect of the algorithm is useful. Additionally, it can be seen that the tour costs for a single partition can increase to reduce the depot return costs for another partition, resulting in lower overall totals. Changing the relative partition sizes balances the batch service cost and depot mixture.

C. Orders Picked Results

The collated simulation results are shown as box plots in Fig. 3. In addition, we tested our servicing policy on the

partitions generated by our local optimisation algorithm at several iterations to demonstrate that with more iterations, performance should improve if the metric being balanced is applicable. Fig. 3(a) shows the results for robots with one-capacity. In this configuration, our method performs similarly to order-picking because the constant depot returns reduce the opportunity for improvement by minimising the travel distance to subsequent items inside a batch. Therefore, the benefit of partitioning is reduced. Figure 3(b) shows the results for robots with three-capacity. In this scenario, the benefit of having robots distributed around the warehouse becomes apparent, as they can reduce wasted travel by being closer to demands when they arrive. Additionally, even less warehouse-specific "bullo" partitioning starts to outperform order-picking. Lastly, Fig. 3(c) shows that these results continue to hold as you increase capacity. Subsequently, our method can service 10-30% more orders over a simulated workday in cases with higher capacity.

VII. Conclusion

In this work, we presented an anytime framework for the dynamic warehouse order picking problem. We defined the problem by modelling the warehouse and the arrival of orders to pick as a Poisson process that generates items from an arbitrary distribution. We designed a computationally efficient partitioning-based algorithm that creates regions of *equal work* suitable for regimes of near heavy-load. A robot is then allocated to each partition to service dynamically arriving jobs using a policy. The important contribution is that we address important aspects such as travel time to the depot and non-Euclidean space, which is often neglected. We validated the approach in simulations over a two-month operation under varying capacity constraints; the proposed framework has been proven to outperform typical single-order picking and a comparable equitable partitioning-based method under heavy loading conditions.

Acknowledgements

The authors would like to thank *Premonition.io* (https://www.premonition.io/) for their contributions to the motivation of this work, support during its development and logistics expertise.

References

[1] V. Khanzode and B. Shah, "A comprehensive review of warehouse operational issues," *Int. J. Logist. Syst. Manag*, vol. 26, p. 346, 01 2017.

[2] R. M. B. M. de Koster, T. Le-Duc, and K. J. Roodbergen, "Design and control of warehouse order picking: A literature review," *Eur. J. Oper. Res.*, vol. 182, pp. 481–501, 2007.

[3] A. Gademann, J. P. Van Den Berg, and H. H. Van Der Hoff, "An order batching algorithm for wave picking in a parallel-aisle warehouse," *IIE Trans.*, vol. 33, no. 5, pp. 385–398, 2001.

[4] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[5] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," Stanford, Tech. Rep., 2006.

[6] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[7] F. Aurenhammer, "Power diagrams: properties, algorithms and applications," *SIAM J. Comput.*, vol. 16, no. 1, pp. 78–96, 1987.

[8] E. Ardjmand, H. Shakeri, M. Singh, and O. S. Bajgiran, "Minimizing order picking makespan with multiple pickers in a wave picking warehouse," *Int. J. Prod. Econ.*, vol. 206, pp. 169–183, 2018.

[9] İ. Muter and T. Öncan, "Order batching and picker scheduling in warehouse order picking," *IISE Transactions*, vol. 54, no. 5, pp. 435–447, 2022.

[10] L. Zhou, C. Lin, Q. Ma, and Z. Cao, "A learning-based iterated local search algorithm for order batching and sequencing problems," in *Proc. of IEEE CASE*. IEEE, 2022, pp. 1741–1746.

[11] J. P. van der Gaast, B. Jargalsaikhan, and K. J. Roodbergen, "Dynamic batching for order picking in warehouses," in *15th IMHRC Proceedings*. Digital Commons Georgia Southern, 2018.

[12] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Manage. Sci.*, vol. 6, no. 1, pp. 80–91, 1959.

[13] P. Toth and D. Vigo, *The vehicle routing problem*. SIAM, 2002.

[14] D. Schubert, A. Scholz, and G. Wäscher, "Integrated order picking and vehicle routing with due dates," *OR Spectr.*, vol. 40, no. 4, pp. 1109–1139, 2018.

[15] A. Scholz, D. Schubert, and G. Wäscher, "Order picking with multiple pickers and due dates–simultaneous solution of order batching, batch assignment and sequencing, and picker routing problems," *Eur. J. Oper. Res.*, vol. 263, no. 2, pp. 461–478, 2017.

[16] D. J. Bertsimas and G. Van Ryzin, "Stochastic and dynamic vehicle routing in the euclidean plane with multiple capacitated vehicles," *Oper. Res.*, vol. 41, no. 1, pp. 60–76, 1993.

[17] P. Grippa, D. A. Behrens, F. Wall, and C. Bettstetter, "Drone delivery systems: job assignment and dimensioning," *Auton. Robot*, vol. 43, no. 2, pp. 261–274, 2019.

[18] T. Patten, R. Fitch, and S. Sukkarieh, "Large-scale near-optimal decentralised information gathering with multiple mobile robots," in *Proc. of ARAA ACRA*, no. 0.1, 2013, pp. 0–15.

[19] V. Giannikas, W. Lu, B. Robertson, and D. McFarlane, "An interventionist strategy for warehouse order picking: Evidence from two case studies," *Int. J. Prod. Econ.*, vol. 189, pp. 63–76, 2017.

[20] R. D'Haen, K. Braekers, and K. Ramaekers, "Integrated scheduling of order picking operations under dynamic order arrivals," *Int. J. Prod. Res.*, pp. 1–22, 2022.

[21] M. Pavone, A. Arsie, E. Frazzoli, and F. Bullo, "Equitable partitioning policies for robotic networks," in *Proc. of IEEE ICRA*. IEEE, 2009, pp. 2356–2361.

[22] F. Bullo, E. Frazzoli, M. Pavone, K. Savla, and S. L. Smith, "Dynamic vehicle routing for robotic systems," *Proc. IEEE*, vol. 99, no. 9, pp. 1482–1504, 2011.

[23] A. Breitenmoser, M. Schwager, J.-C. Metzger, R. Siegwart, and D. Rus, "Voronoi coverage of non-convex environments with a group of networked robots," in *Proc. of IEEE ICRA*. IEEE, 2010, pp. 4982–4989.

[24] B. Boardman, T. Harden, and S. Martínez, "Limited range spatial load balancing in non-convex environments using sampling-based motion planners," *Auton. Robots*, pp. 1–18, 2018.

[25] R. de Koster, "Performance approximation of pick-to-belt orderpicking systems," *Eur. J. Oper. Res.*, vol. 72, no. 3, pp. 558–573, 1994.

[26] F. Chen, Y. Wei, and H. Wang, "A heuristic based batching and assigning method for online customer orders," *Flex. Serv. Manuf. J.*, vol. 30, no. 4, pp. 640–685, 2018.

[27] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Rob. Auton.*, vol. 12, no. 4, pp. 566–580, 1996.

[28] D. G. Kendall, "Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain," *The Annals of Mathematical Statistics*, vol. 24, no. 3, pp. 338 – 354, 1953.

[29] J. Kingman, "The single server queue in heavy traffic," in *Math. Proc. Cambridge Philos*, vol. 57. Cambridge University Press, 1961, pp. 902–904.

[30] H. Xu, "Optimal policies for stochastic and dynamic vehicle routing problems," Ph.D. dissertation, Massachusetts Institute of Technology, 1994.

[31] H. Ratliff and A. Rosenthal, "Order-picking in a rectangular warehouse: A solvable case of the traveling salesman problem," *Oper. Res*, vol. 31, pp. 507–521, 06 1983.

[32] S. Arora, "Polynomial time approximation schemes for euclidean tsp and other geometric problems," in *Proc. of IEEE FOCS*, 1996, pp. 2–11.