Project 2:

Multi-agent Planning and Delivery in a Restaurant

A. Benichou, W. Gharbi, G. Shindel, N. Turney, & T. Vu

CS420: Artificial Intelligence

Lafayette College

October 12, 2017

— Submitted to

Prof. Chun Wai Liew

A. Introduction

Despite its reputation of being a quiet college town, Easton, PA is on the up-and-coming. A cutting-edge ramen restaurant has just opened in downtown Easton with its focus on satisfying the curious palate of the younger generation. Inspired by modern restaurants of the Far East, this restaurant will have no human servers but instead robot agents. Customers will place their orders using a vending kiosk at the entrance of the restaurant, which displays menu items with corresponding prices. Once the customers have made their selection, they will pay at the kiosk and find an unoccupied seat within the restaurant themselves. When seats have been chosen, the customers then scan their receipts at the tables. The robot servers will be notified once an order is complete and will deliver the order to each customer's seat. Each robot must take an efficient path from the robot waiting station to the order counters, the order counters to the customer seat, and from the customer's seat back to the waiting station or to the order counters for another delivery. It is imperative that these robots completely avoid physically colliding with other robots, as per restaurant standards. This problem is a case study of an autonomous restaurant where robots are the physical agents executing its the operations. These agents are expected to operate in a dynamic environment. Our team had developed a path planning and navigation control system for the restaurant.

B. Parameters

I. Input

The input includes a combination of information from the restaurant, the customers, and on-field robot agents. A floorplan of the restaurant is a requirement prior to formatting an algorithm and was given by the restaurant. An estimate of customer arrival rate and total time spent are required to determine the ideal number of agents and help handling unmapped static obstacles. When a customer arrives at the restaurant, they are prompted to input their food order into a kiosk which links to the central system and is managed by the kitchen. Orders may consist of an entree (ramen), a drink, and a dessert. When an order is completed, the kitchen will notify our planning and delivery system of the contents of the order and the location of the customer in the restaurant. The different order components will be available for pickup at three separate locations around the perimeter of the

restaurant. As agents traverse the floor of the restaurant, they may encounter obstacles. There are several different kinds of obstacles that can be encountered: mapped static, unmapped static or dynamic obstacles. All static obstacles which remain throughout a day are already mapped to the system when a floorplan of the restaurant is delivered for preprocessing, such as tables, chairs, etc. If an obstacle encountered by an agent is classified as a static or unmapped obstacle, then it will be reported to the system as an input for proper management. When deliveries are completed or obstacles are encountered, agents will notify the system that they are available for the next order delivery or request rerouting.

II. Knowledge

Practical robotic agents need to embody prior knowledge about itself, its physical environment and the tasks that they will perform in order to learn quickly and perform safely. These server robots are physical agents that perform tasks by manipulating the physical worlds. They have effectors, such as legs, arms, and wheels, that assert physical forces on and move around the environment. They have sensors which allow them to perceive their environment by capturing signals generated by other sources of the environment. The robots will have passive sensors to detect obstacles along their course but they can only see obstacles that are directly in front of them, as well as in their immediate course of action. These obstacles will be reported to the central control system so that it can update path planning accordingly.

This restaurant has a predefined lay out of 3 rows or 3 rectangular tables where each side of the table can seat four people. Thus, the restaurant capacity is 72 people. In addition, the meals are decomposed into three separate components: drinks, ramen, and dessert where each component comes from its own kitchen located at different walls of the restaurant. The basic layout of the restaurant is shown in *Figure 1* where each of the tables is labeled with a letter and the seats on each side of the table are represented as blue squares. The order stations for the three different components of the meals are shown on three separate walls as “Drinks”, “Ramen”, and “Dessert.”

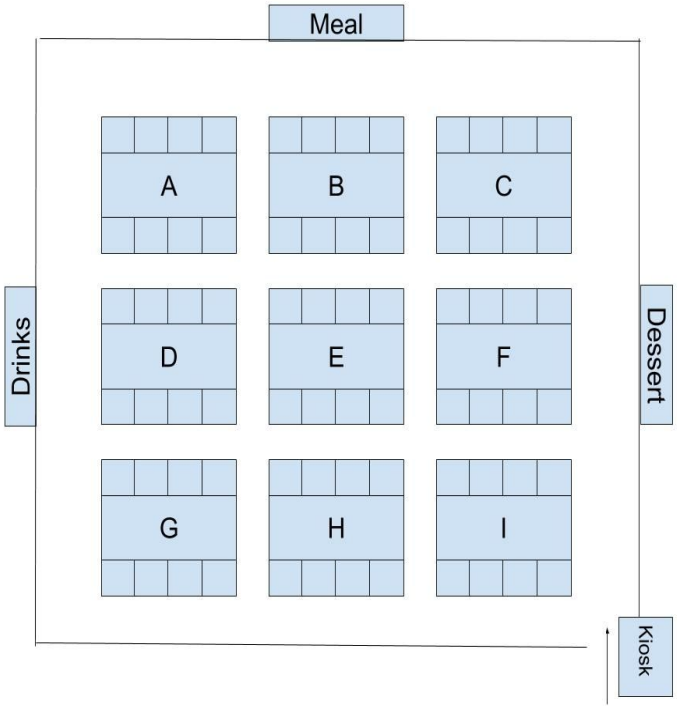

Figure 1: Layout of the restaurant

III. Output

After receiving a signal from the kitchen notifying the system that an order is completed, the system assigns the agents to tasks in order to deliver the orders to destinations. After an agent requests routing aid, the system routes agents to tables, order counters or waiting stations. When receiving information regarding an obstacle, the system uses the updated floor plan to reroute for the robots. When robots are adjacent, they can swap tasks which includes swapping orders and their various components. Using data received by the robots and from the kitchen, the centralized system produces efficient, obstacle-free paths from (1) the waiting station to the order counter, (2) the order counter to destination tables, and (3) the destination tables back to the order counter or the waiting station. In addition, the system should have procedures in handling all kinds of obstacles.

IV. Assumptions

Assumptions reduce the complexity of the problem by reducing use cases, creating a more homogenous problem state. Assumptions about the external environment and baseline capabilities of the system allow for the description of the best possible problem state. The assumptions that define our system and problem are as follows:

- Robots will have the ability to exchange food in one time step.
- The robot's sensors can detect obstacles directly in front of them.
- The robot will stop when it detects an obstacle, avoiding a physical collision.
- The robots have an infinite battery power source.
- The central control system and robots can synchronize in real time, with minimal delay.
    - The control system will know all the information about each robot at every time step.
- The initial layout of the restaurant, the tables, chairs, and pick-up counters, are static as initially described.
- Static and dynamic obstacles in the restaurant are on the same order of magnitude as the robots, taking up the same space as one robot.
- The restaurant map can be effectively decomposed into square cells.
- The customers will clean up their meals.
- The customers will order individually when in a group.
- The customer will carry their order to a new location after it is delivered.

V. Constraints

Constraints, imposed by the physical limitations of the problem, as well as by the software, must be considered to appropriately design a solution. The constraints on our system are as follows:

- All paths in the restaurant, excluding those along the pick-up counters, are as wide as one robot, allowing only one to travel through at a time.
- All collisions must be avoided.
- Each robot has a carrying capacity of 4 units.
    - Ramen has a weight of 2.
    - A drink has a weight of 1.
    - A dessert has a weight of 1.
- The robots can only detect obstacles in the direction they are facing or traveling.
- The robot's detection range is limited.
- Customers seat themselves without any restriction.
- Orders must be delivered to the location directly adjacent to the seat specified by the customer.

VI. Use cases

Use cases define the range of parameters and situations that our system must be able to handle. How the systems and its algorithms handle each case is described in-depth in section C.IV. All users are handled separately, as individual users. After placing an order, the user will select a seat at one of the tables and scan their receipt barcode at the location, informing the system where the user is seated. Customers in a group will order individually and seat themselves together at a table.

Seating is handled implicitly by the path finding system. If there are a small number of customers seated sparsely throughout the restaurant, there will be few restrictions on the robot's movement. Paths will be generated for each order and delivered by a robot. If there are a small number of customers and they are densely seated, the central control system will generate paths for each individual robot to an order location, grouping orders on robots if it is beneficial for efficiency. If a robot detects an obstacle, it will report it back to the control system, which will calculate a new path for the robots accounting for the obstacle if required. If the density requires orders to be swapped between robots in order to reach a seat, the robots will do so. A large number of customers are handled in the same fashion as a small number of customers seated densely, as they pose similar constraints on robot movement. A customer changing their seating location is handled based upon which step in the delivery process the robot is currently in. If a customer changes their seat before the order is picked up at the counter, the control system will be notified when they scan their barcode and calculate the path to the new location for the assigned robot. If the customer changes their seat while the robot is traversing to the first location, the control system will generate a new path using the current location of the robot as the starting location. If a customer changes their seat after the order has been delivered to the original location, we assume they will carry their order to the new seat. If a seating arrangement, combined with obstacles placed by the user, result in an undeliverable location, the robot will deliver to the nearest spot on the table.

Canceled orders require unique handling by the system. If an order is placed and the food is prepared but canceled before a robot picks up the food from the counter, the order is canceled and removed from the counter, as well as the queue of unassigned orders. If the order is canceled while a robot is in the process of delivering the order, the robot will be assigned a path by the central control

system to return the food back to the food counter. If an order is canceled after a robot has delivered an order, we assume that another customer will clean up the food as if it were trash.

VII. Evaluation Criteria

Evaluation criteria provide the system and context to evaluation the effectiveness of our system. We need to consider how a variety of factors, such as use cases and execution time, are handled by our system. With each factor, a list of relevant questions is composed to thoroughly assess the performance of the system. Our evaluation criteria are as follows:

- Use Cases
    - Can the system handle a variety of use cases effectively?
    - Are different use cases handled separately or is the system generic and flexible enough to handle multiple cases in the same way?
    - Can the system handle extreme use cases such as a lunch rush, dense groupings of customers, or many customers placed obstacles? How well does the system handle these situations?
- Time
    - How long does it take for the system to generate paths for each robot? In the general case? When there are many customers? When there are many obstacles?
    - How long does it take for an order to be delivered? In the general case? In the worst-case?
- Robot Pathing
    - Are the paths optimal for each robot in terms of time and distance? Suboptimal?
    - How are deadlock situations handled?
    - How are obstacles handled? Static objects like walls and tables? Dynamic obstacles like children, strollers, wheelchairs?
- Others
    - What is the optimal number of robots to handle the variety of use cases?
    - Is the system scalable and flexible?
    - What are the strengths of the system?
    - What are the limitations of the system?

C. Algorithms

I. Previous Attempts

1. Skeletonization

An alternative approach to decomposing the configuration space is skeletonization. This approach collapses the free configuration space to a one-dimensional representation which is known as the skeleton of the configuration space. A requirement is that the paths lie along the skeleton. This method offers more possible routes which deals with open space better. It is based on the idea that the shortest path consists of obstacle-free straight-line segments connecting all obstacles vertices from a start to end node. To construct a skeleton, the following steps must be taken: if the obstacles are polygonal, then the path will be polygonal and the paths inner vertices will be the vertices of the obstacles. These vertices can be used at the nodes of the skeleton. We decided not to utilize this approach because the free space of our restaurant was very clearly defined as paths that can either fit one or two robots at a time. As a result, the restaurant has a lack of wide, open space which is what skeletonization uses to model the connectivity. In addition, it counts on obstacles having a polygonal shape so that their edges can be used to construct a path.

2. Conflict Based Search (CBS)

An approach for pathfinding is Conflict Based Search (CBS) which is a two-level approach to solving multi-agent pathfinding [2, 11]. The high level of CBS utilizes a constraint tree: a binary tree where the root contains a set of agent location constraints. Each child node inherits the constraints of the parent node and adds an additional agent constraint. A node is classified as a goal node if there are no agent conflicts. The high-level system uses a best-first search of node costs, the number of constraints. Each node is processed by the low-level system, which utilizes any path finding algorithm but usually utilizes A* with true-shortest distance as the heuristic, for each agent individually [2]. These paths are then tested by simulating movement of agents along each of their paths. If no agents conflict, the node is returned as a potential solution, else the simulation is halted and the node is declared as a non-goal node. CBS can effectively handle a variety of agents and obstacles but still render several situations unsolvable due to the lack of exchangeability [2]. Moreover, CBS seems to be

less efficient compared to Flow-based ILP formulation, an algorithm that allows transferability, given the specific type of map of the restaurant. More details about the reason why exchangeable-task setup and Flow-based ILP algorithm were selected over CBS can be found in IV.4.

3. Token Passing with Task Swapping (TPTS)

An alternative multi-agent path finding algorithm that was considered was Token Passing with Task Swapping (TPTS). TPTS has each agent plan their paths sequentially to prevent conflicts [3]. Agents pass a token among each other, which determines if they can make a decision or not. Tasks are stored in a set; as new tasks are generated they are added to the set. When any agent reaches the end of their path, they request for the token. When the agent receives the token, they can choose a task from the set as long as the path of the agent does not end in the pickup or delivery location of another agent. TPTS allows for the tasks to include both unassigned tasks as well as incomplete tasks. Thus, an agent with the token can select a task that is assigned to another agent but that is incomplete as long as it is to the benefit of the overall system and fits the path selection criteria. As long as the assigned agent is enroute to the task's pickup location, another agent can take the task. If this occurs, the previously assigned agent requests the token. TPTS requires that the problem states of the multi-agent path finding problem are well formed: the number of tasks must be finite, there are no fewer non-task endpoints than the number of agents, and that for any two endpoints there is a path between them that traverses no other endpoints [3]. An endpoint is any location that may be scheduled as a pickup or delivery location. Because of the constraints imposed upon our system by the physical layout of the restaurant, we cannot guarantee that every state of the restaurant is well formed for a multi-agent pathfinding problem. Deliveries must occur in narrow corridors with other potential delivery locations adjacent on either side. Therefore, we did not select TPTS for our pathfinding algorithm.

II. Overview of the Final Design

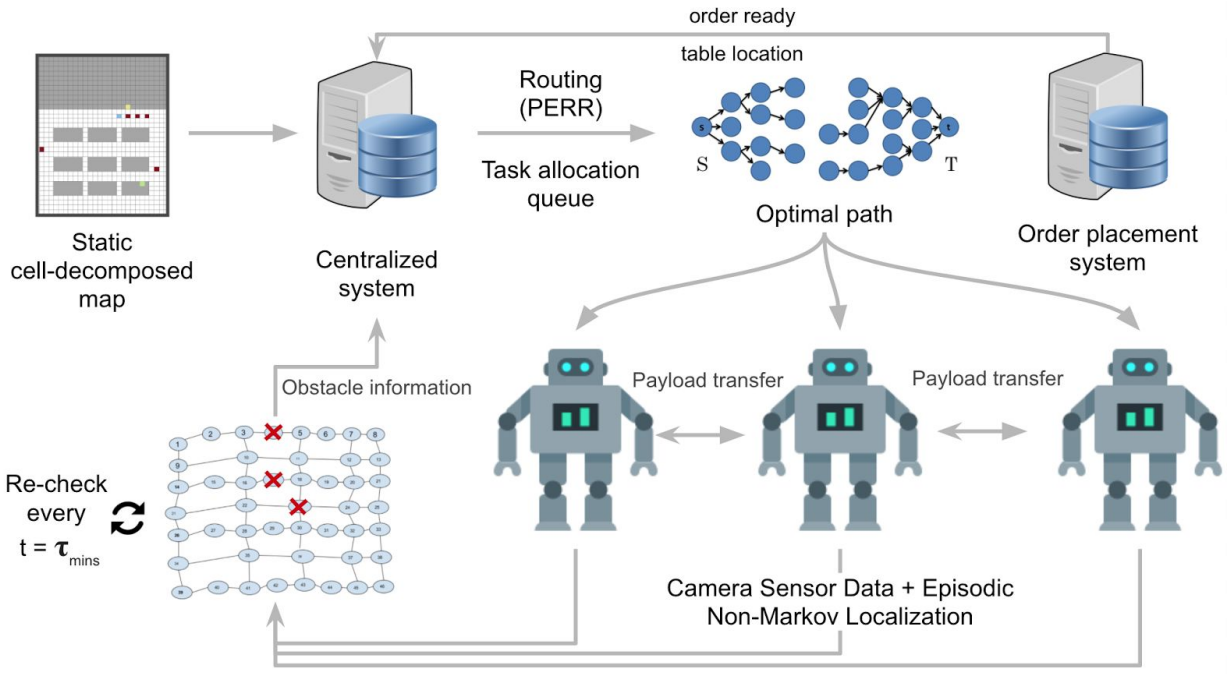

Figure 2: Overview of the architecture of our system

Our approach can be decomposed into multiple interconnected sub-problems. As *Figure 2* demonstrates, we have modeled our system through a pipeline architecture given that the nature of the problem imposes the use of multiple intelligent goal-based agents featuring sensors and effectors that can be acted upon in parallel.

The centralized system in our structured intelligent system acts as the central hub for commands and knowledge acquisition. Through a pre-processing step, the centralized system is initialized with the static cell-decomposed map of the restaurant (implemented as a Graph structure) augmented with the locations of permanent obstacles such as tables or chairs. An external order placement system managed by the restaurant (a combination of ordering kiosks and a database) is also connected to the centralized intelligent system through push-like signals that notify the system as soon as an order is ready for pick-up and delivery. The order placement system is assumed to provide the artificial intelligence with information about the items ordered as well as the destination (client's position in the map).

As soon as the first order is received, a cycle begins: A task allocation algorithm – detailed in section IV.3 – kicks-in and efficiently combines orders and determines (based on available robots and orders) which robots should delivery which order(s). As soon as the task allocation step is completed for a time step, a multi-agent path-finding algorithm solving the Package-Exchange Robot-Routing problem (flow-based ILP algorithm, detailed in section IV.4) allows the system to efficiently route robots from the food counter to their destination(s) (client locations) with the possibility of robots exchanging payloads (orders). The PERR-solving algorithm is run for all robots at every destination change (ex. when a robot finishes delivering an order and needs instructions to deliver the next order or when a robot finishes delivering all orders and needs instructions to go back to the waiting station).

Although the centralized system provides the bulk of intelligence in the system and allows routing of the robots in a discrete fashion (from cell to cell), the robot agents are also equipped with minimal intelligent algorithms to allow for continuous localization and obstacle avoidance within a single cell: Featured with camera sensors of limited field-of-view, the robots make use of an episodic non-Markov obstacle handling algorithm to map the environment around them, classify obstacles encountered, and respond properly to a type of obstacle.

The localization algorithm embedded in the robot agents then reports dynamic (transient) obstacles to the central system as a signal indicating which cell in the restaurant's map the obstacle was found in. This shared intelligence model allows for minimal data transfers over the network thus achieving minimal latency: Instead of having "dumb" agents that transfer the full information of their sensors to the central system to analyze (usually information about millions of pixel data from the camera sensors), we make use of the individual robots' hardware to locally analyze sensor data and report the analyzed results (usually just a few bits indicating the location of obstacles).

To differentiate between dynamic, static, and permanent obstacles added to the map, we make use of a timer system managed by the centralized intelligent system: The first time an obstacle is reported, it is considered transient and the node containing the obstacle would be disabled (robots aren't allowed to cross) for a limited time $\boldsymbol{\Gamma}$ determined through experimentation. This limited time will ideally be in the magnitude of a single time step. After $\boldsymbol{\Gamma}$ time steps have elapsed, the node is reevaluated. The next time a robot crosses the node either it finds an obstacle (in which case we consider it a static obstacle and deactivate the node again with an exponentially growing timeout) or it doesn't encounter the obstacle (in which case we disable the timeout and can safely assume that the

obstacle was dynamic). As the timeout grows, at some point the obstacles can be asymptotically interpreted as permanent obstacles, resulting in the blocked cells being removed from the map for a long period of time. For our problem, we focus mostly on dealing with unmapped dynamic and static obstacles, which will be discussed in greater detail in section IV.5. Long-term permanent obstacles can be handled similarly using the same timeout pattern.

III. Preprocessing And Knowledge Discovery

There are two preliminary steps that must be taken prior to utilizing the main algorithm: map decomposition and knowledge discovery. For map decomposition, the blueprint of the restaurant must be represented as a data structure that the algorithm can process. For knowledge discovery, data must be collected from current Easton restaurants about customer flow rate in order to determine the best number of robots to have in the restaurant at all times.

1. Approximate Cell Decomposition

In order to plan motions, the system must have an accurate model of itself and the environment surrounding it while avoiding collisions with both static and dynamic obstacles. For this, we have selected approximate cell decomposition which decomposes the free configuration space into a finite number of contiguous cells that are assigned coordinates and are used to create a connectivity graph which captures the adjacency relationships of these cells. This connectivity graph is used by the main algorithm to plan paths for the agents. The approximate means that the structure of the configuration space, as well as the obstacles, are approximated. In this way, all cells share a predefined shape of a square to aid in representation [9]. In addition, no matter the shape or size of the obstacle detected within a cell, it will be seen by the system and the robots a solid body that occupies a well-specified area of the workspace which is an entire cell [10]. This way, the robots are certain not to attempt to pass an obstacle on its route.

The first step in cell decomposition is to break down the blueprint of the restaurant into a finite set of non-overlapping square cells. Compared to the layout of the restaurant shown in *Figure 1*, *Figure 3* shows the same layout on top of a grid. From *Figure 3,* it is easy to differentiate the configuration space. The configuration space of the restaurant represents the workspace and it can be

split into two subspaces: free space and occupied space. Free space is the space of all configurations a robot may attain and occupied space is the space of unattainable configurations [10].

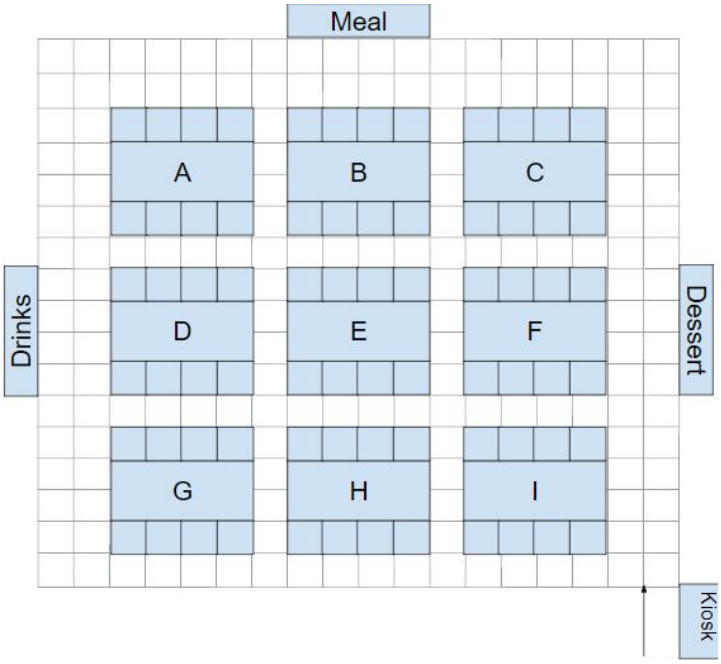

Figure 3: The restaurant in grid representation

The next step in cell decomposition is to assign each cell a coordinate depending on which row and column is falls under. *Figure 4* shows the grid representation of the restaurant with each cell having unique coordinates.

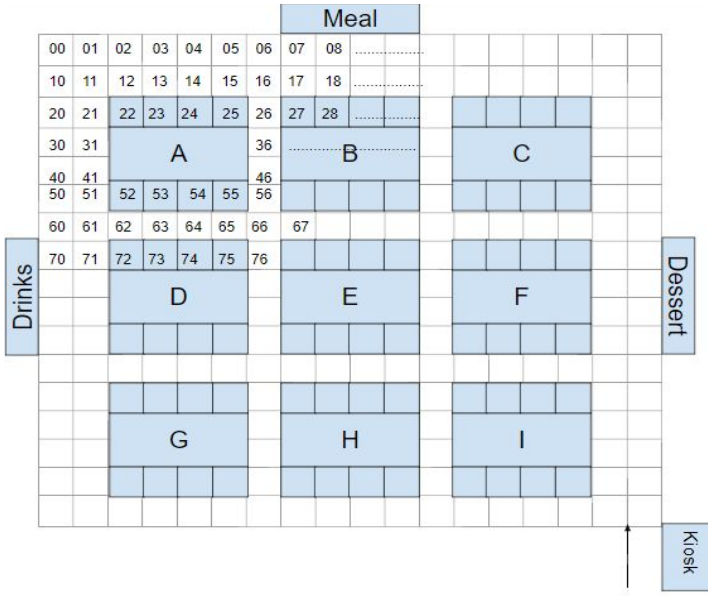

Figure 4: The grid representation with assigned coordinates

The final step of cell decomposition is to use the cells with their assigned coordinates to create a connectivity graph which represents the cells in the configuration space. *Figure 5* shows the beginnings of this connectivity graph. This undirected, connected graph is made up of nodes corresponding to cells in the decomposition and edges corresponding to the adjacency of contiguous cells. In particular, two nodes are connected by an edge if and only if the two corresponding cells are adjacent [9].

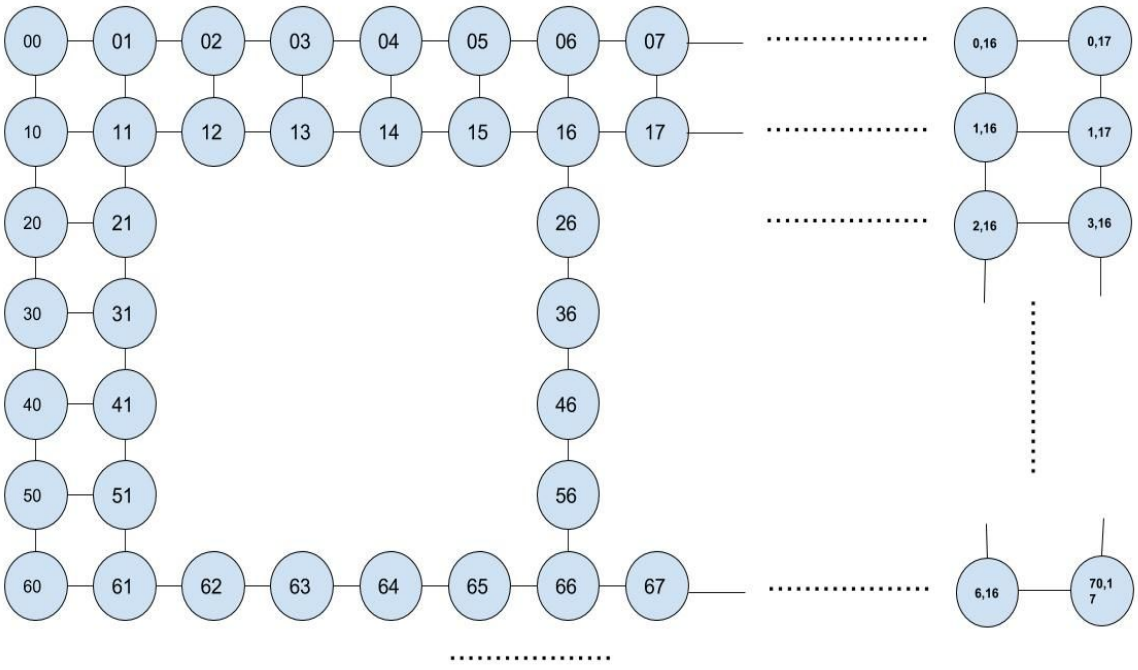

Figure 5: The connectivity graph whose nodes correspond to cells in the free configuration space

2. Survey and Simulation

As part of the case study of the restaurant, one important factor that will contribute to the costs/benefits of implementing an automated robot system is the number of robots required to be bought as well as the optimal number of operating robots to deliver meals efficiently. A major uncertain element which influences transport planning is the arrival time of customers because this places demand on the restaurant to prepare their orders and on the agents to deliver their orders. For this reason, our team will survey current Easton restaurants about their customer flow rates at the times and days that they are open. This can be done by asking the restaurants directly if they keep track of this information or by surveying the restaurants ourselves. This customer flow rate data can be

averaged and then used to create a simulation which we can use to test the best number of robots to have in the restaurant at all times, taking into account the flow rate of customers, the estimated order preparation time of the restaurant and the layout of the restaurant.

A reasonable metric that we can optimize for is the number of orders waiting to be delivered at each time-step. We can evaluate our algorithm's performance by plotting the number of orders in the queue as a function of an incremental number of available robots.

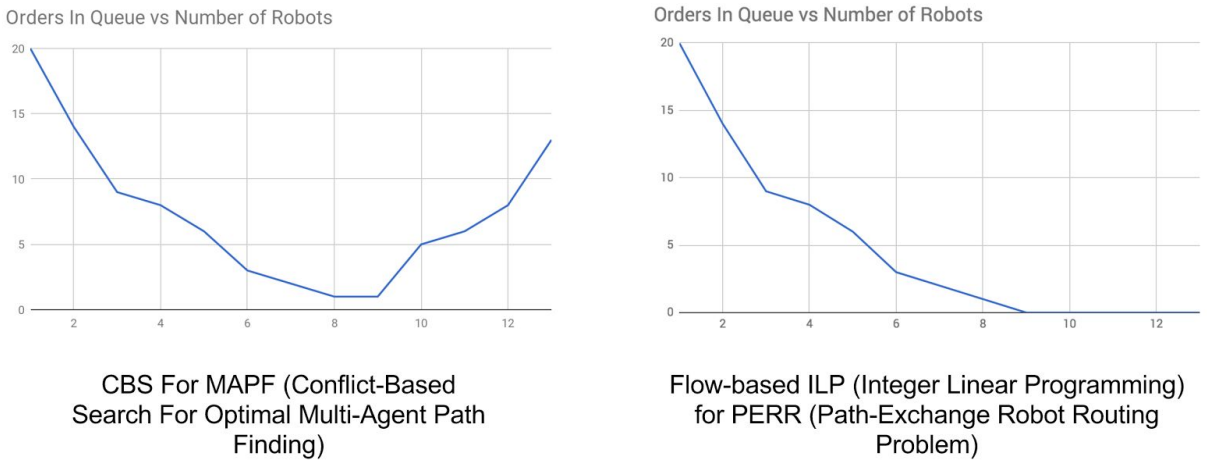

Figure 6: Estimated curve shape of the average orders in queue plotted against the number of robots used, for both CBS and PERR approaches detailed above.

If we were to use one of our attempted non-exchangeable payload solutions, we would expect to see a sharp decrease in queued orders as we increase the number of robots employed until we hit an optimum, then the number of queued orders would rapidly increase since more robots cause more collisions and clutter the restaurant's space making it harder for each-other to deliver their respective orders. However, since we are using an exchangeable-payload, we will not see an increase in queued orders as we add more robots (since even at the worst-case scenario where all cells are occupied by robots, a meal can reach the customer through multiple exchanges between robots). On the other hand, we will see a continuous decrease in queued orders indicating that after the optimal number of robots, adding more robots will yield no improvements in delivery time. Thus, we will need to choose a reasonable number of robots that both minimizes cost and maximizes efficiency.

IV. Main Algorithms

1. Preview

We identified three main challenges while tackling the problem of multi-robot navigation in a restaurant: collision avoidance, continuous delivery, and unmapped obstacle handling. First of all, this problem requires generating paths for each and all robots to move from the waiting station to the order counter, from the counter to the customer table, and from the table to the next targeted location without colliding with each other. In order achieve these collision-free routes, we formalize the problem as a Package-Exchange Robot-Routing (PERR) problem and solve it with an optimal, reduction-based solution, namely Flow-based ILP formulation [1, 2]. The details are presented in section IV.4 below.

Second, although the problem of multi-agent pathfinding (MAPF) is well studied, a majority of the work focuses on a one-trip version, where each agent has only one task to complete. In many real-world situation, however, robots tend to engage in multiple jobs with back-to-back trips. Ma et al. refer to this as a lifelong multi-agent pickup and delivery (MAPD) problem [3]. Our problem is similar to this. Robots in the restaurant are continuously assigned with back-to-back tasks: moving from the station to the counter to pick up order, delivering order from the counter to a table, returning to the station or to the counter for new order, etc. Thus, inspired by Ma et al.'s Token Passing with Task Swaps (TPTS) algorithm, we added a higher layer of event-handler on top of the Flow-based ILP, which dynamically recalculates routes every time the environment status is updated. These events can include new orders being ready, robots successfully delivering meals, robots successfully arriving at the waiting station, etc. This approach takes advantage of the small size of the restaurant map (about 18x17 grid) and Flow-based ILP's promising performance presented in [1]. Since the runtime for pathfinding algorithm can be as low as 2s for 10 agents in a 20x15 grid map, we do not expect rerunning it upon new events to produce significant negative effects.

Another challenge that we focus on is how to handle unmapped obstacles. Flow-based ILP, as well as most PERR and MAPF algorithms, assumes the map to be static and free spaces will stay "free" as long as no other agents are occupying them. In other words, the only type of collision that it handles it agent-agent collisions. For our restaurant problem, there is another type of potential collision: agent-obstacle collision - when robots run into real-time dynamic (transient) obstacles such

as customers or customers' luggages while executing its calculated path. To handle this, we exploit our proposed Event Handler and consider the detections of obstacles as another type of event. Upon the occasion of unmapped obstacles, a classification algorithm inspired by Biswas et al.'s Episodic Non-Markov Localization algorithm [4] will be deployed to categorize the objects into static obstacle and dynamic obstacles. Depending on the obstacle type, the system will then make system-wide updates on robots' paths via Event Handler. Obstacle handling is explained in detail in section IV.5.

Challenge	**Solution**
Collision avoidance	- PERR: consider the problem as a PERR problem - Flow-based ILP: apply Flow-based ILP formulation to solve
Continuous delivery	- Event Handler: dynamically recalculate robot's paths
Obstacle handling	- Obstacle classification: categorize static vs. dynamic obstacles - Event Handler: make system-wide updates

Table 7. Challenges and our approaches

On one end, our system is centralized and decoupled, which separately handles all planning tasks such as order assignment and pathfinding. On the other end, on-field robots execute the generated paths, detect unmapped obstacles, and report back to the main system for updates of instructions.

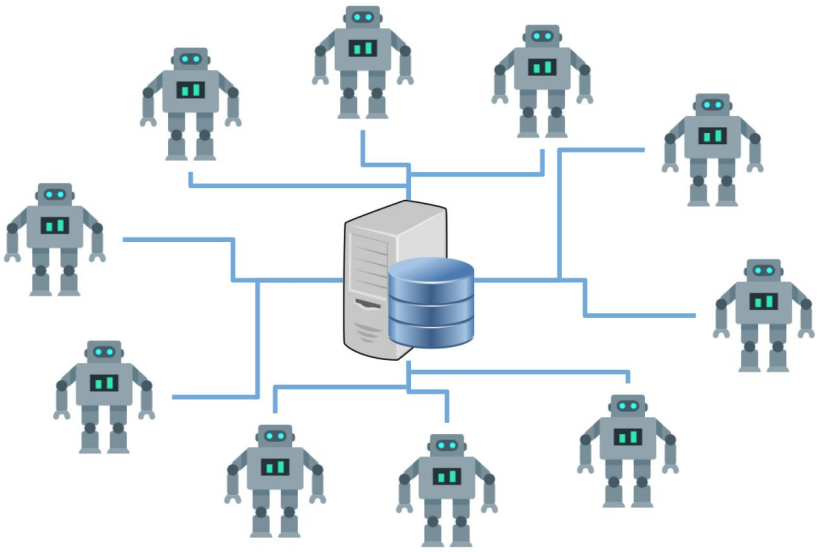

Figure 8: A centralized system-robot architecture.

2. Algorithm

Let's first define the following terms:

- $orderTasks = pickUpAt(counter) + deliverTo(customerSeat)$
- $agentTasks = sum\ of\ orderTasks\ for\ the\ agent(without\ duplicates)$
    - Ex:
        - 1 order of ramen → *pickUpAt(ramenCounter)* + *deliverTo(A)* → 2 tasks
        - 2 orders of ramen for 2 customers) → *pickUpAt(ramenCounter)* + *deliverTo(A)* + *deliverTo(B)* → 3 tasks
        - 1 full order (drink, ramen, dessert) → *pickUpAt(ramenCounter)* + *pickUpAt(drinkCounter)* + *pickUpAt(dessertCounter)* + *deliverTo(A)* → 4 tasks

Here is the main algorithm (inside Event Handler):

```
// Check for new events for all robots at each time step t
If newEvent == ordersReady
|       For all free agents:
|       |       run TaskAssignment
|       |       |       Input: list of (order, customerSeat)
|       |       |       Output: map of (agent, pickupLocation, deliveryLocation)
|       |       |       Details: (see Task Assignment section)
|       |       End
|       End
|       Run Pathfinding
|       |       Input: graph, currentTimeStep, list of (agent, deliveryTask)
|       |       Output: a map of (agent, path)
|       |       Details: (see Pathfinding section)
|       End
End
If newEvent == obstacleDetected
|       run ObstacleHandling
|       |       Details: (see Obstacle Handling section)
|       End
End
If newEvent == taskFinished
|       If agentTask > 0
```

```
|       |       re-run Pathfinding with
|       |       |       this agent: (currLoc, nextDest)
|       |       |       other agents: (curLoc, currDest)
|       |       End
|       End
|       Else
|       |       this agent = free agent
|       |       If ordersReady = 0
|       |       |       re-run Pathfinding with
|       |       |       |       this agent: (currLoc, waitingLoc)
|       |       |       |       other agents: (curLoc, currDest)
|       |       |       End
|       |       End
|       End
End
If newEvent == destinationChanged
|       re-run Pathfinding with
|       |       this agent: (currLoc, newDest)
|       |       other agents: (curLoc, currDest)
|       End
End
```

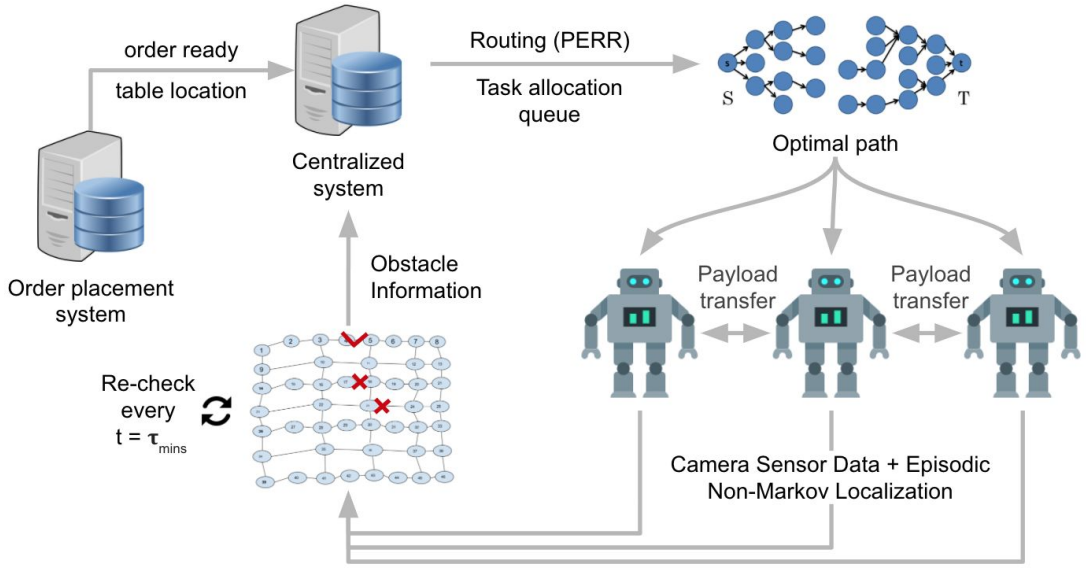

Figure 9: Overview of the main system

3. Task Assignment

As the order placement system (ordering kiosk + database system) starts sending signals to the centralized system indicating that orders are ready to be picked up and delivered to customers, the centralized system needs to figure out an efficient assignment of tasks to robots to minimize the delivery time for each customer.

Although an intuitive way to assign tasks would be to prioritize orders with the closest delivery location to each robot, this mechanism defies the first-come first-serve nature of restaurants and biases each customer's service quality based on their sitting location (seats near the food counters will always be served first). This led us to think of an intelligent task allocation system that doesn't infringe the first-come first-serve principle.

Since robots are allowed to pick up individual items of an order (Ramen/Drink/Dessert) separately, we decided to take advantage of this parameter and combine orders when possible as to maximize the robot's capacity and thus, minimize the number of trips necessary to deliver orders while at the same time maintaining the first-come first-serve paradigm.

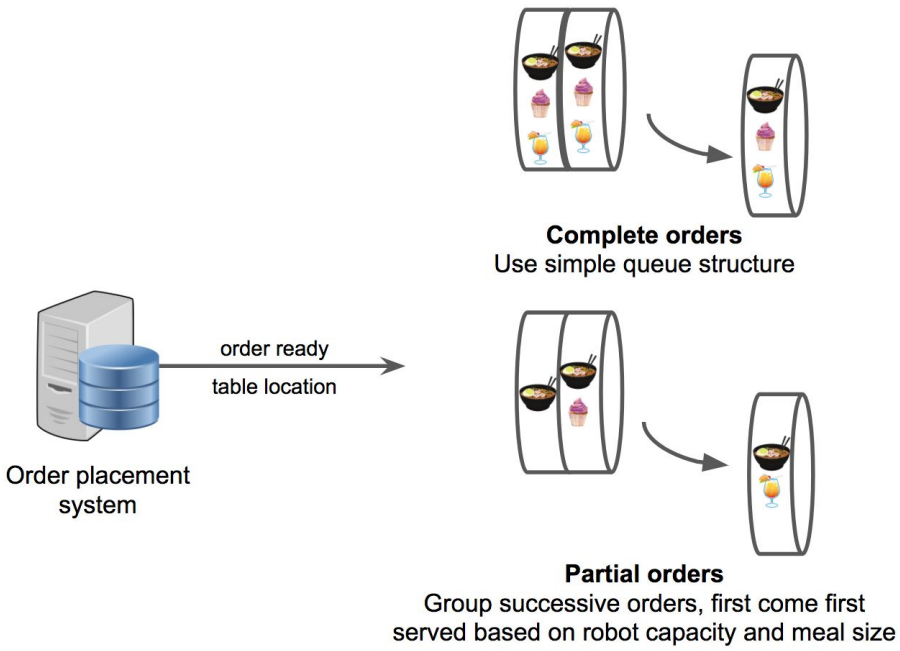

Figure 10: Overview of the task assignment step

Our task assignment algorithm consists of a sorting/combining step that, given a list of orders (comprising of meal elements, destination / customer location and a timestamp indicating when the order was made available), splits the orders into complete (3-element meals worth a full robot's capacity) and partial (less that a robot's capacity). Orders are then sorted into queues based on their timestamp (to preserve the first-come first-serve order). Available robots will then be assigned orders from either of the queues (based on the timestamp of the top elements in the queue) with the possibility to "pop" more than one element from the "partial orders queue" if the robot is able to transport more than the top order in one trip.

Example. Given the following orders:

00001	Ramen + Drink (total cost 3)
00002	Ramen + Ramen (total cost 4)
00003	Drink (total cost 1)
00004	Drink + Dessert (total cost 2)

the system would create two queues, the first one containing only order 00002 (the only "complete" order) while the second one contains the other three "partial" orders. When the first robot is ready, the central system would compare the timestamps of the top elements of both queues (in this case, 00001 and 00002). Since 00001 has a smaller timestamp, it will be prioritized. However, since 00001 is "partial" (of cost 3), we can look at the next element in the partial queue and try to fit it in on trip. Since the next order is 00003 (of cost 1), the system combines the orders and assigns the available robot to ship 00001 and 0003, then assigns the second available robot to 00002 and finally 00004.

Tasks are re-evaluated every time a robot becomes available or an order is added to the system (for example, if by the time we are assigning order 00004 (of cost 2) another order of cost 1 or 2 is placed immediately after 00004 or immediately after a series of complete orders following 00004 then 00004 will be combined with that order and delivered in a multi-stop fashion).

4. Pathfinding

a. What is PERR?

Package-exchange robot-routing problem (PERR) is modified version of the standard multi-agent path-finding problem (MAPF). In PERR, each agent is in charge of delivering one package to a predetermined destination and any two agents in adjacent locations can exchange packages [1]. The problem is formalized as follows:

- *Input* :
  - *Undirected connected graph* $G = (V, E)$
  - *M packages* $\{P_1, P_2 ... P_M\}$
  - *Each* P_i *includes* :
    - S_i = *Source vertex*
    - D_i = *Destination vertex*
- *Define* :
  - $l_i(t)$ = *vertex of* p_i *at time* $t = 0 ... \infty$
- *Constraints for a solution* :
  - *All packages start at source vertices* : $l_i(0) = S_i$
  - *All packages end at destination vertices* : $\exists T_i^{end} : l_i(t) = d_i \ \forall t > T_i^{end}$
  - *At every time step, packages either remain at the current vertex or move to an adjacent one* : $l_i(t) = l_i(t+1)\ OR\ (l_i(t),\ l_i(t+1)) \in E\ \forall i, t$
  - *Exchanges happen in one time step* : $l_i(t) = l_j(t+1)\ AND\ l_j(t) = l_i(t+1)$
  - *Only one package can be at a vertex at a time* : $l_i(t) \neq l_j(t)\ \forall i, j, t$

K-type Package-Exchange Robot-Routing problem (K-PERR) is a generalization of PERR, where the packages and their destinations are partitioned into K types. Agents can only exchange two packages of the same type. Thus, MAPF problem is an instance of K-PERR, particularly M-PERR, where the number of type is equal to the number of packages, resulting in no package exchange. In our case, we consider the restaurant problem as 1-PERR problem where all packages are exchangeable. A package can be any number of dishes that a robot is currently carrying. For instance, a “package” can range from a full four-unit meal (1 ramen + 1 drink + 1 dessert) to a single glass of water or even nothing at all (when robot travelling to the order counter).

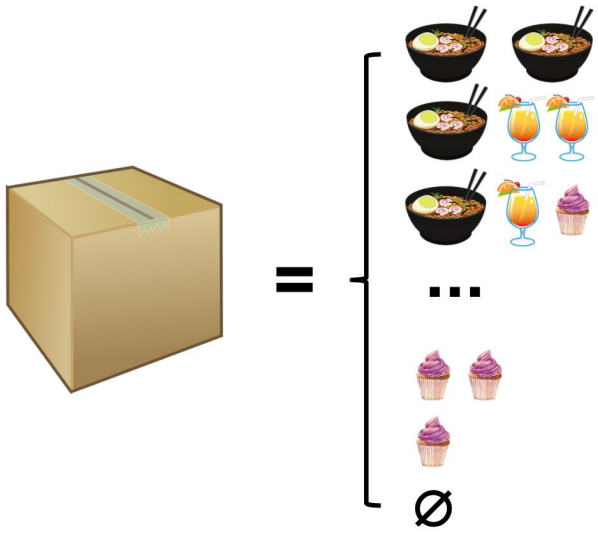

Figure 11: Examples of possible combinations of ramen, drink, and dessert that can be contained in a package carried by a robot servant.

b. Why PERR?

One observation that is shared between a PERR problem and our restaurant problem is that packages or orders are non-anonymous or non-exchangeable, meaning each package has a specific assigned destination and cannot be replaced by another package. However, it does not matter what agent actually delivers those orders. Thus, the agents themselves are anonymous. This characteristic enables us to apply a PERR view to our restaurant problem and take advantage of PERR's strengths. To illustrate this, we will compare PERR formulation (with package exchange) with the standard, well-studied MAPF formulation (without package exchange).

First of all, it is proven by Ma et al. that all instances of PERR are solvable and solution with polynomial makespans (total trip length) and flowtimes (sum of trip lengths) can be found in polynomial time. This is not the case for MAPF instances; in certain cases, a MAPF formulation might potentially result in no solution or inefficient ones. In *Figure 12* (a), two agents need to cross-deliver two packages but there is only one edge connecting two vertices. Without exchangeability, no solution exists for this example. In *Figure 12* (b), the situation is similar to (a) but now there is another path, a long one, that an agent can use. In this case, even if the example is solvable, it would still be much less efficient without package exchange.

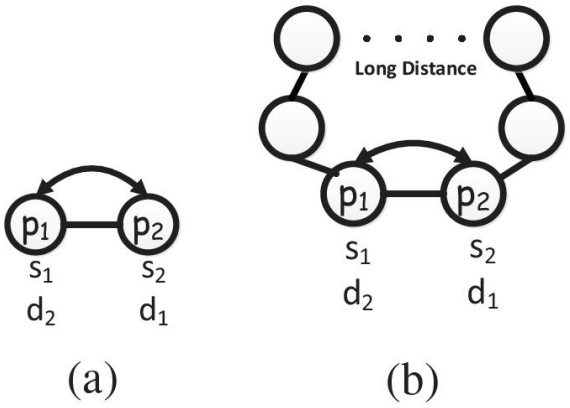

Figure 12: Examples where MAPF is inefficient. P_i is an agent with package i. s_i and d_i are source and destination vertices of package i respectively.

When compare two algorithms for PERR and MAPF, namely Flow-based ILP and CBS, we were convinced that PERR and Flow-based ILP is more applicable to our problem. We considered CBS as a representative solver for MAPF and a candidate for our problem because it appears to be one of the most flexible, well-studied, and well-performed among other state-of-the-art algorithms. However, although CBS work better with narrow pathways than with open space, the experiments in [6] and [1] show that CBS might not be the best for minimum width of corridors [6]. shows that when W=1 (pathway's width = robot's width), algorithms such as EPEA* and ODrM* outperform CBS. Moreover, [1] shows that Flow-based ILP significantly outperformed CBS when dealing with the Two-Loop map (*Figure 13*), which is an essential component of our restaurant map. The algorithm took less than 0.1 seconds to solve all instances while CBS was reported to have exceeded the 10-minute limit in some instances.

Thus, after considering the algorithm's feasibility and high performance with Two-Loop problem, we decided that PERR and Flow-based ILP can be a good approach to our restaurant navigation problem.

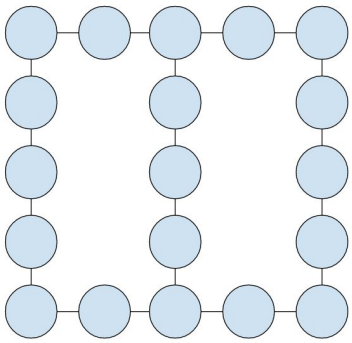

Figure 13: An example of Two-Loop problem

c. How to solve a PERR?

One way to solve a K-PERR instance is to reduce it to a known problem and utilize existing solvers. In general, a K-PERR problem can be reduced to an Integer Multicommodity Network-Flow problem (IMNF), which can be expressed as an Integer Linear Program (ILP). Ma et al. named this reduction-based approach as Flow-based ILP formulation [1]. We refer to it as Flow-based ILP for short. Given a flow network and K commodities, each with a source and a sink, IMNF ask to find an assignment for all source-to-sink flows that satisfies the four constraints of link capacity and flow conservation. From our point of view, after reduction, each valid flow would correspond to a valid robot path and a valid flow assignment correspond to a valid routing assignment for our restaurant problem. Thus, in order to exploit countless solvers of IMNF, we must create a flow network to represent our problem. Such network can be constructed from all possible robot movements as follows:

- *Input* :
  - $a\ K-PERR\ instance$
  - $a\ fixed\ number\ of\ time\ steps\ T$
- *Network construction* :
  - $Create\ 2\ vertices\ v_{in}\ and\ v_{out}\ for\ each\ vertex\ in\ the\ original\ graph\ at\ time\ t$:
    $V' = \cup_{v \in V} \cup_{t=0}^{T} \{v_t^{in}, v_t^{out}\}$
  - $For\ each\ package,\ set\ both\ supply\ at\ source\ and\ demand\ at\ destination\ to\ 1$:
    $(s_i)_0^{out} = (d_i)_T^{out} = 1$
  - $E' = all\ directed\ edges\ with\ unit\ capacity$:
    - $\forall v \in V, t = 0 \ldots (T-1),\ create\ 1\ green\ edge\ (v_t^{out}, v_{t+1}^{in})$

→ *representing robots staying at a vertex*

- $\forall (u,v) \in E,\ t = 0 \ldots (T-1),$ *create* 2 *black edges* $(u_t^{out}, v_{t+1}^{in}),\ (v_t^{out}, u_{t+1}^{in})$

  → *allowing robots to move to an adjacent vertex or exchange a package*

- $\forall v \in V, t = 0 \ldots (T-1),$ *create* 1 *blue dashed edge* (v_t^{in}, v_t^{out})

  → *limiting at most one robot staying at a vertex at a time*

- *Output* : *flow network* $N(V', E')$

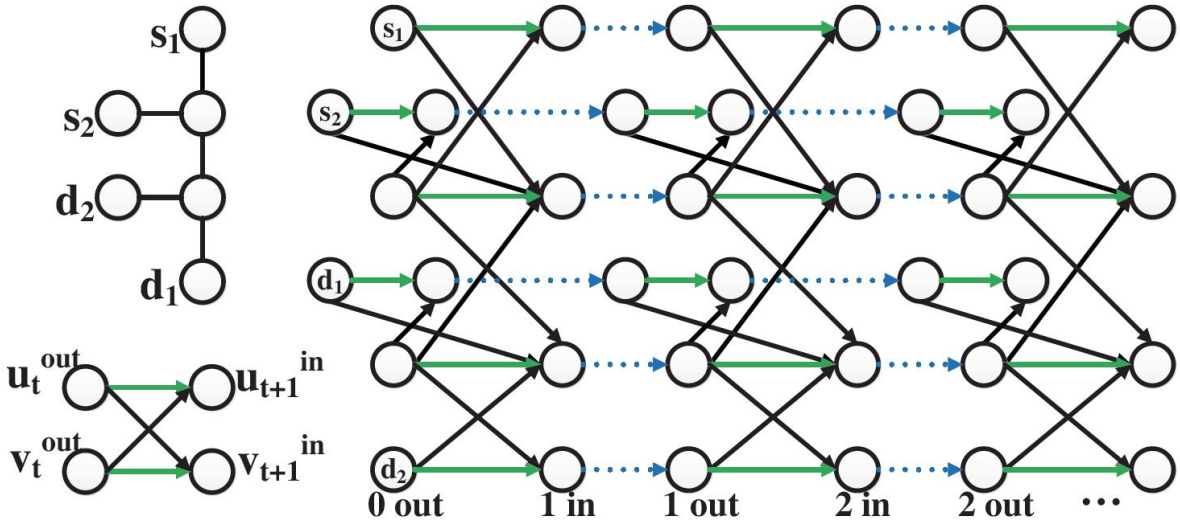

Figure 14: Illustration of IMNF reduction from PERR

Because every feasible integer multicommodity flow on network $N = (V, E)$ now yields a K-PERR solution with makespan of at most T, we can find the optimal solutions by minimizing the number of time steps T (makespans). This can be done through a binary search with complexity of $O(\log|V|)$ [1] on the range:

$$max[shortest\ paths\ (s_i, d_i)] < T < (M-1)(2|V|-3)$$

With this process of network construction, solving a K-PERR problem can generally be reduced to solving an IMNF problem. However, since we only using 1-PERR for our restaurant, the problem becomes much simpler. When all packages are of the same type, the IMNF problem can be easily converted to a Maximum Flow problem with all supply and demand values of one. This maximum flow instance can be solved by any polynomial time algorithm such as Dinic's algorithm (complexity $O(V^2E)$) [7] or Ford-Fulkerson method (complexity of $O(E \max|f|)$) [8].

5. Obstacle Handling

Obstacles are handled by an algorithm inspired by episodic non-Markov localization[4].

a. Inspiration from Episodic Non-Markov Localization

Episodic Non-Markov Localization classifies obstacles into three categories: Long-Term Features, Short-Term Features, and Dynamic features. We relate these three categories to our problem as mapped static objects (reflected in the floorplan inputted at preprocessing such as tables, chairs, etc.) which are always in the restaurant, unmapped static objects (wheelchairs, large bags, strollers, etc.) which usually remain for the duration of a customer's experience, and dynamic objects which only briefly obstruct passage and are currently in motion or will be in motion almost immediately. A Markov process is considered to be predictable based solely off its present state just as well as it could be predicted knowing that process' past and/or future. Episodic non-Markov localization considers the process observed as non-Markov, or considered more predictable based on knowledge of the process' future and past. We considered our obstacles to be non-Markov processes because observations stored overtime help better understand and handle the obstacles.

The episodic nature of episodic non-Markov localization allows a combination of manageable runtime and sufficiently informed decisions. State information regarding an obstacle is stored and referenced over time to properly classify and track the obstacle. As the number of recorded observations of a point increases, the runtime required to process new observations of the same point increase. At the same time, as an observation ages, its pertinence to the current state of the environment may no longer be significant. For these reasons, Episodic Non-Markov Localization defines 'episodes' as periods of time over which observation are considered relevant. We believe these reasons also relate to the obstacles encountered in our problem. Therefore, we define an episode over which observations should remain relevant within our system to a duration related to the length of an average customer experience.

b. Episodic Non-Markov Obstacle Classification and Handling

Our algorithm for obstacle classification and handling uses relevant elements from Episodic Non-Markov Localization to efficiently and intelligently manage previously unknown blockages in an

agent pathway. Obstacle handling is initiated when an agent encounters an obstacle which prevents its desired motion in a timestep **τ**. The agent is informed of an obstacle by a single-cell length range finder which points in the forward direction of travel. In the timestep **τ** in which an obstacle is observed, the obstacle is considered dynamic. The agent will remain stationary for a short time period **ς** in the previous location. **ς** should be in the magnitude of a single to a few timesteps. At time **τ** + **ς** the cell in which the agent desires to move will be reevaluated. If a dynamic obstacle like a customer in motion was previously encountered, the obstacle should be cleared quickly and the agent's observation should report an ability to continue its route. In this case the agent will inform the central system of it's delay and request a solution to the new PERR problem accounting for delay **ς**.

If, at timestep **τ** + **ς** an observation reports the obstacle remaining, it is now assumed to be a static, unmapped obstacle. These obstacles are expected to include wheelchairs, baby chairs, large luggage, and other objects a customer may need to set aside while seated in the restaurant. At timestep **τ** + **ς** the agent will inform the central system of the static obstacle's location and the edges connecting the corresponding cell in the decomposed graph will be removed. A new PERR instance will then be generated given the state at **τ** + **ς** without the blocked edges and the central system will recalculate a solution for all agents. The existence of this obstacle will be assumed for an initial period of time **λ**. This initial time period, **λ**, will be related to the duration of the average customer experience in the seating area. At time **τ** + **ς** + **λ** the obstacle will be assumed to have clear and the edges will be re-add to the graph, however the observation of the obstacle will remain in the corresponding cell's history. A recalculation is not required for agents enroute. If, at a time later than **τ** + **ς** + **λ**, the cell is successfully traversed, then the obstacle is known to have cleared and the history for the cell can be forgotten. If, on first encounter with the cell after time **τ** + **ς** + **λ**, the cell is still obstructed, a new timer is added and the process repeated for handling an unmapped static obstacle, however, the time period **λ** will be asymptotically reduced.

c. Why not Episodic Non-Markov Localization?

Various concepts presented in Episodic Non-Markov Localization proved valuable to handling the obstacles encountered in the restaurant problem, however, the algorithm itself served goals unnecessary and not achievable by our system. The algorithm was designed to localize agents in an unknown environment by repeatedly observing the location of a point in their forward direction with a laser range finder of much farther range. A point could be distinguished in a 2D plane using the angle

of orientation of the sensor and the range to the point. Our algorithm already can localize and navigate permanent obstacles using the cell decomposition of the floorplan. Also, objects were distinguished by the number of repeated observations over a period of time. Certain locations in the environment which often attracted a static obstacle became regions of unmapped static object. This can be seen as cars in parking spaces in the image below. Our environment has a normal distribution of likelihood a dynamic or static obstacle will appear at almost any location. Applying the same technique may lead to believing multiple sightings of a moving person in a path to be a static object, even though it is a different dynamic object each time. This problem is also related to the fact that the algorithm was designed to traverse 2D space. Most of the pathways through our floorplan are 1D and observations are only ever useful in up to four directions around a intersection and two direction in a single lane path.

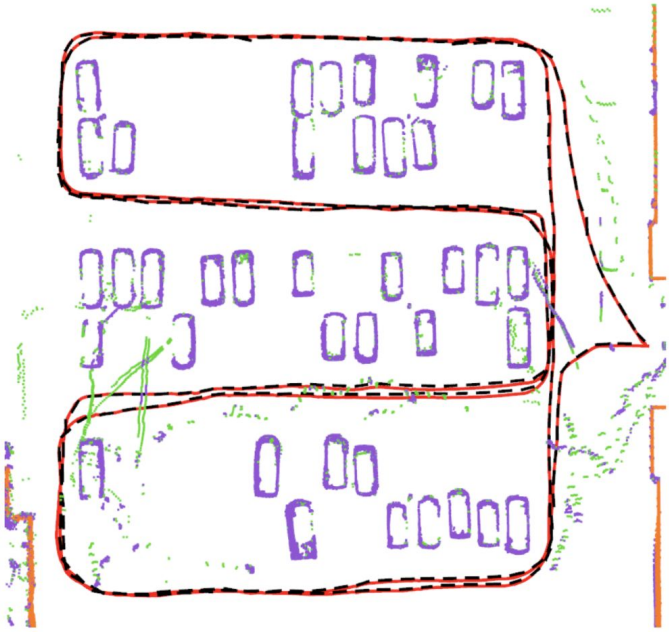

Figure 15: Example Output of Episodic Non-Markov Localization from a parking lot. The red line represents the path the robot traversed, the black dotted line is the expected path, orange markings are static objects, purple markings are static unmapped object, and green are dynamic objects.[4]

V. Algorithm evaluation

Criteria	Our solution
Use cases	
Can the algorithm handle different use cases?	- Yes. Please refer to section A. VI. for more details
Are different use cases handled separately or is the system generic and flexible enough to handle multiple cases in the same way?	- Different use cases are either automatically handled by the algorithms themselves or handled using similar method via Event Handler - The specific number of customers as well as whether they arrive in groups or individually do not affect the validity of system. - Cases such as customers canceling orders, customers blocking robots' paths, etc. are generally considered new events that alter the status of environment, and thus, yield system-wide updates.
Can the system handle extreme use cases such as a lunch rush or dense groupings of customers? How well does the system handle these situations?	- High frequency of customers coming results in frequent updates of new events. In such case, Event Handler would call pathfinding algorithm frequently. In worst-case scenario, Flow-based ILP is executed at every time step. - Dense grouping of customers are efficiently handled by optimal pathfinding algorithm with exchangeability
Time	
How long does it take for the system to generate paths for each robot? In the general case? When there are many customers? When there are many obstacles?	- Based on [1], the runtime of optimal Flow-based ILP for a 20x15 grid map with 10 robots is about 2 seconds. Since our restaurant map is roughly of the same size, we expect a similar runtime for the pathfinding algorithm. However, pathfinding does not happen every time step but rather only when new events happen. Thus, on average, the runtime at each time step is expected to be less than 2s

	- Having more than 20 robots simultaneously operating in a small map like this could increase the runtime for pathfinding algorithm to 8s or higher. - If runtime is a bottleneck, we can potentially switch to using suboptimal Flow-based ILP, also introduced in [1]. In most cases, suboptimal Flow-based ILP would cut the runtime burden in half.
How long does it take for an order to be delivered (in terms of makespan)?	- For a 20x15 grid map, makespan is 20 to 30 steps for 10 robots and 23 to 33 steps for 20 robots - We chose to prioritize and optimize makespan for our algorithm. However, optimizing flowtime is also an option when using Flow-based ILP formulation.
Robot Pathing	
Are the paths optimal for each robot in terms of time and distance? Suboptimal?	- Flow-based ILP optimizes the makespan (the longest time for an agent to deliver an order) and generates optimal paths.
How are obstacles handled? Static objects like walls and tables? Dynamic obstacles like children, strollers, wheelchairs?	- Static obstacles are made known to the system through the pre-processing stage (assumed to be part of the restaurant layout). - Dynamic obstacles are handled through a trial-and-error system that combines collaborative knowledge from the multiple agents' sensors with dynamic timeouts that block the region where the obstacle was detected for a certain time period then re-check and increase the timeout whenever the obstacle is encountered again, giving an asymptotically permanent interpretation of the obstacle if it obstacle persists for a long period of time.
How are deadlock situations handled?	- Agent-to-agent deadlock situations are handled inherently by the exchangeability property of our algorithm solution combined with the ability to re-route whenever a new obstacle is discovered. - Agents trapped in a space by transient obstacles (think placing two obstacles on both ends of a passage while a robot is crossing it)

	would not be handled by the system, however audio-visual signals can be emitted to inform customers that their belongings are blocking the robot. The same signal can be transmitted to a human supervisor of the network.
Others	
What is the optimal number of robots to handle the variety of use cases?	- Based on the experimental results from [1], we would suggest the restaurant to have no more than 20 robots for best performance. - Practically, taking into account the size of the restaurant, a team of 10 robots would be sufficient.
Is the system scalable and flexible?	- The system can easily be scaled to account for larger maps, more robots and obstacles as well as multiple entry/delivery points since the elementary instance of the problem is trying to solve a routing problem with single source / target combinations that is then extended to multiple sources / destinations through re-running the algorithm after each new event with the updated parameters.
What are the strengths of the system?	- PERR setup tends to be easier to solve than the corresponding MAPF instances - All PERR instances are solvable. - Pathfinding algorithm performs well with Two-Loop setup (*Figure 14*) compared to the state-of-the-art MAPF solver CBS. - Algorithms produce optimal solution for pathfinding problem. - Optimal Flow-based ILP algorithm can be replaced by a suboptimal version if pathfinding runtime becomes a bottleneck, reducing 50% the runtime in most cases.
What are the limitations of the system?	- The system has to rerun pathfinding algorithm for all agents upon new events. Worst-case scenario happens when the event incoming frequency is high, resulting in the need to recalculate paths at every time step - Centralized system creates a single point of failure

VI. Future work

Below is some directions and ideas for follow up works to improve the system.

- Learning from real-life data and re-running the simulation to optimize for the number of robots needed (either ask for more or less robots)
- Evaluate for other generic restaurant plans where new constraints might arise
- Designating unique routes for tables that are always occupied
    - If the customers prefer a certain table so that it is always fully occupied, then it might mean that it would be more efficient to have its own route

D. Conclusion

Artificial Intelligence has been regarded as a double-edged sword considering it has the power to greatly aid yet severely hurt people and society. Before implementing and deploying a system, it is imperative to consider the advantages and the disadvantages that the system may bring about. Implementing a planning and navigation system for robot waiters will increase restaurant productivity. The utilization of robots will be chiefly beneficial to restaurant owners who employ them. The robots speed, reliability, and consistency makes them the ideal employee. Robots do not need a bathroom nor a lunch break and they can carry significantly more weight. Robots are programmed to minimize errors thus making them more precise in their decisions. Additionally, every action a robot takes can be recorded and analyzed for continuous learning and improvement. This gives robot programmers immediate feedback thus leading to technological advancements in that industry. Restaurant owners can save money on labor costs and customers can save tip money. In fact, restaurant owners will save so much money on labor costs that they will be more cost effective than human waiters in the long run [5]. The robots will be able to serve more customers which increases revenue for the restaurant. Lastly, a restaurant with robot servers will definitely be a unique attraction which will attract more customers.

On the other hand, there are many aspects of a human that robots cannot replicate. Robots cannot replace the human interaction people have with their waiters, who have unique personalities. A major aspect of going out to eat is the human connection and the atmosphere within the restaurant. Robots are also not as nimble as humans because they rely on sensors to avoid collisions. Humans are

able to avoid collisions on their own and are much more mobile. While it would happen slowly, robots would replace vital jobs which would increase the unemployment rate. Many waiters rely on their jobs and for human kindness through tips for income. Additionally, robots cannot process customer complaints or requests. Unless they are specifically programmed to, they cannot handle unexpected situations. If a disaster were to strike within the restaurant, such as a medical emergency, an atmospheric catastrophe, a terrorist attack or worse, the robots would not be able to save human lives like a waiter might be able to.

Overall, perhaps automating robots for tedious human processes is the natural progression of the restaurant industry but it could also be detrimental to those who rely on waiting tables to provide for themselves and their families. Fortunately, humans are considered more efficient than robots in many domains such as decision making, handling difficult situations, and bringing a sense of empathy into a workplace [5]. Despite this, robots may prove to be a feasible option for the foodservice industry once their technology and competence improves. In conclusion, we believe that our system efficiently implements a task assignment, path planning and navigating system for robot waiters. The restaurant is a major attraction of Easton, PA and customers of the restaurant will be able to have an enjoyable, entertaining and a productive visit to the top-rated ramen restaurant.

Team Members

- Agathe Benichou
- Wassim Gharbi
- Greg Shindel
- Nick Turney
- Thanh Vu

References

[1] Hang Ma , Craig Tovey , Guni Sharon , T. K. Satish Kumar , Sven Koenig. *Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem*. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona

[2] Hang Ma, Sven Koenig, Nora Ayanian, Liron Cohen, Wolfgang Honig, T. K. Satish Kumar, Tansel Uras, Hong Xu, Craig A. Tovey, and Guni Sharon. *Overview: Generalizations of multi-agent path finding to real-world scenarios*. In IJCAI-16 Workshop on Multi-Agent Path Finding, 2016.

[3] Hang Ma , Jiaoyang Li , T.K. Satish Kumar , and Sven Koenig. *Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks*. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, May 08-12, 2017, São Paulo, Brazil

[4] Joydeep Biswas and Manuela Veloso. *Episodic non-Markov localization: Reasoning about short-term and long-term features.* In Proceedings of International Conference on Robotics and Automation (ICRA), 2014, pp. 3969–3974.

[5] Winn and Patrick. *The Rise Of The Robo-Waiter.* NPR, 5 May 2011.

[6] Felner, Ariel, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan R. Sturtevant, Glenn Wagner and Pavel Surynek. *Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges.* In The International Symposium on Combinatorial Search (SoCS), 2017, Pittsburgh, Pennsylvania

[7] Yefim Dinitz (2006). *Dinitz' Algorithm: The Original Version and Even's Version*. In Oded Goldreich; Arnold L. Rosenberg; Alan L. Selman. Theoretical Computer Science: Essays in Memory of Shimon Even. Springer. pp. 218–240. ISBN 978-3-540-32880-3.

[8] Ford, L. R. and Fulkerson, D. R. (1956). *Maximal flow through a network.* Canadian Journal of Mathematics. 8: 399–404. doi:10.4153/CJM-1956-045-5

[9] Greg Fodero, Ashleigh Swingler, and Ferrari Silvia. *A model-based cell decomposition approach to online pursuit-evasion path planning and the video game Ms. Pac-Man*. In IEEE Conference on Computational Intelligence and Games, 11-14 Sept. 2012, Granada, Spain.

[10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a Modern Approach*. Pearson, 2016.

[11] Guni Sharon, Roni Stern, Ariel Felner, and Nathan Sturtevant. *Conflict-Based Search for Optimal Multi-Agent Pathfinding*. *Artificial Intelligence*, vol. 219, 2015, pp. 40–66., doi:10.1016/j.artint.2014.11.006.