Game-Theory Based Task Allocation

Jason Chan, Justin Chan, Yuxi Yang

I. Introduction

Overview: Over the years, jobs that normally are done by people are slowly starting to become automated. This started from small household tasks such as vacuum cleaning and washing clothes to jobs in industry, such as assembly line processes in creating products for a business. In recent times, self-driving cars have been invented, and businesses such as Dominoes have been partnering with robotics company Nuro have been testing them for delivery jobs [1]. However, if we are to achieve full automation, for delivery services, delivery allocation and acceptance must also be automated. Currently, delivery services such as Uber Eats and DoorDash allows people that work for them to choose which job to deliver. They are given a destination and delivery fee, the money they can make, and they can decide whether the money is worth the effort to deliver. Delivery orders are sent to a single driver and wait for them to accept. If the order is not accepted, or fifteen seconds pass, the order is immediately passed to a different driver nearby and waits to see if it gets accepted. Similarly, our goal is for self-driving cars to be able to accept delivery options based on factors such as revenue from delivery fees and the costs of driving. For UPS delivery, their system has a daily task allocation of package delivery based on destination (habitual). However, the delivery occurs only once a day to allow a stock up of orders, which stalls the time packages are delivered. In this paper, we want to figure a system that balances both time of delivery and the profits of using self-driving cars. We will consider the scenario of long lists of orders that must be delivered as soon as possible given a set of self-driving cars which we will call them "robots". We want to use a game theory inspired system to efficiently allocate orders to robots that allows for a better delivery service.

Motivation: Game theory algorithms have been widely used for resource allocation and decision making within a system. Especially when considering problems that are related to resource and task allocation where the agents are generally selfish. This approach allows for more optimal resource allocations that guarantee some degree of fairness amongst users, as well as reduction in resource fragmentation, which in turn increases the efficiency. When looking at algorithms that apply game theory, market-based negotiations in the form of auctions play a huge role. The advantages of a market-based resource allocation mechanism includes scalability of the negotiation system, as well as the flexibility to deal with uncertainty in a real-time environment. This means that the system can not only handle problems of varying sizes, but is also faster than heuristic approaches that have a heavier computational requirement. For our approach, we have chosen to compare the effectiveness of a sequential single-item auction(SSI), a dutch auction, and a combinatorial auction.

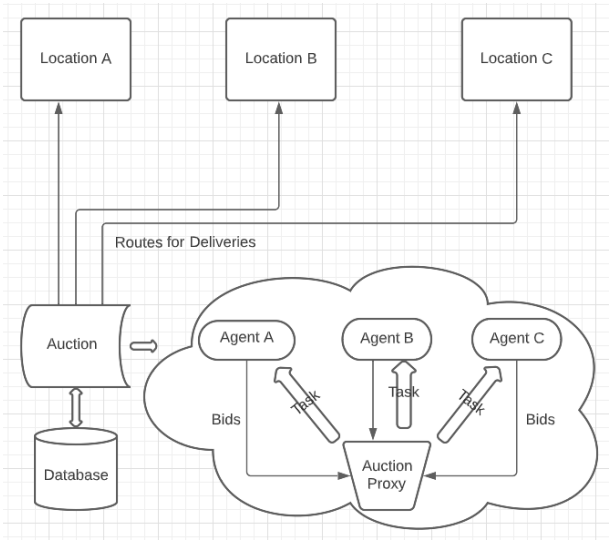

Fig. 1: Task Allocation Framework for Multi-agent Delivery System

I

In an SSI auction, there will be sequential rounds of auctions for a single item at a time. In the case for resource and task allocation, agents can bid for resources and individual tasks that are being auctioned. In a dutch auction, a single item is also auctioned off every round. However, the auction will display an "asking price" that will decrease every auction round until one of the bidders accepts the asking price. In a combinatorial auction, bidders can bid for multiple items at a time in the form of a "package". The SSI auction guarantees constant-factor performances for agent travel distances, regardless of the use of approximations. Compared to combinatorial auctions, SSI auctions require less computing resources, but have poorer and less efficient task assignments. The Dutch auction on the other hand, avoids bidding wars and is more transparent.

Due to the fact our model is based around a delivery system, route planning is an important part of our model. When establishing the model for bidding rules, distance is an important factor to consider. Thus, we shall first implement a simple euclidean distance estimation of the distance, and then develop a route planning algorithm to decide on most efficient route given a stochastic map. We can then evaluate the effectiveness of our route planning algorithm by comparing it to the estimation method. Furthermore, in order to help with this we will create a simple virtual environment in order to run simulations of our algorithm.

As our model is game-theory auction based, we will have to establish bidding rules to allow the agents to send participate in the auction for task allocation. As the nature of the tasks in our model is delivery based, the bid will be comprised of weighed factors to evaluate the most profitable route. Example factors to consider are battery efficiency, route for charging ports, cost of trip, time and distance to the destination, and revenue from delivery. Also, depending on the auction model, the tasks may be allocated differently which would affect the factors of the bidding model. Thus we will have to construct various bidding models to conform to the auction type.

We also wish to compare the efficiency of the auctions given a set of totally selfish agents, and another set of agents that balance personal profits and team profits. Thus we will compare the use of two computation algorithms that are based on a non-cooperative game, and a cooperative game respectively. As the cooperative game will have team profits to consider, team goals will have to be established that will also affect that bidding models.

On top of the auction process, we will look at the wireless communication aspect between the server and agents. Delivery services will have a server that contains information on delivery tasks and will be required to send the information to the agents. The reliability and time taken to send the data is important to consider when engineering the auction system and its communication protocols.

Contributions: In this paper, we make the following contributions:

- We compare the work efficiency between using a sequential single-item auction(SSI), a dutch auction, and a combinatorial auctions model.
- We propose a method to establish bidding rules in order to evaluate the most profitable route, by building a model related to various factors such as: battery efficiency, route for charging ports, cost of trip, time and distance to the destination, and revenue from delivery.
- Based on the bidding system, we compare the use of two computation algorithms. One is non cooperative game where agents are selfishly focusing more on personal gains and individual profits, and another one is a cooperative game that balances self benefits of agents and team profits.
- We look in the wireless communication aspect of the system and compare different communication protocols. From the analysis, we will engineer the optimal simulation for the different auction models and compare the efficiency of communication.

II. Related Works

Mosteo states that not much analysis between different auction types. He compares single item auctions (SSI) and parallel auctions (while running simple summation of costs as the bidding computation) and determines that SSI is more efficient. He goes further and adds a time delay between each auction and discovered that task allocation was more efficient than without it [2]. This was a big motivation to focus on sequential single-item auction. Then, we delve a bit into how to improve single-item auctions by increasing the performance of sequential single-item auctions by increasing similarity to combinatorial auctions without resulting in a significant increase in communication and computational requirements. This can be accomplished with the use of hillclimbing and rollouts [3]. Another paper discusses a variation of the combinatorial auction, specifically a combinatorial auction with item bidding, where agents participate in multiple single-item auctions simultaneously. They analyze the impact on equilibrium performance depending on the type of single-item auction implemented [4]. In a different paper, Schneider shows that sequential single-item auctions actually massively outperform other single auction types in static scenarios where agents start clustered together. However, once target points start to be dynamically allocated with distributed agents, they noticed that the performance takes a large hit [5].

Besides single-item auctions, we also take a look at combinatorial auctions. As mentioned earlier, combinatorial auctions have very efficient task allocations, but requires heavy computations to achieve a good result. Thus, Tennenholtz proposes the use of B-matching techniques that are especially good when there are linear goods and binary bundles [6]. Lin et al. saw that combinatorial auctions had drawbacks but appreciated combinatorial bidding. He worked cooperation mechanism that also allows agents to group with each other to bid for tasks, allowing for a compromise that benefits everyone instead of just one. [7] In another paper, Lin's auction system is further improved on this basis. This part of the optimization takes into account the contingency of an increase or decrease in the number of robots, allowing the route planning scheme to follow the actual situation [8]. Then let us focused on problems need to be solved when packing the task set. Elango firstly integrate and classify all tasks into multiple sub-tasks by clustering techniques with K-functions and then the robots will bid on them. Similar to our idea, he also takes into account the travel cost and distance of the journey in the calculation [9].

Other papers discussing task allocation also have to take into consideration costs or utility (which is the inverse of the cost). For example, Wei tries different ways to calculate utility in a cooperative auction system an allocation matrix to allocate all resources to different agents and recursing with functions that find efficiency loss of a reallocation within the matrix until no other efficiency loss can be found, ensuring a best possible scenario [10]. In a different method, Lagoudakis tests with different bidding strategies, specific with distance based tasks that takes into account paths and trees (paths of individual robots vs a representation of allocation of multiple tasks). They discussed different potential bidding rules within a team environment and tested them with three different bidding computations. Their mathematical calculations of costs for paths was an inspiration for our different testings of bidding [11]. Roldan also compares competitive and cooperative drones in an environment with drones working to create a map of a big area. Two different algorithms (one for competitive and the

other for cooperative) were made to compare the efficiency of the allocation of areas for the drones, and then the efficiency of creating the map overall [12].

As we are taking distance into consideration for the computation of the bidding, we also have to implement route planning for the delivery system. In their paper, Yan proposes the use of a two stage architecture in order to find optimum coordinated routes and optimum combination of the various objective functions. This method of route planning would likely be effective for the implementation of combinatorial auctions and team objectives [13]. From a route planning point of view, a large amount of computation and and storage is inevitable. To solve the problem, Dang proposes a cloud computing platform on targeted services for robots in order to facilitate real-time interactive functions for path computation and comparison [14]. In addition to the static influences that can be taken into account when performing route computation, the identification of dynamic obstacles is also important. Yin proposes an adaptive weighted fusion algorithm for sensors, which effectively solves the problem of one-sided data from a single sensor and provides more comprehensive information for route planning [15]. When the route planning is completed, the robot needs to obtain its activity route through the auction system. Hwang derives a cooperative auction system based on a single auction, which triggers a secondary auction system to split and redistribute the path when a robot's work task is significantly higher than the average or when the previous route planning cannot be ideally implemented , which is very similar to our idea of path planning. [16].

Inspiration for wireless communication simulation came from Weigle's simulation of traffic generator. The system takes in a packet header trace from a network link. The trace is read to see the characterization of different TCP protocol connections. Then a ns2 module is used to read the socket-level behavior. Combined with a simulation of network path, traffic is generated and recorded [17]. The different protocols we use was inspired by the paper "Performance comparison between TCP and UDP protocols in different simulation scenarios." The paper explains the different characteristics of TCP and UDP protocols and explores how changing things like bandwidth or packet sizes affect the running of communication with both protocols. Metrics measured included average throughput, packet loss, and end-to-end delay. The paper discusses different scenarios where one protocol might be better than the other [18].

III. Algorithms

Path Finding Algorithms (Euclidean Distance and A*): In our paper, we are implementing stochastic maps with various task locations into our simulations to better observe results. Thus it is imperative that we have some sort of route planning Algorithm in order to compute the distances needed to be traveled from the base station to task destination. We first start off with a very simple estimation using the Euclidean distance to find the direct distance between the two points. The euclidean distance formula can be taken as

$$d(p,q) := \sqrt[2]{(q_1 - p_1)^2 + (q_2 - p_2)^2} \tag{1}$$

This is a good approximation but unfortunately does not take into account any routes or different paths. As the focus of our paper is on a delivery based system, inherently routes and path finding play an important part in maintaining the integrity of our model. Thus, we will also implement a simple route planning algorithm dubbed the A* Algorithm. The reason why we chose the A* algorithm as our heuristic as it is one of the most simple and efficient methods in finding the shortest path. The most important equation for this algorithm can be represented as follows:

$$F = G + H \tag{2}$$

In this algorithm, the various points of the map we observe are referred to as nodes. G represents the distance between the current node and the start node, while H is the heuristic, or estimated distance from the current node to the end node. Interestingly enough, we can implement our earlier euclidean distance formula as our heuristic in this algorithm. Thus, we have

$$H = d(p,q) := \sqrt[2]{(q_1 - p_1)^2 + (q_2 - p_2)^2} \tag{3}$$

F represents the total cost of the node, and is thus equivalent to $G + H$. As can be seen in the pseudo code for the A* algorithm described in *Algorithm 1*, we use loops to re-iteratively branch out and find the most optimal node until we reach the destination.

The following sections will discuss different bidding strategies and auction styles. Our goal is to implement them, find different combinations of bid and auction, and compare how they increase in time efficiency of the task allocation as well as the efficiency of the allocation itself. We want to create a system with task allocation that kind find the most efficient allocation as fast as we can. The time it takes to delivery and the amount of fuel (battery) used for the task are the factors we will look into.

Selfish bidding First, we must talk about the properties of the robots. As this is a delivery system simulation, each robot represents a car designed for delivery. Each robot is a class of entities that will perform the tasks. Each one has different characteristics: battery consumption rate (BCR), battery capacity (B_c), average velocity (V_a) when traveling, and cost to fully recharge battery from depletion (c). This is important because for this paper, we give each robot uniqueness: Robot one (R_1) for control, Robot two (R_2) that has a preference to fuel efficiency, and one more (R_3) with a preference for time. The preferences are indicated by adjusting BCR (battery consumption/distance) and Va; R_2 has a low BCR but low V_a which means it prefers longer distance, while R_3 has high V_a but also high BCR, which means it prefers shorter distance. Each robot will have a an optimal, or preferred, distance d_p that they want to take, and bound variables $bound_c$ and $bound_t$ representing closeness for preference for cost and time (this

Algorithm 1 A* Algorithm

```
openL := {start}
closedL := ∅
while openL ≠ ∅ do
    current = leastF()
    openL := openL \ {current}
    closedL := closedL ∪ current
    if current == goal then
        finalp = currentp
    end if
    current.children := current.adj
    for c ∈ children do
        if c ∈ closedL then
            gotostartoffforloop
        end if
        c.g = current.g + D(c, current)
        c.h = D(c, end)
        c.f = c.g + c.h
        if c ∈ openL then
            if c.g > openL.g then
                gotostartoffforloop
            end if
        end if
        openL := openL ∪ {c}
    end for
end while
```

will be relevant for co-op bids mentioned later)

Additionally, all robots have individual task queue T_R and a distance D_R required to travel based on its task queue. This, which will be explained later in depth, is for auction styles where a robot can bid for more tasks after initially having one already.

When an auction requires a bid from the robots, the system will give details of task t (the location of delivery). The robot must first decide if it is possible to perform the task, especially if it already has tasks in the task queue. The battery consumption is multiplied by the sum of the distance required for the task up for bid and the distance required to complete all tasks in its task queue to get what we refer to in the pseudo-code as B_r

$$B_{possible} = (B_c >= BCR * (D_R + d)) \tag{4}$$

Factors that are given as input are fees for delivery (F), the distance required (d) and cost for fully recharging a batter (c). The distance required will vary; the shortest and simplest would be the distance to the delivery point and back to base, but this can change to adjust for algorithms mentioned later such as hill climbing.

In this paper we will compare two different bidding objectives: selfish and cooperative. Selfish bids are simple computations on purely revenue. The computation is as follows:

$$BidSelfish(task, distance) = task.F - BCR * distance * c \tag{5}$$

Figure 2 shows a simple diagram of this bidding algorithm. The bid is directly calculated by the total revenue using input of the task and distance. The winner of the auction would be the robot with the highest profit.

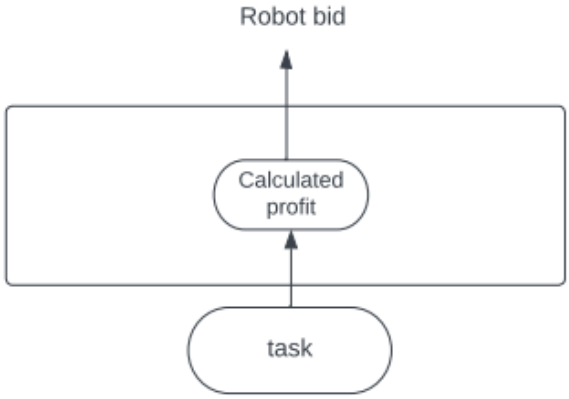

Fig. 2: Selfish Bid Process

Cooperative Bidding What about in a cooperative objective? In a system when all robots are owned by the same person, the revenue of the individual robots doesn't matter anymore because the combined revenue of all tasks will be the same no matter how they are assigned/ In this case, revenue is no longer part of the bid, but rather the time of delivery as well as costs of the trip. In this case we need to adjust the bidding computation. Figure 3 shows the whole algorithm to create the cooperative bid. Assuming a 50/50 split in priority between time and costs, we can set bids to be out of 20, max 10 points for each category. Time can be calculated by velocity * distance, and cost is calculated the same in selfish bid: BCR*d*c. The biggest difference here is now the robots have a preference of distance. Computation for cooperative bidding is adjusted as followed:

$$C_{bid} = 10 * sinc(n_t/b - n_p) \tag{6}$$

n_t stands for the number required for the given task, b represents the bound of preference, or the range, and n_p represents the preferred number. We chose sinc as it allows for a single point highest point of preference, and the closer the number is to the preferred value, the higher the value would be.

Now we can give the computation for cooperative bidding. Algorithm *Algorithm 2* shows the full algorithm of the bid. Given velocity and costs are calculated as

$$time = distance * velocity \tag{7}$$

$$Cost = BCR * d * c \tag{8}$$

$$B_{cooperative} = 10 * sinc(t_t/t_b - t_p) + 10 * sinc(c_t/c_b - c_p) \tag{9}$$

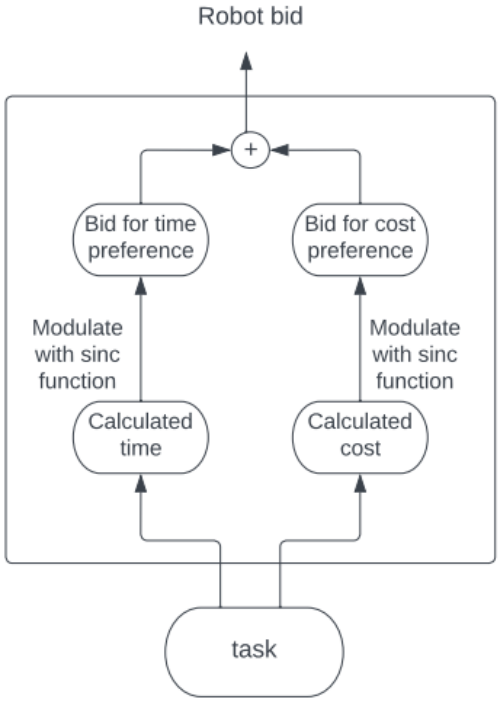

Fig. 3: Cooperative Bidding Process

Algorithm 2 BidCoop(r, task, d_{task})

$Time_{actual} = r.Va*(r.distance + d_{task})$
$Time_{prefer} = r.Va*(r.distance + r.d_p)$
$Bid_{time} = 10*sinc(Time_{actual}/r.bound_t\text{-}Time_{prefer})$
$Cost_{actual} = r.BCR*(r.distance + d_{task})*r.c$
$Cost_{prefer} = r.BCR*(r.distance + r.d_p)*r.c$
$Bid_{cost} = 10*sinc(Cost_{actual}/r.bound_c\text{-}Cost_{prefer})$
return $Bid_{time} + Bid_{cost}$

The subscripts t, b and p stand similar to the ones in C_{bid}: necessary for task, bound and preferred respectively. T represents time while c represents costs and are computed as shown. The preferences are calculated based on the preset preferred distance d_p.

A formal bid function is written out as pseudocode in Algorithm *Algorithm 3*. It contains the check that the task is possible before calling one of the bid functions. To swap between selfish and co-op bid, there is an additional mode input with the bid function, such that 0 represents selfish bid while cooperative bid is represented by 1 (or as the pseudocode is written, it could be any other number).

SSI Auction: One of the core auction algorithms we will discuss in this paper is the Sequential Single-item Auction. The SSI Auction for short, is essentially a series of single-item auctions. The following auction works as follows: All tasks start off unallocated, and the tasks are auctioned off in a multiple round type fashion. During each round of the auction, each agent places a bid that is formulated off a set of bidding

Algorithm 3 bid(r, t, mode)

$d_{task} = A*(t)$
$B_r = BCR * (d_{task} + r.distance)$
if $B_r < r.B_c$ **then**
  if mode $== 0$ **then**
    return BidSelfish(r, t, $d_{task} + r.distance$)
  else
    return BidCoop(r, t, d_{task})
  end if
else
  return 0
end if

rules. The agent that places the highest bid is then assigned the corresponding task. This cycle repeats until all tasks are auctioned off. If we take a look at *Figure 4*, we can see that each round there will be a task allocated, and the winning robot denoted wr will have the task added to its task queue. The importance of this task queue will be explained later on when we implement hill-climbing into the algorithm. However, for a very simple SSI auction, its structured as followed: we let R be the set of robots/agents and T be the set of tasks that have not been allocated, where the set initially contains all tasks. We can then let $T(r)$ be the set of tasks assigned to robot $r \in R$, where $T(r)$ is initialized as an empty set. In each round of the auction each robot r can submit a bid $r.\text{bid}(t)$ for the task. The bidding function is the selfish bidding function we described earlier in the paper.

$$r.bid(t) := F - BCR * d * c \tag{10}$$

We also let wr denote the winning robot of the auction round, and wt the corresponding task that was assigned. After the task has been assigned, we update $T(wr)$ and set T by removing the item from T and adding it to $T(wr)$.

$$T := T \setminus \{wt\} \tag{11}$$

$$T(wr) := T(wr) \cup (wt) \tag{12}$$

This process will then repeat until all the tasks have been allocated to the robots where $T := \varnothing$, as can be seen in the pseudo-code in *Algorithm 4* This SSI-auction can be considered our base or default model in which we will compare the simulation results of other algorithms to.

As mentioned earlier in the paper, compared to combinatorial auctions, SSI auctions require less computing resources, but have poorer and less efficient task assignments. Thus our goal is to improve the SSI auction by increasing the similarity to that of a combinatorial auction through the implementation of hill-climbing. Hill-climbing is an iterative algorithm that starts with an initial solution to a problem, and then attempts improve the solution by making an incremental change to the solution. Essentially what happens in our algorithm, is that once each robot has been assigned at least one task, we then compare the destination of the task being auctioned to all the tasks previously assigned. The robot or agent that would have

Algorithm 4 SSI Auction

for each robot $r \in R$ **do**
  $T(r) := \varnothing$
end for
while $(T \neq \varnothing)$ **do**
  for each task $t \in T$ **do**
    for each robot $r \in R$ **do**
      submit(r, t,bid(r, t))
    end for
    let wr = winning robot and wt = won task
    $T(wr) := T(wr) \cup (wt)$
    $T := T \setminus \{wt\}$
  end for
end while

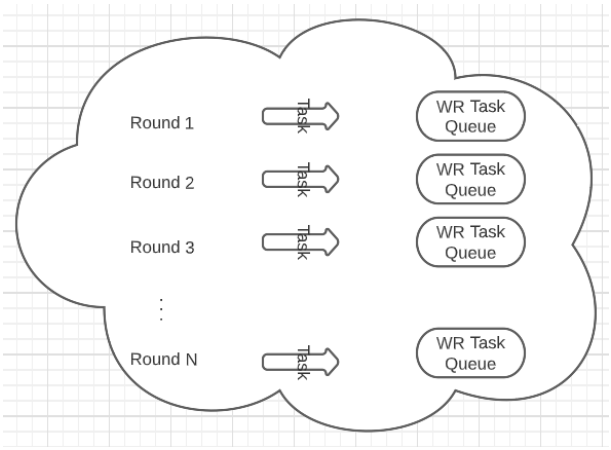

Fig. 4: Single Sequential Item Auction

to travel the least from its previous task will have a higher priority when submitting the bid. In the case of our paper, it is first important to note that we need to establish a team objective or team cost $tc(T_1, ..., T_N)$. The team cost is the aggregate cost of distance required to travel for all agents, given the tasks assigned. Thus, in order to maximize efficiency, we wish to minimize the overall team cost. We implement this in our SSI auction algorithm by first waiting until each robot $T(r)$ is non empty. The auction functions like a normal SSI auction with the same bidding equation in *equation 10*. Once each robot is assigned some set of tasks $T(r_i)$, during each round of the auction we compare the required travel distance of task t to each task t_j in $T(r_i)$. Given Total Team cost is

$$tc(T_1, ..., T_N) := \Sigma_i D(r_i, T_i) \tag{13}$$

where $\Sigma_i D(r_i, T_i)$ is the sum of Distances that need to be traveled to complete each task. Given r_i is assigned a task t where

$$t \notin T_1 \ \cup, ..., \cup \ T_N \tag{14}$$

We can then derive tc_{in} which represents the increase in team cost if the r_i were to accept the task, and submit it as an additional parameter to the bid equation.

Algorithm 5 SSI Auction With Hill-Climbing

for each robot $r \in R$ **do**
  $T(r) := \varnothing$
end for
while $(T \neq \varnothing)$ **do**
  for each task $t \in T$ **do**
    if any $T(r_i) := \varnothing$ **then**
      for each robot $r \in R$ **do**
        r.bid(t):= $F - BCR * D * c$
        submit(r, t,bid(r, t))
      end for
    else
      for each robot $r \in R$ **do**
        $tc_{in} := D(r, T \cup t) - D(r, T)$
        submit(r, t,r.bid(t, tc_{in},0))
      end for
    end if
    let wr = winning robot and wt = won task
    $T(wr) := T(wr) \cup (wt)$
    $tc(wr) := D(wr, T(wr) \cup t)$
    $T := T \setminus \{wt\}$
  end for
end while

$$tc_{in} := D(r_i, T_i \cup t) - D(r_i, T_i) \tag{15}$$

$$bid(t, tc_{in}) := F - BCR * tc_{in} * c \tag{16}$$

We then can then find which robots would occur the lowest increase in team cost which would affect the submitted bid accordingly. The robot that ends up winning the task during that auction round (wr) will also update the team cost of its task queue $T(wr)$ like so.

$$tc(wr) := D(wr, T(wr) \cup t) \tag{17}$$

As can be seen in *Algorithm 5*, the auction algorithm will then continue to run until all tasks in set T have been allocated, and the agents are ready to depart to carry out their task assignments in their corresponding task queue. Compared to the simpler SSI auction method, because hill climbing attempts at reducing overall cost by finding more efficient assignments, the overall task allocation algorithm will become more efficient. Taking into account the metrics we will observe from the simulation results, the SSI auction that implements hill-cimbing should have excel in all metrics compared to the default SSI auction. Due to the focus on minimizing the total distance traveled as a team cost, it also intuitively reduces the overall cost of travel, time for deliveries, and battery consumption.

Combinatorial Auction: Combinatorial auction is another main auction algorithm we use in this paper. It is an intelligent auction system that essentially packages multiple independent tasks to be auctioned. The priority determinant is the efficiency improvement of the overall team. In this auction process, tasks

are grouped together in the form of "packages" that allow for more efficient task completion. These packages are then auctioned off to agents in a multi-round auction. This can be clearly seen if we take a look at *Figure 5*.

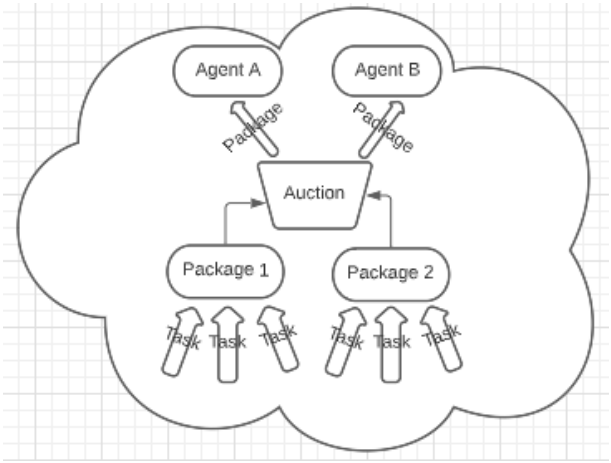

Fig. 5: Combinatorial Auction

The packaging of task needs to follow the principle of finding the shortest route, which means tasks that have nearby destinations will be grouped together as a package. This allows robots to efficiently complete tasks in a general area. Assuming there is an arbitrary number of unallocated tasks $t_i$in set T, we want to group up the tasks in such a way that package p from a set of packages P is assigned to some robot is equivalent to the sum of tasks:

$$P(r_i) = p := \Sigma_i t_i$$

We then dictate how the tasks are grouped together, by applying the K-means clustering algorithm. The K-means algorithm is a non deterministic algorithm for partitioning observations into various clusters. This algorithm is non-deterministic due to the fact that cluster assignments will be different if you run the algorithm on the same set of points twice. This can be seen by the random initialization of centroids in the pseudo-code for *Algorithm 6*. For the sake of this paper, we can set the number of clusters k equal to the number of robots, and initialize set C with centroids $c_1, ..., c_k$. Each robot is assigned a cluster or "package" of tasks to carry out. Given there is a set L that contains the locations of all the tasks in T, we can then find the location of all other tasks by the simple equation:

$$L_{others} := L - C$$

Then, for each task location $l \in L_{others}$, we assign it to the closest cluster $c_i \in C$. In order to identify the closest cluster, we compare the distance of location l to each cluster $c_i \in C$ and find the l that minimizes the distance. We can then update the package containing that cluster with the following equation,

$$P(min_c(l)) := P(min_c(l)) \cup l$$

where $min_c(l)$ represents the cluster in which the task with location l is nearest to. In order to maximize the effectiveness of this algorithm, we will reiterate the process of clustering until we achieve the optimal grouping. Thus, after we have established the clusters in each iteration, we set new centroids to the median of the package.

$$newc_i \in setC = median(P)$$

We then continue reiterating the algorithm until the centroids no longer change like so.

$$C_{last_i} = C_{current_i}$$

Once we have the K-means algorithm to take care of the packaging of tasks with the original set of tasks T and k clusters as inputs, the rest of the combinatorial auction algorithm becomes relatively simple. Taking a look at the pseudo-code for the combinatorial auction in *Algorithm 7*, it utilizes the K-mean *Algorithm 6* to construct the packages. After this process, it is can be seen to be very similar to an SSI- auction with packages being auctioned off each round.

Algorithm 6 K-means Cluster

Given k number of clusters
Randomly initialize k Centroids $c_1, ..., c_k \in$ set C
Repeat
Given $L_{others} := L - C$
for each $l \in L_{others}$ **do**
  for each $c_i \in C$ **do**
    if $dist(l, c_i) < min_c(l)$ **then**
      $min_c(l) := c_i$
    end if
  end for
  $P(min_c(l)) := P(min_c(l)) \cup l$
end for
for each $p \in P$ **do**
  $c_i \in$ set C = $median(p)$
end for
Until $C_{last_i} = C_{current_i}$

Algorithm 7 Combinatorial Auction

for each robot $r \in R$ **do**
  $P(r) := \varnothing$
end for
$Kmean(T, k)$
while $(P \neq \varnothing)$ **do**
  for each task $p \in P$ **do**
    for each robot $r \in R$ **do**
      submit(r, p,bid(r, p))
    end for
    let wr = winning robot and wp = won package
    $P(wr) := p$
    $P := P \setminus \{wp\}$
  end for
end while

Compared to SSI auctions, combinatorial auctions are suitable for cases where the agent presents non-additivity to the task value measure. The goal of this type of auction is multiple tasks being grouped together in the form of packages to be allocated to agents. This type of auction has two significant advantages. First, the combinatorial auction is more effective than the SSI auction in finding the most optimal and efficient solution for task allocation. . Second, the combinatorial auction reduces the number of auctions, thus saving time costs and energy consumption from standby, and reducing the portion of duplicate routes that an agent may take when performing a single task. However, the combinatorial auction comes with a draw back, which is it's much higher computational power requirement.

Dutch Auction: The process of Dutch auction is that the host server puts forward a higher asking price, and then decreases in equal proportion one by one until one agent's bid equals to or more than host server's asking price, then the auction ends. If there are two or more agents giving the same bid, the first agent to bid becomes the winner. To begin the auction, we first set up a start price a. The price goes down by a certain percentage x as time goes. We can assume the host server decreases the price in every s seconds. The agent who bids more than or equal to the current price will win the auction. Therefore, the current price c is $c = a - xs * a$. However, besides the wait time for the asking price to decrease to the point one or more agent's bids are greater than or equal to the asking price, the auction style itself resembles that of a SSI auction.

Algorithm 8 Dutch Auction

```
for each task t ∈ T do
    c = a
    toBeBidded = True
    while toBeBidded do
        for each robot r ∈ R do
            if bid(r, t) >= c then
                r.appendTask(t)
                toBeBidded = False
            end if
        end for
        c = c − cropProportion ∗ c
    end while
end for
```

The Wireless Connection In the communication perspective, an auction is set up by having a number of agents and a server that connects to all agents. We assume for this paper that wifi will be used given its versatility and capability to grow past this paper. In each auction process, the server sends data of the task up for auction as well as the bidding rules. After receiving the information, each robot will use the bidding rules given to calculate its bid and send the resulting number to the server. The ideal auction system would be data containing the task and bidding rules (we will refer to this as *auction data*) would be sent to all agents simultaneously. As soon as each agent receives the auction data, they return the bid number (we will call this *bid data*) simultaneously back to the server. The server should wait until it receives the bid from all agents before starting another auction process for the next task in its queue.

We will explore between two different communication protocols: TCP and UDP. The TCP protocol is connection based and designed to strictly communicate the transaction of the data packets as well as the data itself. Communication must be done between transmitter and receiver to establish a connection, to send the data, to confirm that the data was received, and to finally close the connection. But a key feature of TCP is the ability to adjust send rate of data and resend a package that is lost. This is to acknowledge packet losses and to address them in systems that do not want data to be lost. To do this, Fund says in her experiment that a congestion window is used to control the sending rate based on packets dropped. To describe how this works, Fund exmplains when packets are sent at a rate higher than the bottleneck, the packets are placed in a buffer. As more packets are sent, replaced and added on more into the buffer given the send rate, the buffer will eventually reach full, at which any new packets obtained would be dropped. When the transmitter realizes the packet was dropped (when there is no signal from the receiver that the packet was received), it reduces the congestion window. This repetitive process can be seen in the form of a sawtooth shape, where each drop indicates a reduction in the congestion window. We will refer to this moment as *congestion control*n [19]. The TCP protocol has a lot more weight and complexity compared to the other protocol: UDP. The UDP protocol is connection-less and does not acknowledge packet losses; the transmitter will not resend a packet if it was lost. The trade off with this according to Williams is that the UDP protocol is lightweight as a result of no connections and sends data packets much faster. UDP protocol assumes that the reliability of data sent and data integrity is not important [20]. In this paper we will analyze the TCP and UDP protocols by using the following metrics: end-to-end delay and delivery received ratio. The end-to-end delay is the time it takes for a packet to travel from the transmitter to the receiver. The delivery received ratio is the total number of packets sent over the total number of packets received successfully (i.e. packets that were not dropped). To reiterate the differences given the metrics, the UDP protocol decreases its end-to-end delay at the cost of decreasing the delivery received ratio as well as ignoring moments of packet loss. The latter cost is a big deal because in our ideal auction system the server must wait to receive bids from all agents before sending out the second batch of auction data, but may not receive a bid at all if the auction data is lost from being dropped. But if the data packets can be transferred at a very reliable rate with the UDP protocol, then UDP would be preferred due to its lightweight and low end-to-end delay.

This paper will use Network Simulator 2 (or NS2) to simulate communication. We set up an auction system in NS2 to run for an amount of time. NS2 creates a trace file that

tracks what packages are sent or received and when they happen. Using Linux, code can be used to read the trace file and calculate end-to-end delay by taking the time differences between the packets are sent and received. To calculate the delivered received ratio, the total number of packets sent in the simulation are summed and is divided by the summed number of tracked received packets. In this paper, we will examine the trade offs for TCP and UDP protocols acknowledging the environment to see how to optimally set up an auction system.

IV. Performance Evaluation

A. Python Evaluation Results

For our performance evaluation, we chose to do simulations using Python to simulate the results of the bids and auctions, and NS2 to simulate the communication from the host server and agents during the auction process. For the evaluation of the various bids and auctions, we ran simulations of each auction and compiled data on specific metrics suitable for comparative analysis. For each of these simulations, we performed the same scenario in order to keep as much consistency across the board. Each simulation consists of 100 iterations of the corresponding auction, with a total of 10 tasks being allocated per auction process. In each of the iterations, the tasks are randomly generated before being allocated by the corresponding auction process. There is also a predefined set of 3 robots that have differing parameters to set them apart for the bidding functions. i.e. travel velocity, battery consumption rate, preferred distance, etc. The environment is a simple 100 meter by 100 meter matrix with the base station containing the host server located at location (0,0).

Selfish vs Coop Bid Comparison (Python): For this section of our project, we analyzed the resulting robot task allocation and computational power requirements. Computational complexity can be divided into two separate metrics: computational time and memory usage. As we have set the SSI Auction as the benchmark for out project, we implemented two SSI Auctions with each utilizing one of the two bids. As we take a look at *Figure 6*, we can see the bar graph displaying the task allocation for the various robots in the simulation. It can be clearly seen that the quality of the selfish bid is very poor. In fact, one of the agents is allocated every single task that was to be distributed in the simulation. This is because due to the selfish nature of the bid, it will pursue pure profits. Thus, the robot that has the lowest battery consumption rate, which in turn will maximize the profits, will be able to successfully claim all the tasks in the queue. The cooperative bid on the other hand, can be seen to be a bit better. Although one of the robots is still allocated a fair majority of the tasks, because the bid takes into account other factors like time and efficiency, the other robots are allocated some tasks as well. Normally there are always trade-offs in every case, as no process is perfect. In this case, it can be seen if we look at *Figure 7*, that the average computation time for the cooperative bid is a fair amount higher than the auction with the selfish bid. This is intuitive as the cooperative bid has to analyze a lot more factors before computing a bid. In an example simulation run, the SSI Auction that implemented the cooperative bid had an average computation time of 0.732 ms, while the one with the selfish bid was only 0.311 ms. The cooperative bid method takes more than double the computation time. A similar case can be seen if we take a look at *Figure 8* for the memory usage. The cooperative bid requires a lot more memory than the selfish bid, increasing the computational complexity and required computational power. However, because the selfish bid's task allocation is so terrible, the cooperative bid is simply always better despite the larger computational complexity.

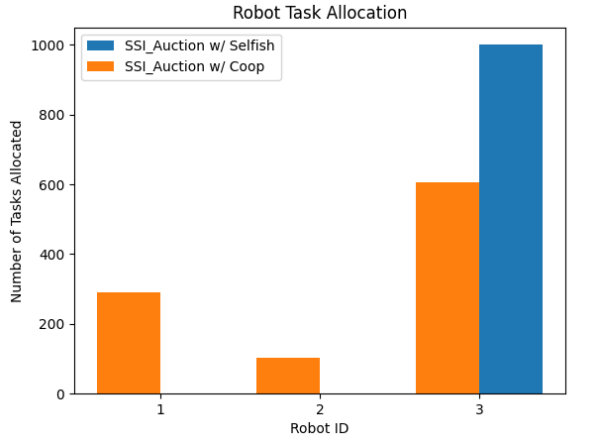

Fig. 6: Task Allocation for Selfish vs. Coop Bids in SSI auction during simulation

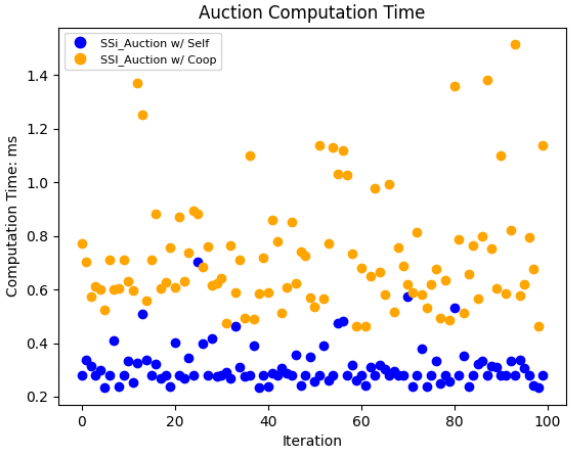

Fig. 7: Computation Time for Selfish vs. Coop Bids in SSI auction during simulation

Auction Comparisons (Python): In our project, we implement 3 different auction types: SSI Auction, Combinatorial Auction, Dutch Auction, and an additional variation of the SSI Auction that implements hillcimbing. In this simulation, each of the auctions implement the coop bid. The various metrics that we compare the auctions with are: Robot Task Allocation, Computational Complexity, Total Required Distance Traveled, and Total Required Travel Time. If we take a look at *Figure*

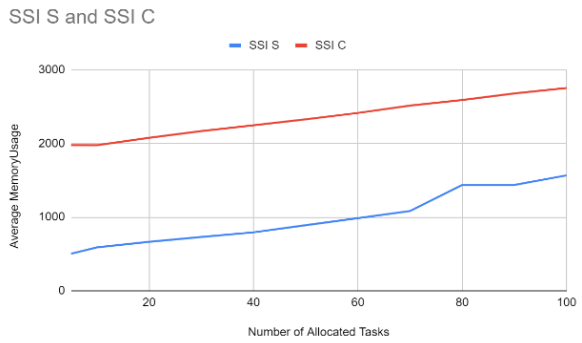

Fig. 8: Memory Usage for Selfish vs. Coop Bids in SSI auction during simulation

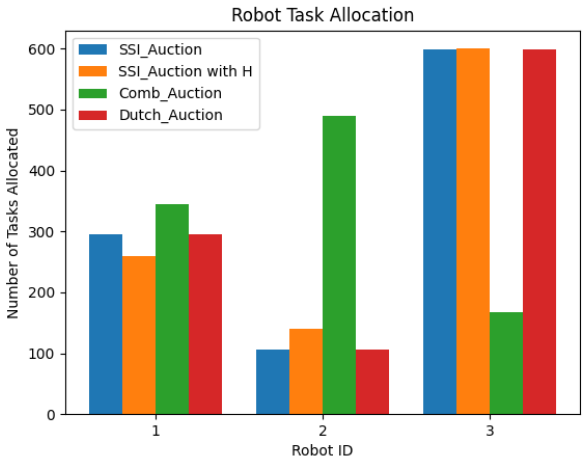

Fig. 9: Task Allocation Comparison between four different auctions during simulation

9 to see the resulting task allocation for the robots, we can see how the distribution of allocated tasks for each robot. Due to the randomness of the tasks that are being generated for each iteration of auctions, each simulations will produce different results. Furthermore, the overall trend of the task allocation after 100 iterations can easily differ if we tinker with the robot's default parameters. An important thing to note however, is that although each iteration of the auctions have a randomly generated queue of ten tasks, the same queue is distributed to each of the auctions. This allows us to observe the task allocation for each auction, given they operate on the same given set of tasks. From the figure, we can see that the SSI auction, SSI auction with hillcimbing, and dutch auction have similar task allocations. In fact, the SSI auction and dutch auction have identical task allocations. This result is consistent due to the way the dutch auction operates. The dutch auction will iteratively decrease its asking price until some participant accepts the price. In this case, the algorithm will merely wait until the asking price decreases to a point lower than the highest bid, before allocating the task to that agent. Due to the fact that all the auctions are using the same cooperative bid for consistency, and the bid is not dynamic to account for the change in the dutch auction's asking price, it eventually functions the same as an SSI auction. The SSI auction with hillclimbing is a different case, as the initial set of allocated tasks utilize the bid function, but the rest of the tasks are allocated with hillcimbing. However, there is still a distinct difference in the task distribution when compared to the SSI auction, and the similarities can be attributed to the parameters of the default robots and the cooperative bid. The combinatorial auction on the other hand has a very different distribution of tasks, which is largely because besides the coop bid it implements, the allocation relies heavily on the k-means algorithm to cluster packages in the most ideal groupings. The importance of this matter will be evident as we take a look at the following metrics.

The next metric we will discuss is the Computational Complexity of each auction. Firstly we can take a look at *Figure 10* for the computation time of each auction. As the Dutch auction functions practically the same as a SSI auction, its natural that they have very similar computation times. However, because the Dutch auction has to wait before the asking price is low enough before the bids of the robots go through, the computation time is slightly higher. The auction that has the next lowest computation time is the Combinatorial auction. If the SSI and Dutch auction have average computation times at around 0.617 ms and 0.635 ms respectively, the combinatorial auction has a much lower computation time of 0.332 ms. This is due to the fact that the auction only has to auction off a small set of packages in each iteration of the auction, rather than each Task individually. This speeds up the auction process itself tremendously. However, the SSI auction that implements hillcimbing actually has the lowest average computation time of around 0.216 ms. The massive decrease in computation time for the auction is due to the hillcimbing process. It allows the auction to skip the bidding process and assign tasks directly to robots that have Tasks with locations already in the general vicinity. The reason the Combinatorial auction's computation time is a bit higher, is because the k-means algorithm takes more time to generate clusters, as it will run iteratively until it generates the ideal groupings for packages. Furthermore, if we take a look at *Figure 11* to analyze the memory usage, its clear that the combinatorial auction uses the most memory. This is intuitive as the auction has higher memory requirements due to the algorithm generating clusters for the Auction process. We also notice that the memory usage also scales linearly with the number of tasks. As the more tasks there are in the task queue, the larger the number and size of packages need to be generated, which will in turn increase the amount of memory required. The memory usage for the SSI auction and Dutch auction have similar behaviors as the number of tasks are increased. Furthermore, due to the similarity in their functions, the memory usage is practically the same. Both of the auctions have an overall lower memory usage requirement than the Combinatorial auction as their auction processes are simpler and easier on the compiler computation wise. The

auction with the lowest memory usage is the SSI auction with hill climbing. This can be attributed to the hill climbing part of the auction process, that allows the auction to skip the bidding process after a set number of tasks have already been allocated. Whats even more interesting is the fact that as the number of tasks increases, the overall average memory usage stays constant. Because the hill climbing part of the algorithm is independent to the total number of tasks, the memory usage doesn't change even if the task queue size is expanded. This is a very good feature as it improves the overall scalability of the algorithm. Overall, the computational complexity of the SSI auction with hillcimbing is clearly better than the other auctions.

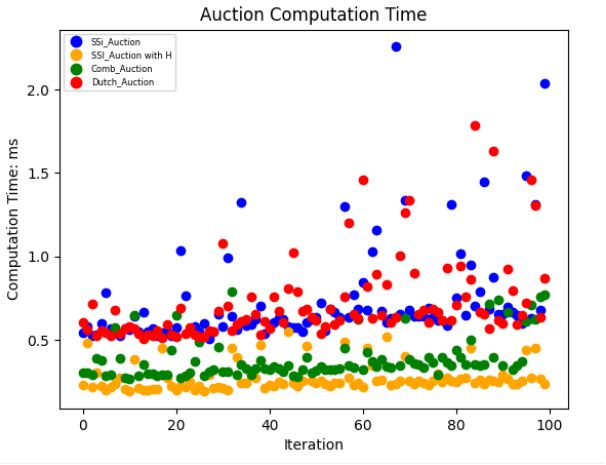

Fig. 10: Computation Time Comparison between four different auctions during simulation

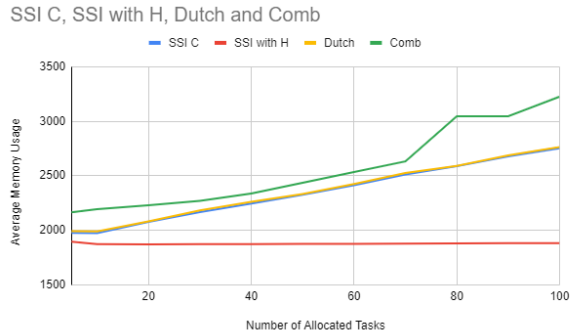

Fig. 11: Auction Memory Usage Comparison between four different auctions during simulation

The next metric we will discuss is the Total aggregate distance traveled for all the agents after they have received their tasks. All the auctions implement the A* path finding algorithm to identify the ideal path after the tasks have been allocated. From this path, (which is a list of nodes in the environment matrix), we can then identify how much distance the robots have to travel. If we take a look at *Figure 12*, the first thing to note is that all the values for the SSI auction are slightly higher than the dutch auction. However, because the SSI auction and dutch auction have the same task allocation, their total distance required are actually the the same. Unfortunately because of this fact, all the values for the SSI auction are covered up by the Dutch auction's values in the graph. Therefore, merely for visibility, we increased the SSI auction values by a notch. Taking the sum of all the data points, we found that the SSI auction and Dutch auction have an aggregate Distance of 38,693 meters over 100 iterations. Essentially, in order to perform the 1000 allocated tasks, the robots together have traveled a total of 38,693 meters. If we take another look at the figure, the combinatorial auction has a similar but slightly lower average distance, with a total sum of 37,800 meters. The SSI auction with hillcimbing however, has an even lower average distance, with a total sum of 32,715 meters. The difference in required distance to be traveled is quite significant, as the larger the distance, the larger the cost in battery consumption, which will result in a decrease in profits. Thus, in the battery consumption department, the SSI auction with hillclimbing is the most ideal with a 13.45 percent lower required distance compared to the next lowest auction.

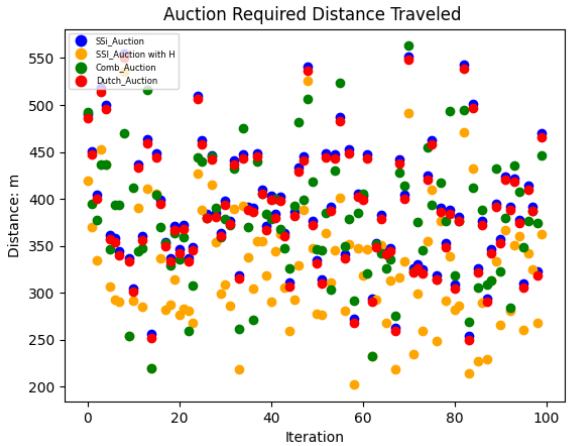

Fig. 12: Required Distance Traveled Comparison between four different auctions during simulation

The final metric to discuss is the Total time required for all the tasks to be delivered. From the A* path finding algorithm mentioned earlier, we were able to calculate the total distance required. However, an important thing to note is that each robot has different parameters, one of them being velocity. Thus, a larger distance value might not necessarily mean a larger time value. If we examine *Figure 13*, we can see that once again the SSI and Dutch auctions have the worst average times. Like mentioned in the previous section, due to the two auctions having the same allocations, the actual values are the same. The only reason there is a small relative increase in the SSI auction values, is to prevent the values from being overlapped and hidden. An interesting to note from the figure is that the SSI auction with hillclimbing has some of the lowest values, but also many values that are as high if not higher than the SSI and Dutch auctions. This

shows that in terms of the metric of time, the SSI auction with hill climbing is actually not very consistent. This is largely due to the fact that the hillclimbing technique focuses solely on reducing and optimizing the overall distance traveled. On the other hand, this metric is where the combinatorial auction out performs the rest, as the values are low and much more consistent. Thus, the average aggregate time for the combinatorial auction is only 9,710 seconds, while the SSI auction with hill climbing requires 12,317 seconds, and the SSI and Dutch auction require 15,376 seconds. In this regard, the combinatorial auction has a 21.17 percent lower delivery time than the SSI auction with hill climbing, and a 36.85 percent lower delivery time than the SSI auction and Dutch auction.

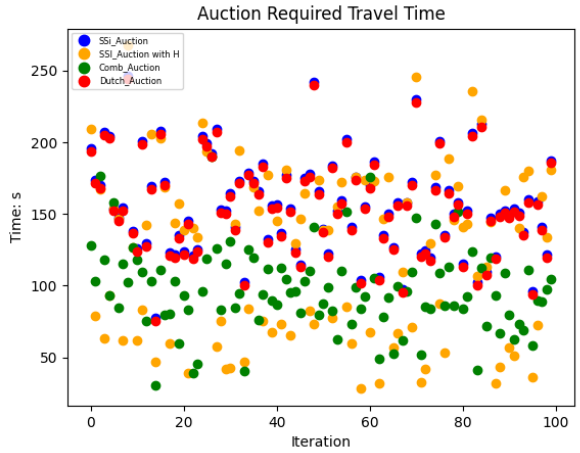

Fig. 13: Required Travel Time Comparison between four different auctions during simulation

B. NS2: TCP vs UDP

The first task to accomplish in ns2 is to determine what protocols are better for the auction and bid. To simulate an auction, three (we limited the number of agents to 3 for this paper) nodes were used to represent the car agents and one node was used to represent the server. Each agent node is connected to the server node representing a data link to communicate with each other by sending data packets. Either the TCP or UDP protocol was used to make the connection. The size of data packets sent can be adjusted to represent different data. In an auction, the server sends data to each agent about the task being auctioned and the bidding rules. A task is given with inputs of coordinates of delivery location (2 integers), the money made for completing the task (1 double) and an id number (1 integer). Therefore each task is assumed to be 20 bytes. Looking at the python file of the bid functions, the cooperative bid will be used for all simulation due to its superiority over selfish bid. The cooperative bid function takes more space than selfish bid as well. Combining the task class, bid function and actual tasks, the total memory size was just under 500 bytes. Thus the system was configured such that the server sends data packets to each robot with the size of 500 bytes. The simulation would run for 20 seconds when comparing protocols.

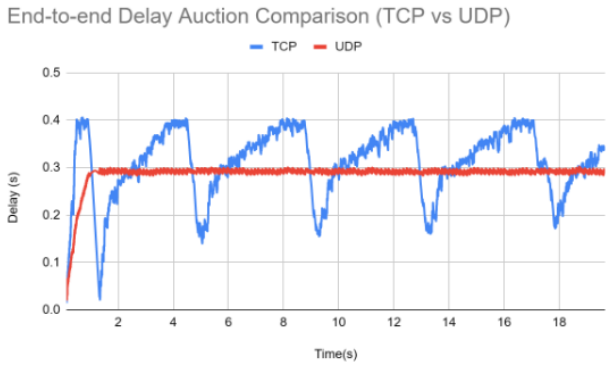

Fig. 14: End-to-End Delay: Comparison for Auction Data. Sawtooth shape and flat line of TCP and UDP can be observed.

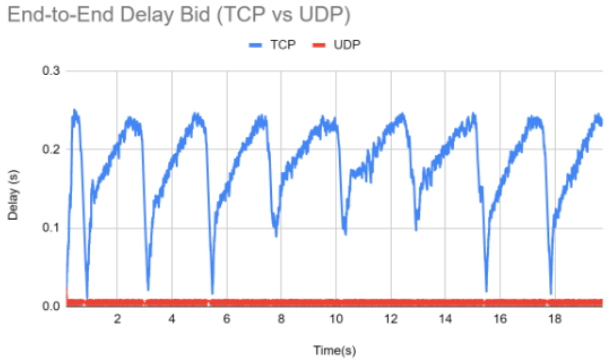

Fig. 15: End-to-End Delay: Comparison for Bid Data. Same shapes, different magnitude. UDP drops to no more than 0.01 seconds

Figure 13 shows the comparison of end-to-end delay between TCP and UDP for auction data. We can observe the sawtooth shape from the TCP protocol mentioned in the Algorithms section. We can see the moments when the congestion bandwidth gets overflowed. The server must take time to resend the packets lost, forcing a stall. This stall is what causes the end-to-end delay to drop in magnitude. In 20 seconds, the protocol sent about 5000 total data packets (not all data packets were the 500 byte data package; some were for communication confirming sent or received the packet). The UDP protocol had its data transmission rate adjusted to match the same amount of data packets. We can see that the TCP protocol goes up to .4 seconds of delay before the congestion control causes the delay to reset. The UDP protocol shows a linear increase in delay until it plateaus at a constant 0.3 seconds. For end-to-end delay, UDP wins out but a more alarming observation can be made with the first two delivery ratio displayed by *Figure 16*. The UDP protocol has under a 2/3 chance to successfully send a packet, while the TCP protocol has a success ratio of 98.6 percent. Since the ratio is fairly low considering the simplicity of the system of 500 bytes packet size and only 3 agents, the

trade-off to slightly increase end-to-end delay with TCP is worth over using UDP. To determine the protocol for bids, the bid simulation was created by keeping the system for auction but adjusting the packet size. The bid functions only require the robots to send one number representing their bid. The data type was assumed floating point, thus the packet size was changed to 64 bytes. To mimic the bid algorithm such that data would be sent from the agents only after data was received, the interval in which data was sent was was adjusted to 0.014285. This was the lowest end-to-end delay in the TCP protocol simulating the auction. As *Figure 15* indicates, the difference in end-to-end delay is significant; the delay is only reduced for the TCP protocol to a cap of 0.25 before congestion control stalls sending more auction data. In fact this is not a good sign; congestion control occurs more in the bid simulation, which means data packets were lost in more intervals. The UDP protocol shows that the end-to-end delay doesn't exceed 0.01 seconds, a significant difference to TCP. The delivery received ratio shown in *Figure 16* is also extremely close to 100 percent, which proves its high consistency. The data collected indicates that reducing the packet size decreases end-to-end delay and increases the delivered received ratio; 64 bytes is small enough of a packet size to have a reliable delivered-received ratio with UDP. Thus, it was concluded that while the TCP protocol is better for sending auction data, the UDP protocol is preferred for sending bid numbers.

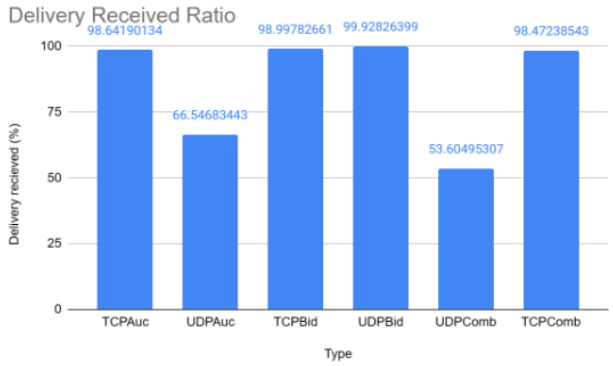

Fig. 16: Delivered Received Ratios for different cases of TCP/UDP implementation and purpose (auction data for SSI and Combinatorial and bid data)

C. NS2: Comparing Auctions

After concluding the optimal protocols for the sections of an auction, the next step is to simulate the different auction systems. Both auction and bid simulations were implemented with 10 auctions were simulated, each auction having a run time of 0.1 seconds. Simulating the SSI auction needed no changes to the system used to compare the communication protocols. Simulating the Combinatorial auction only requires a change to the packet size in the auction system. A package of 10 tasks is assumed, increasing the packet size sent and received to 680 bytes (500 initially with 1 task, then adding 9 more tasks for another 180 bytes). *Figure 17* displays the

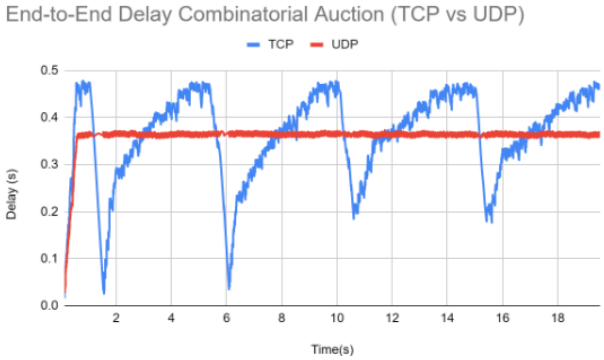

Fig. 17: End-to-End Delay: Comparison for Combinatorial Auction. Same shapes as Figure 14 but different magnitudes of peak delay.

testing of TCP and UDP protocols for sending auction data. The observations that TCP works better for sending auction data and UDP works better for sending bid data is reflected in the figure; with an increase of data packet size, the delivered received ratio decreases. Taking everything observed so far in this paper, a special system was implemented for the Dutch Auction. Given the nature of sending multiple bid numbers from the server to the agents, each auction would require multiple small packets being sent back and forth. To simulate this, both a TCP and a UDP protocol are established for the server to send to the agents. The TCP link is used to send auction data, and then turned off. The UDP link would be used to send bid numbers with the TCP link off until an auction is complete. It should be noted that the first auction the server sends only one bid number after auction data, the second auction simulates two iterations of sending bid numbers, and the third and onward simulates three iterations. This is because three is the maximum number that can fit to keep the rate of 10 auctions per second.

Figure 18, *Figure 19*, and *Figure 20* show the end-to-end delay. All auctions drop some packages that require about 500 ms to resend auction data and bids for a rate of 10 auctions done in 1.5 seconds. The combinatorial auction consistently gives the highest end to end delay plateau, which gives the most average delay. Given the system is perpetually identical to that of the SSI auction, the SSI Auction is better in terms of end-to-end delay as the congestion control kicks in at a similar time. The Dutch auction had the lowest plateau delay but looked to lose packages at shorter intervals. This makes sense as the Dutch auction sends the most number of packages at once, making the congestion window fill up faster. The data matches the observation made previously; increasing packet size increases the end-to-end delay and the chance of dropping packets. It even looks like the time it takes to complete the 10 auctions is slightly longer, but increasing the packet size to 680 does not seem to have a huge effect on it.*Figure 21* confirms this as the the Delivered received ratio is slightly lower for combinatorial auction than SSI auction. Dutch Auction also has a lower ratio as its system is similar to SSI but has

an additional UDP link from the server to the agents. In other words at the moment congestion control happens, more packages are lost at the moment. The Dutch auction seems to be a little worse than SSI in both metrics.

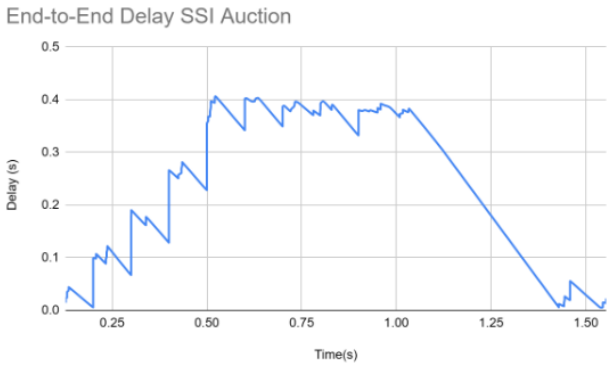

Fig. 18: End-to-End Delay: SSI Auction (10 auctions run at 10 auctions per second)

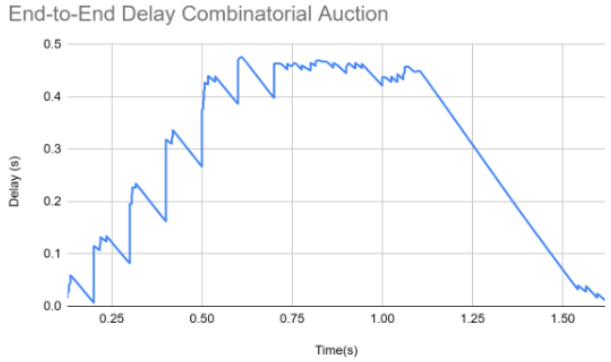

Fig. 19: End-to-End Delay: Combinatorial Auction (10 auctions run at 10 auctions per second)

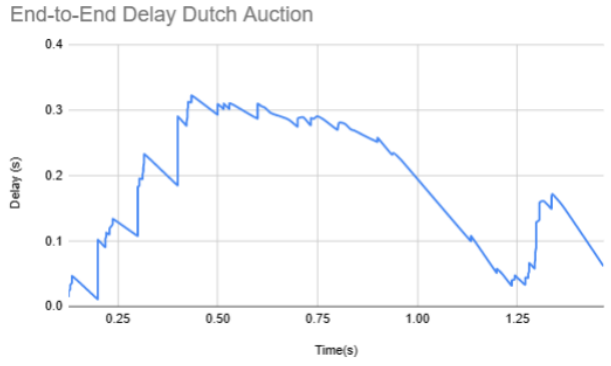

Fig. 20: End-to-End Delay: Dutch Auction (10 auctions run at 10 auctions per second)

V. Conclusion and Future Work

Throughout this paper, we have implemented two different bidding systems and four different auctions. Regarding the task allocation process which is simulated using Python, there

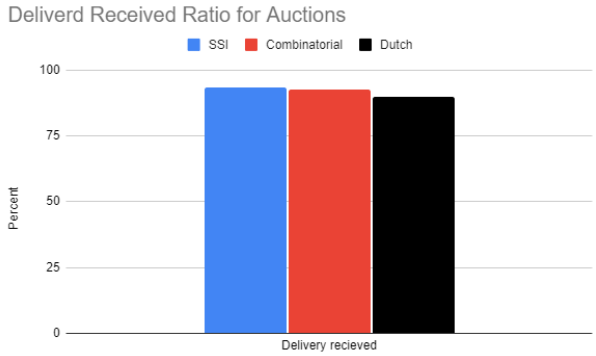

Fig. 21: Delivered Received Ratio of SSI, Combinatorial and Dutch auctions (10 auctions run at 10 auctions per second)

are a few things we can gather. Regarding the two different bid types, it was found that the Cooperative Bid was always better. Although it had higher computation times and memory usage, the task allocation utilizing the selfish bid was so abysmal that we deemed the Cooperative bid far better. For the different auction types, the Sequential Single-Item auction is a simple auction to implement and integrate, but the lack of complexity is followed with poor efficiency and results across all metrics. If the primary concern is how easy it is to implement an auction process without regard for costs in resources, then the SSI auction can be a viable option. The Dutch Auction on the other hand, is also relatively easy to implement. However, the reason why it functioned the same as the SSI auction in this experiment was due to the lack of a dynamic bid. The Dutch Auction is more suitable for scenarios where agents are able to alter their bid, or if the bid is not static throughout the auction rounds. The SSI auction that implements hill climbing has by far the best results in the performance evaluation. The hillclimbing process increases the similarity to that of a Combinatorial Auction without increasing computation complexity too much, and being relatively easier to implement. Furthermore, the scalability of the model in regards to memory usage is a superb feature. However, in regards to time efficiency which was measured by the total required travel time, it falls a far cry short of the Combinatorial auction. This is because the Combinatorial auction's complexity is way higher than the other auctions, increasing implementation difficulty and computational power requirements. This is due to the algorithm attempting to find the most ideal allocation in regards to task completion. However, in regards to the total distance traveled and overall battery consumption as a result, the SSI auction with hillclimbing is more efficient as the hillclimbing process focuses heavily on optimizing this factor. Thus, the hillclimbing variation of the SSI auction is ideal if the goal is to maximize profits and reduce consumption of resources. On the other hand, if resource consumption and computational power aren't as big as an issue, the Combinatorial auction is ideal for applications that require the completion of tasks efficiently in as little time as possible. Overall, for the python

simulations there is a lot more testing that can be done for future work. More auction algorithms can be implemented, a larger variety of agents, a more complex environment for the map, and different contents for the task. Currently we have three pre-programmed robots, with a very simple matrix environment without obstacles, and a delivery system for the tasks. However, to further test the robustness of the algorithms it is imperative that further testing is done for other application scenarios and more complex variables and environments.

In the wireless communication perspective, the balance in metrics between the three auctions must be considered to set up a system. It is clear that the larger the packet size, the more end-to-end delay and the lower the delivered received ratio becomes. However if the packet size is small enough, UDP can still be used as a reliable protocol that runs faster than TCP. This is possible because of the limitations of a fixed number of agents as well as the simplicity of the bid number. Further work may be done by analyzing the effect of adding more agents to the system to the stability of communication. We also showed simulation assuming agents were static. In scenarios where cars want to auction while on the road (perhaps in the process of performing tasks), simulations could be tested when agents are on the move, to test different protocols and analyze our metrics with the variable of movement.

REFERENCES

[1] B. Carlson, "self-driving service: Big companies using autonomous delivery cars," *MotorBiscuit*, 2022.

[2] A. R. Mosteo and L. Montano, "Comparative experiments on optimization criteria and algorithms for auction based multi-robot task allocation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3345–3350, IEEE, 2007.

[3] X. Zheng, S. Koenig, and C. Tovey, "Improving sequential single-item auctions," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2238–2244, IEEE, 2006.

[4] K. Bhawalkar and T. Roughgarden, "Simultaneous single-item auctions," in *International Workshop on Internet and Network Economics*, pp. 337–349, Springer, 2012.

[5] E. Schneider, E. I. Sklar, S. Parsons, and A. Özgelen, "Auction-based task allocation for multi-robot teams in dynamic environments," in *Conference Towards Autonomous Robotic Systems*, pp. 246–257, Springer, 2015.

[6] M. Tennenholtz, "Some tractable combinatorial auctions," in *AAAI/IAAI*, pp. 98–103, 2000.

[7] L. Lin and Z. Zheng, "Combinatorial bids based multi-robot task allocation method," in *Proceedings of the 2005 IEEE international conference on robotics and automation*, pp. 1145–1150, IEEE, 2005.

[8] J.-L. Lin, K.-S. Hwang, and H.-L. Huang, "Variable patrol planning of multi-robot systems by a cooperative auction system," *Cybernetics and Systems*, vol. 43, no. 6, pp. 476–492, 2012.

[9] M. Elango, S. Nachiappan, and M. K. Tiwari, "Balancing task allocation in multi-robot systems using k-means clustering and auction based mechanisms," *Expert Systems with Applications*, vol. 38, no. 6, pp. 6486–6491, 2011.

[10] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," *The journal of supercomputing*, vol. 54, no. 2, pp. 252–269, 2010.

[11] M. G. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. J. Kleywegt, S. Koenig, C. A. Tovey, A. Meyerson, and S. Jain, "Auction-based multi-robot routing.," in *Robotics: Science and Systems*, vol. 5, pp. 343–350, Rome, Italy, 2005.

[12] J. Jesús Roldán, J. Del Cerro, and A. Barrientos, "Should we compete or should we cooperate? applying game theory to task allocation in drone swarms," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5366–5371, 2018.

[13] P. Yan, M.-Y. Ding, and C.-P. Zhou, "Game-theoretic route planning for team of uavs," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826)*, vol. 2, pp. 723–728, IEEE, 2004.

[14] S. Dang, M. Meng, D. Mathews, and R. Kakimzhanov, "Control modeling and signal processing of a library self-delivery robot and its applications," in *Proceedings of the 2013 International Conference on Electrical and Information Technologies for Rail Transportation (EITRT2013)-Volume II*, pp. 383–390, Springer, 2014.

[15] Z. Yin, J. Liu, B. Chen, and C. Chen, "A delivery robot cloud platform based on microservice," *Journal of Robotics*, vol. 2021, 2021.

[16] K.-S. Hwang, J.-L. Lin, and H.-L. Huang, "Cooperative patrol planning of multi-robot systems by a competitive auction system," in *2009 ICCAS-SICE*, pp. 4359–4363, IEEE, 2009.

[17] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "Tmix: a tool for generating realistic tcp application workloads in ns-2," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 3, pp. 65–76, 2006.

[18] F. T. AL-Dhief, N. Sabri, N. A. Latiff, N. Malik, M. Abbas, A. Albader, M. A. Mohammed, R. N. AL-Haddad, Y. D. Salman, M. Khanapi, *et al.*, "Performance comparison between tcp and udp protocols in different simulation scenarios," *International Journal of Engineering & Technology*, vol. 7, no. 4.36, pp. 172–176, 2018.

[19] F. Fund, "Tcp congestion control." WITest, October 29, 2020 [Online].

[20] L. Williams, "Tcp vs udp: Key difference between them." Guru99, Nov. 5, 2022 [Online].