Multi-Robot Task Allocation for Asset Protection

Apoorva Sharma
Aeronautics and Astronautics
Stanford University
apoorva@stanford.edu

Benoit Landry
Aeronautics and Astronautics
Stanford University
blandry@stanford.edu

Joseph Lorenzetti
Aeronautics and Astronautics
Stanford University
jlorenze@stanford.edu

Abstract

As commercial and hobby drone use is becoming more popular, it is important that privacy and security problems related to their use be addressed. Protection of critical infrastructure, such as airports, is a particularly important case that needs addressed. Pursuit-evasion problems have been intensively studied, and companies like Airspace Systems are developing one-on-one defense drones. However for large regions that need protecting, or for protection against multiple intruders, coordinated multi-robot teams could provide higher levels of security.

We propose to develop an multi-agent coordination algorithm that can be used to protect a set of resources against multiple attackers. In this research we will focus our developments on the multi-robot task allocation problem (MRTA). The goal is to allocate defenders to attackers in such a way that will minimize "damage" to the resources, based on predetermined values of each resource.

1 Introduction

We seek to develop an algorithm to determine the policy for a multi-robot system that protects a set of resources from a set of attackers. This multi-robot task allocation problem (MRTA) can further be classified using the taxonomy given in (Gerkey and Mataric, 2004). We assume that the defenders are single-task (ST) robots, meaning they only do one type of task and can only do one task at a time. The tasks are to eliminate attackers, which only requires a single robot to perform, so we say these are single-robot (SR) tasks. The time considered during the allocation could either be instantaneous or extended. An instantaneous allocation (IA) is one that only considers the current state. A time extended allocation (TA) would build a schedule of tasks for each robot, meaning that future states of the system are considered when planning the allocation. This problem would then be classified as ST-SR-IA or ST-SR-TA. Often, for simplicity, the ST-SR-TA problems are approximated by considering the ST-SR-IA variant. In this paper we will make this approximation. Under this approximation the problem seems similar to the Optimal Assignment Problem (OAP), which has been thoroughly studied and solution methods exist. However this problem is complicated beyond the OAP because we assume that tasks can be allocated to more than one robot.

In this paper we present and evaluate two algorithms to compute the best task allocations. The first is a greedy algorithm that evaluates an explicitly defined cost function and the second is a market based algorithm that uses an auction mechanism to allocate tasks. Both of these algorithms are also benchmarked against a brand and bound algorithm that computes the exact optimal allocation for the given explicit cost function.

System Definition The system is defined by the resources, the defenders, and the attackers. The resources are assumed to be heterogeneous and defined by their value, which we mathematically define in the formal problem definition as the scalar cost, c_i, that would be experienced if the resource was hit by an attacker. The attackers and defenders are also heterogeneous. The attacker's attributes are strength and speed. Defenders have three attributes: speed, reach, and strength. The relative strength between an attacker and defender defines how likely it is that the defender would be able to eliminate the attacker. The defender's reach defines the proximity of the defender for which there is a non-zero chance of eliminating the attacker.

Attacker and Defender Interaction We assume that an interaction consists of only one attacker, but potentially multiple defenders. The interaction period begins when a defender moves within reach of an attacker. In the interaction between an attacker and a defender, the outcome of the interaction is defined by a probability of success that is a function of their relative strength. We assume that the first attempt to eliminate an attacker is special (i.e. shooting a net), but if not successful then further attempts occur at $\Delta t_{attempt}$ intervals and the probability of success is discounted heavily (i.e. the defender tries to simply collide with the attacker).

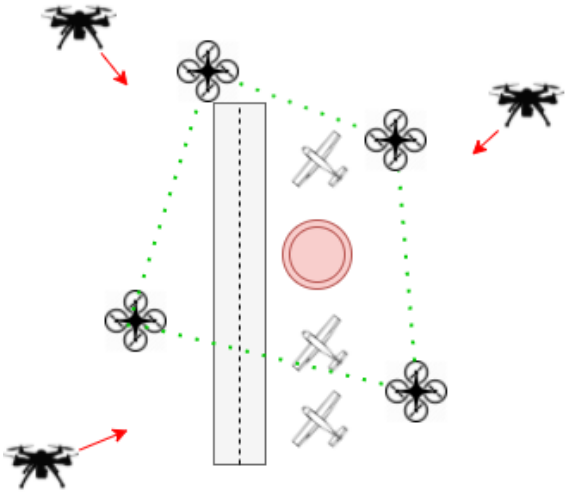

Figure 1: Resource Defense

Attacker Strategy The attackers will follow linear trajectories directly towards their given target and are assumed to not interact to the state of their environment. Each attacker target pairing is also predetermined and constant. Further, we assume that the defenders can measure the attacker's state, so that accurate interception paths can be planned. Since the attacker control policy is known it would theoretically be trivial for the defender to determine the attacker target pairings, so we also just assume those are known a priori.

2 Related Work

The field of multi-robot task allocation has a rich prior literature. Based on the field review done by (Gerkey and Mataric, 2004) and (Khamis et al., 2015), two primary methods of task allocation have been studied: optimization approaches and market based approaches. Optimization approaches generally require more computational effort but also yield better coordination because they are optimizing a global cost function. Market based approaches tend to be more robust to dynamic environments and can be run in a distributed manner more easily.

(Sheridan et al., 2010) presents an optimization approach that allocates tasks to minimize time to intercept attackers. This potentially could be adapted for our problem if the cost was a function of resource value and if it could reallocate when new attackers enter the environment. (Earl and D'Andrea, 2002) presents another optimization based algorithm for a similar problem, choosing to minimize the number of invaders that reach a target. They present a mixed-integer programming solution approach, agents are modeled as single integrators therefore keeping the constraints and objectives linear. One drawback of this approach is that it does not scale well in the number of agents. (Raboin et al., 2013) focuses on a very similar problem, except they have a single resource and they assume uncertainty about attacker identity. Rather than optimize over all possible task allocations, the authors propose repeatedly optimizing over the neighborhood of the current

assignments at regular time intervals, where the neighborhood is defined by simple market-like interactions like task swapping. The utility of an assignment is computed through a model-predictive simulation.

For market based approaches, we reviewed the work of (Vail and Veloso, 2003),(Amador et al., 2014), and (Kalra et al., 2005). Each of these papers discuss slight variations to the market based approach, and each solve different problems. (Vail and Veloso, 2003) is a more traditional auction style algorithm for task allocation, and the coordination is then done with potential fields. The interesting part is that each robot computes the entire auction with information it gathers from the other robots, and assumes the other robots will match its conclusions. (Amador et al., 2014) looks at a "Law Enforcement Problem", which is also similar to ours in the sense that new tasks can arrive and that tasks have different values. This paper uses a Fischer market clearing algorithm, which is slightly different form of an auction. Robots in this case can have more than one task allocated to them at any time, which we may not necessarily want. (Kalra et al., 2005) presents two types of planning using market based algorithms, the passive coordination uses an iterative auction where robots select most profitable plans for themselves and other robots. Then they broadcast this and other robots react with their own. An active coordination is also presented for cases where the passive plan is "marginally profitable" that potentially allows for a more globally optimal plan than a locally optimal plan.

After the task allocation, the problem of intercepting invaders is similar to a pursuer/evader problem, a problem that is well studied. Work like (Pierson et al., 2016) has studied the multi-pursuer multi-evader problem, and provided a solution using safe-area minimization using voronoid regions to ensure capture of the evaders. To allow us to focus on the task-allocation side of this problem, we have chosen to limit the invaders to linear trajectories rather than evasive ones, but relevant literature exists if we wish to extend the work to evasive invaders. Finally, various formulations of the pursuer-evader (Lin et al., 2013; Kaminka et al., 2010) and capture-the-flag (Huang et al., 2015) problems have been made in the context of game theory, specifically differential games. Most of these involve solving an approximate solution of the Hamilton-Jacobi-Isaacs (HJI) equation, with mixed results with respect to scalability.

For our work, we focus our attention on task allocation, studying both optimization based approaches and market based approaches. We simplify the invader dynamics to make the interception problem trivial, but treat interception as a probabilistic event, making our work robust to more complicated interception models as long as the interception success probability can be estimated.

3 Problem Formulation

We first define two sets of agents. The first set is a set of n defenders $D = \{d_0, d_1, \ldots, d_n\}$. The defenders are modeled as single integrators $\dot{x}_d = u_D$ where $x_d = [x_d^{iT} x_d^{iT} ... x_d^{nT}]^T$ and x_d^i is the state in $\mathbb{R}^2$ of defender d_i. We also define a set of m attackers $A = \{a_0, a_1, \ldots, a_m\}$ also modeled as single integrators $\dot{x}_a = u_a$ where $x_a = [x_a^{0T} x_a^{1T} \ldots x_n^{iT}]^T$ and x_a^i is the state in $\mathbb{R}^2$ of attacker a_i. We define a set of k resources $R = \{r_0, r_1, \ldots, r_k\}$ with each their own constant position x_r^i in $\mathbb{R}^2$.

We define a scalar cost $c_i = c(r_i)$ where the cost c_i is the cost incurred by the control system if *any* attacker were to reach the position of resource r_i ($\|x_a^i - x_r^i\| \approx 0$) in finite time T. We later use two different formulations of the problem: *incremental damage* and *total damage*. In the total damage formulation, a resource can only add its own cost c_i at most once to the total cost (the resource is completely lost as soon as one attacker reaches it). In the incremental damage formulation, every time an attacker reaches the resource, the resource's cost c_i is added to the total cost (with no limit on how many times it can be added).

Thus, the goal is to find an assignment m and policies π for the given state of the system $x = [x_d^T x_a^T]^T$, our resource locations x_r, resource costs c, and mapping of attackers to resources γ. Here $m\colon \{1 \ldots n\} \mapsto \{\emptyset, 1 \ldots m\}$ maps every defender d_i to an attacker a_i (or the empty set). We define C as the set of resources at least one attacker successfully reaches in finite time T.

To simplify the planning task, we make the following assumptions regarding the attacker and defender motions: First, we assume that each attacker moves at a fixed velocity corresponding to its maximum speed in the direction of the single resource that it is targeting. Next, we assume that given a specific attacker to target, defender d_i follows the following policy p_{simple}: it first moves at its maximum speed in the direction which intercepts the target attacker in minimum time, and after intercepting the target attacker, the defender can immediately switch to a trajectory matching the attacker and continue along the shared trajectory until the attacker is neutralized or the attacker reaches the target resource. While p_{simple} may not be the optimal low level control policy, it allows the planning algorithm to only consider the different defender/attacker assignments, and not plan over a lower level set of actions.

Our task can then be written as designing an algorithm that solves the following optimization problem:

$$\begin{aligned} \phi(x_d, x_a, x_r, c, \gamma) &= \underset{m,\pi}{\text{argmin}} \quad J(x_d, x_a, x_r, c, \gamma, m, \pi) \\ &\text{subject to} \quad \pi_i = p_{simple}\ i = 1, \ldots, n. \end{aligned}$$

where $J(x_d, x_a, x_r, c, , \gamma, m, \pi)$ is a cost function which measures the expected resource loss induced by the attackers.

We can solve this problem either by directly choosing ϕ to optimize the cost J, or designing a market based algorithm by defining cost and revenue functions that relate to J.

3.1 Cost Function

We design our cost function to relate to the expected resource loss induced by the attackers. We studied performance in two scenarios. The *total damage* scenario has a resource incur damage equal to the total cost of the resource only when the first attacker reaches it, with no further damage occurring when subsequent attackers reach it. The *incremental damage* scenario has the resource incur a fixed amount of damage equal to the resource cost any time an attacker reaches it. Both of these variants are shown below as functions of the intercept success probabilities of all attacker/defender pairs, p_{d_i/a_j}.

Total Damage Cost Function

$$J(x_d, x_a, x_r, c, \gamma, m, p_{simple}) = \sum_{k=1}^{R} c_k \left(1 - \prod_{j \in \gamma_{r_k}} 1 - \left(\prod_{i \in m_{a_j}} 1 - p_{d_i/a_j}\right)\right) \tag{1}$$

Incremental Damage Cost Function

$$J(x_d, x_a, x_r, c, \gamma, m, p_{simple}) = \sum_{k=1}^{R} \sum_{j \in \gamma_{r_k}} c_{k,inc} \prod_{i \in m_{a_j}} \left(1 - p_{d_i/a_j}\right) \tag{2}$$

The derivation of both of these cost functions can be found in the appendix.

3.2 Probability of Defender Eliminating Attacker, $p_{d/a}$

With either cost function, an important step is to determine the probability that a particular defender will eliminate a particular attacker. We assume that the model for intercept involves a first intercept attempt

(e.g. the deployment of a net), followed by repeated, less effective attempts (e.g. collision attempts), with the sequence terminating either when the defender successfully intercepts or the attack reaches its target.

Let p_{ij} be the probability of the defender's first attempt succeeding. We assume the subsequent attempts have a diminished probability of success given by $f \cdot p_{ij}$, where $0 < f < 1$. For our simulations, we chose $f = 0.25$.

The total probability of a successful intercept before the attacker reaches its target is given by the expression:

$$p_{di/aj} = 1 - (1 - p_{ij})(1 - fp_{ij})^{\lfloor \frac{t_{rem}}{t_\delta} \rfloor}$$

where t_δ is the time between subsequent attempts and t_{rem} is the time remaining before the attacker reaches its target.

We model p_{ij} as a function of the relative strengths of the attacker and defender, choosing a sigmoid relation such that when the defender and attacker have equal strength, $p_{ij} = 0.5$. The expression is as follows:

$$p_{ij} = \frac{1}{1 + e^{-\beta(S_{d_i} - S_{a_j})}}$$

β is a parameter that determines how quickly these probabilities approach zero or one. We choose $\beta = 5$.

Note that the task allocation algorithms discussed in the next section only take these probabilities as input, rather than operating on strengths and speeds directly. This makes the algorithms agnostic of the particular interaction model used, making them more generally applicable than for this specific model and application.

3.3 Comments

Note that the cost function that we have defined calculates the expected cost of a fixed allocation. However simply minimizing this function, even with an exhaustive search, will not necessarily give the best action. This cost function cannot account for the fact that a defender could potentially take out multiple attackers if done in the right order. In other words picking the optimal allocation based on this cost function is naive in the sense that it assumes a defender can only do one thing.

However, considering the possibility for reallocations makes the problem significantly more complictated, since the expected loss computed for the allocation would need to consider the change in intercept probabilities over time as a function of changing allocations.

4 Algorithms

4.1 Branch and Bound

A combinatorial optimization problem like the one described here can always be solved through exhaustive search of the solution space. This is however highly inefficient, because entire subspaces of the solution space can be clearly identified as suboptimal from the start. This is the intuition behind branch and bound.

Branch and bound is one of the most commonly used method to solve combinatorial optimization problems (Lawler and Wood, 1966). It consists of exploring the solution space by constructing partial solutions, and discarding sets of possible solutions by evaluating a lower bound on the cost of any of the solutions that can be constructed from a partial solution (see algorithm 1). The key challenge in leveraging branch and bound is coming up with a true lower-bound that is tight enough such that it offsets the computational complexity added by having to evaluate it on every partial solution of a problem.

Algorithm 1 Branch and Bound

```
1: procedure BranchAndBound
2:     A := ∅
3:     x* = heuristic-based solution
4:     J* = COST(x*)
5:     A ⇐ x_root
6:     while A is not ∅ do
7:         choose a branching node x_partial from A
8:         remove x_partial from A
9:         generate children of x_partial
10:        for each child in children do
11:            if child is a full solution then
12:                if COST(child) < J* then
13:                    x* = child
14:                    J* = COST(child)
15:            else if LOWERBOUND(child) < J* then
16:                A ⇐ child
17:    return x*
```

In order to use branch and bound on the problem described above, we propose the following lower bound: for each resource, compute its most optimistic outcome, which corresponds to having every single unassigned defender be assigned to itself. The lower bound is then the sum of each resource's *optimistic* expected cost.

$$C_{lower}(x_{partial}) = \sum_{R} c(r_i | x_{partial}, D_{remaining} \text{ assigned to } r_i)$$

Intuitively, this lower bound is valid because adding defenders to defend a resource can only improve or not change the resource's chances of survival (and therefore its contribution to the total expected cost). Also note that this computation assumes that each attacker's probability of success is independent, which agrees with our formulation of the cost function.

Our implementation uses the depth-first-search formulation of branch and bound. This has the advantage of generating complete solutions quickly, which helps branch and bound in cases where the bound is not guaranteed to be very tight.

4.2 Greedy Algorithm (or Passive Coordination)

Even with the heuristics used in Branch and Bound, the algorithm still scales exponentially with the number of defenders, making it intractable to use for scenarios with many defenders. The practical solution to this problem is to give up exactness for scalability, and the simplest solution to this is a greedy algorithm. Presented formally below, it essentially considers each defender once, and allocates it in a way that minimizes total cost given the current allocation of the other defenders. This is equivalent to the passive coordination scheme discussed in (Kalra et al., 2005). While each defender is assigned optimally given the current assignment of the others, this is still globally suboptimal as the first defenders are allocated assuming the other defenders are not allocated to any attacker, a flawed assumption.

4.3 Market Based Algorithm

Auctions are a market mechanism that have been used for centuries to allocate goods and resources. Auctions seem particularly well suited for multi-robot systems because they are scalable, allow for significant

Algorithm 2 Greedy Task Allocation

```
1: procedure GreedyAllocation(D,A)
2:     current allocation ← {d_i ⇒ NIL | d_i ∈ 𝒟}
3:     current cost ←Cost(current allocation)
4:     for d ∈ 𝒟 do
5:         best allocation ← current allocation
6:         best cost ← current cost
7:         for a ∈ 𝒜 do
8:             allocation ← current allocation
9:             allocation[d] := a
10:            cost ← Cost(allocation)
11:            if cost < best cost then
12:                best allocation ← allocation
13:                best cost ← cost
14:        current allocation ← best allocation
15:        current cost ← best cost
16:    return current allocation
```

compartmentalization of data and control, and are efficient in computation and communication (Gerkey and Mataric, 2002). The market algorithm we developed aims to capture these benefits while performing comparably or better than the other algorithm types discussed in this paper. This algorithm conducts two sequential rounds: a preliminary contract auction round, followed by a subcontract auction round.

Contract Auction The contract auction round produces a one to one pairing between defenders and attackers, called a "contract". At the beginning of the round the defenders will compute the "profit" they would gain for going after each attacker. In this algorithm the profit is the expected damage that would be mitigated to the system. The defenders then choose which contract would give them the best profit and bid to take the contract. The defender with the best bid (that would expect to mitigate the most damage) receives the contract. This is repeated with all remaining attackers until all contracts have been awarded.

Subcontract Auction When multiple defenders desired the same attacker contract, only one is awarded the contract. This round allows for defenders who either didn't get a contract or didn't get their preferred contract in the first round to switch to another contract or subcontract. We affectionately label these defenders as "losers" and denote this set $\mathcal{L}$. In each round of the subcontract auction, the losers will request quotes from the set of defenders who received contracts in the first round, $\mathcal{D}_{contractors}$, who were allocated to attackers that would have produced a better profit than the loser's current allocation. The new profit that is returned in the quote is the marginal increase in expected damage mitigated by having that loser join the contract owner. If any of those marginal profits are better than the loser's current profit it will submit a bid for the best of the marginal profits. Then, just as in the contract round, the best bid is awarded the subcontract. When a loser is awarded a subcontract, it is removed from the set $\mathcal{L}$ before the next round. This round ends when $\mathcal{L} = \emptyset$ or when no defender in $\mathcal{L}$ can increase their profit by leaving their current task for a subcontract.

Note that removing a defender that is awarded a subcontract from the set $\mathcal{L}$ won't affect the results of the next rounds. The only way a defender who received a subcontract would want to switch is if another subcontract better than the current one appears. But all possible subcontracts that are better than the current are ones that the defender had already bid for and lost. Thus if no subcontract that has already been awarded vanishes, then the defender won't leave its current subcontract.

Computation Time The number of rounds in the contract auction will always be $\min(D, A)$. In the subcontract auction, there will always be at least one subcontract awarded (if none are awarded it is because

no defender wanted to switch, and subcontract auction would be over). Further, when a defender receives a subcontract it will not be able to find a better option anywhere and is removed from the next round. Therefore the number of rounds in the subcontract auction will be upper bounded by $D-1$. In addition to the auctions, each defender must compute locally its profits for each attacker and prioritize them in order to bid in the auctions. Overall, the computation time for this algorithm is $\mathcal{O}(AD)$.

Algorithm 3 Market Based Task Allocation

```
1: procedure ContractAuction(D,A)
2:     for d ∈ 𝒟 do
3:         d.bids ← ComputeBids(d, 𝒜)
4:         d.preference ← SortByProfit(d.bids)
5:         d.choice ← d.preference[0]
6:     ℒ ← ∅
7:     while Contracts Remain AND Bidders Available do
8:         for d ∈ 𝒟 do
9:             Bid(d.choice)
10:        for a ∈ 𝒜 do
11:            [d_winner, {d_losers}] ← EvaluateBids(bids on a)
12:            AwardContract(d_winner, a)
13:            ℒ ← ℒ + {d_losers}
14:            𝒟 ← 𝒟 − d_winner
15:            𝒜 ← 𝒜 − a
16:        for d ∈ ℒ do
17:            d.choice ← UpdateChoice(d.preference, 𝒜)

1: procedure SubcontractAuction(D,A,L)
2:     𝒟_contractors = 𝒟 − ℒ
3:     while ℒ ≠ ∅ do
4:         for d ∈ ℒ do
5:             d.potential ← d.preference[0 : d.choice]
6:             d.newprofits ← GetQuotes(d, 𝒟_contractors[d.potential])
7:             d.best ← EvalutateNewProfit(d.newprofits, d.currentprofit)
8:             if d.best ≠ ∅ then
9:                 Bid(d.best)
10:        ℒ ← ∅
11:        for d_contractor ∈ 𝒟_contractors do
12:            if {bids on d_contractor} ≠ ∅ then
13:                [d_winner, {d_losers}] ← EvaluateBids(bids on d_contractor)
14:                AwardSubcontract(d_winner, d_contractor)
15:                ℒ ← ℒ + {d_losers}
```

Algorithm Characteristics This algorithm has distributed and centralized traits, and is able to capture benefits from each. It is centralized in the sense that the auctioneer must be able to send and receive communication from all agents in order to award contracts to the most qualified agent. However, the auctioneer role can be taken on by any agent, which means that the system is robust to single point failures of any of the agents. The main benefit from the centralized structure is that better coordination emerges in the behavior of the system. It is worth noting that market based algorithms could also be completely distributed, but generally speaking auction algorithms are most efficient in a centralized scheme.

The algorithm is distributed in the sense that each agent computes its own best actions based on communication it receives from other agents about tasks. This enables the algorithm to be scalable to large systems of agents. However, since the auction algorithm requires that each agent be able to bid on all tasks, this algorithm may not scale well with number of tasks. Adjustments could be made to handle large task

numbers, (such as local markets) but at the expense of global efficiency.

5 Results

We implemented the three algorithms described above in Matlab and evaluated their performance.[1] The benchmarks were all run on the same single core computer. We looked at how each algorithm scaled with respect to the number of attackers and defenders, by comparing their run-time and the expected cost of the solutions they reported.

Note that a scenario has a large number of variables: the number of attackers, defenders, and resources, the strengths and speeds of the attackers and defenders, the initial locations of the attackers, the assignment of attackers to resources, and the value of the resources. We focused our analysis on two variables in particular: the dependence on the number of attackers and defenders (i.e. the size of the problem). Thus, we chose to measure the performance and runtime in expectation over a representative sample of the other variables at different fixed values on the number of attackers and defenders. These variables were drawn from uniform distributions over the following ranges:

Parameter	**Range**
Resource Values	[1,10)
Attacker Strength	[0,1)
Attacker Speed	[5,15)
Attacker Initial Position	1000m at uniform random angle
Attacker Target	Uniform over resources
Defender Strength	[0,1)
Defender Speed	[5,15)

The number of resources, resource positions and initial defender positions were all fixed.

In Figure 2 we can see that the execution time is as expected for the different algorithms. The branch and bound algorithm execution time become intractable for large systems, while the execution times for the greedy and market algorithms only increase with $O(AD)$.

Below in Figures 3, 4, and 5 we present a comparison of the performance of the different algorithms. There are a couple of things to note in these figures. First is that the branch and bound algorithm was not run for all cases due to long execution time for larger problems. Second is that the cost is normalized by the total amount of potential damage that could be incurred by the system for the configuration. This is why we tend to see the cost percentage increasing for larger problems under the total damage scenario (total potential damage stays constant), but see a relatively constant cost percentage for the incremental damage scenario (total potential damage increases with A).

[1] All the code for this project is available on GitHub at https://github.com/blandry/defend_the_flags

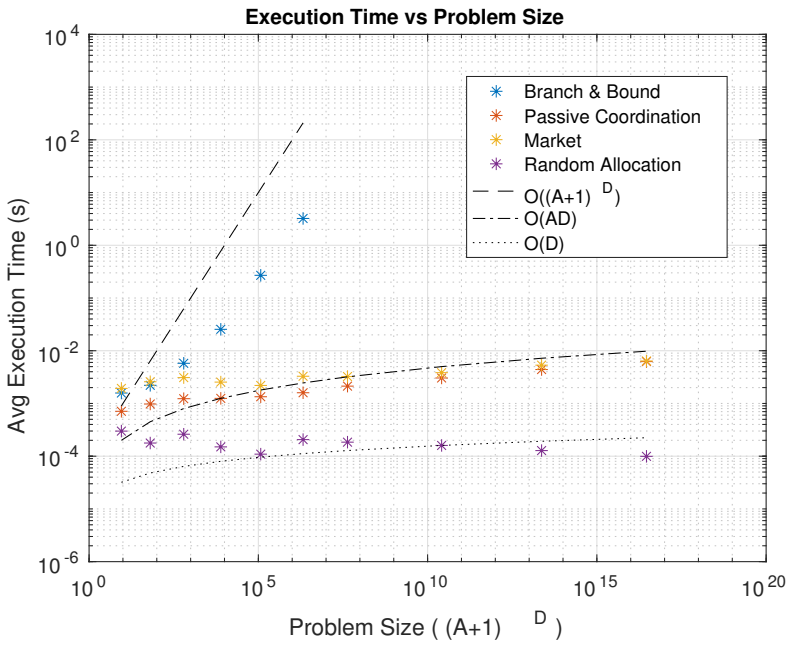

Figure 2: Execution Time as a function of problem size. Slopes of curves representing $O((A+1)^D)$, $O(AD)$, and $O(D)$ are plotted in black as reference.

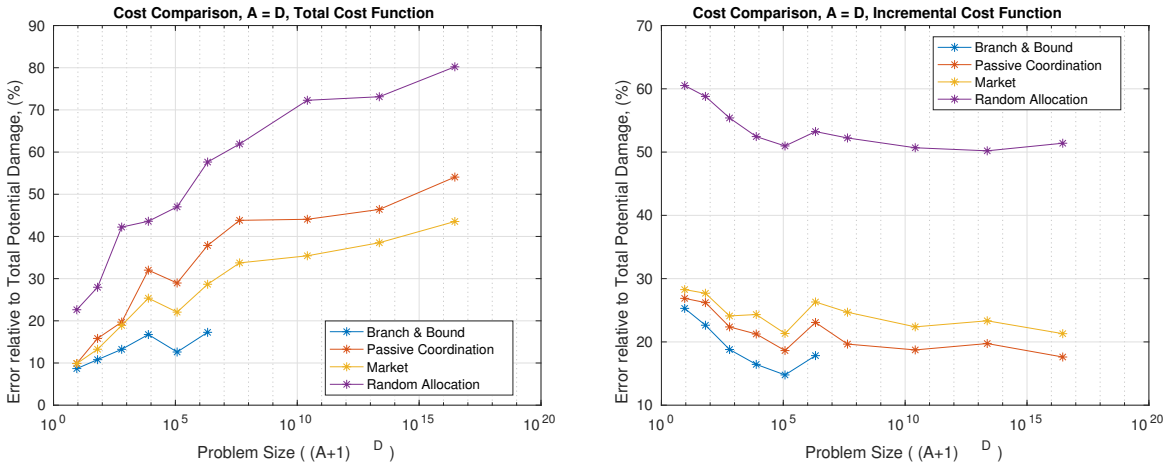

Figure 3: Cost Comparison for Total and Incremental Damage Scenarios, $A = D$

In Figure 3, when the number of attackers is the same as defenders, the market and greedy algorithms outperform random allocation. When considering the total resource loss case, the market algorithm outperforms the greedy one by a 5-10% margin. When considering incremental damage accrual, the greedy algorithm outperforms the market by a 1-5% margin.

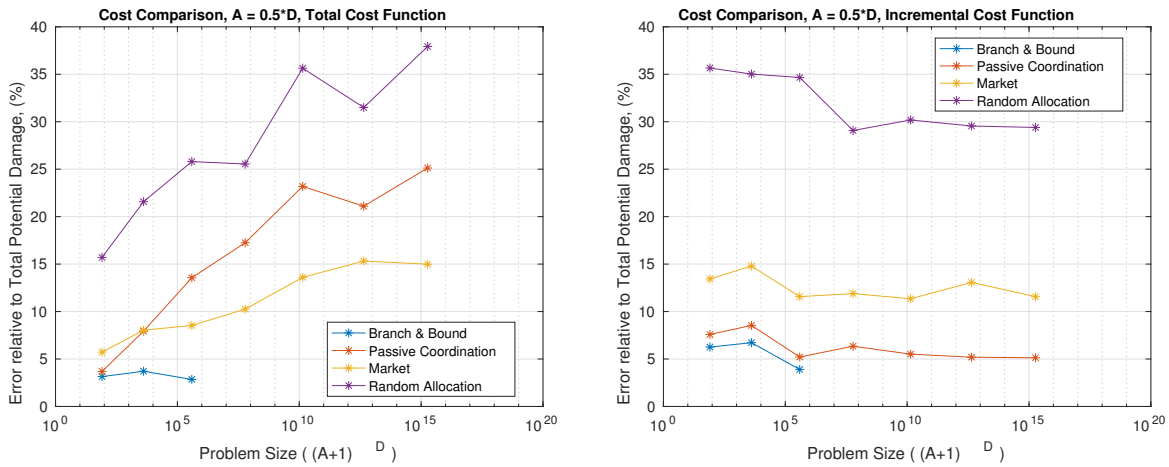

Figure 4: Cost Comparison for Total and Incremental Damage Scenarios, $A = \frac{1}{2}D$

In Figure 4, when the number of attackers is half the number of defenders, the market and greedy algorithms still outperform random allocation. When considering the scenario of total resource loss, the market algorithm outperforms the greedy one by a 5-10% margin. When considering incremental damage accrual, the greedy outperforms the market by a 5-10% margin.

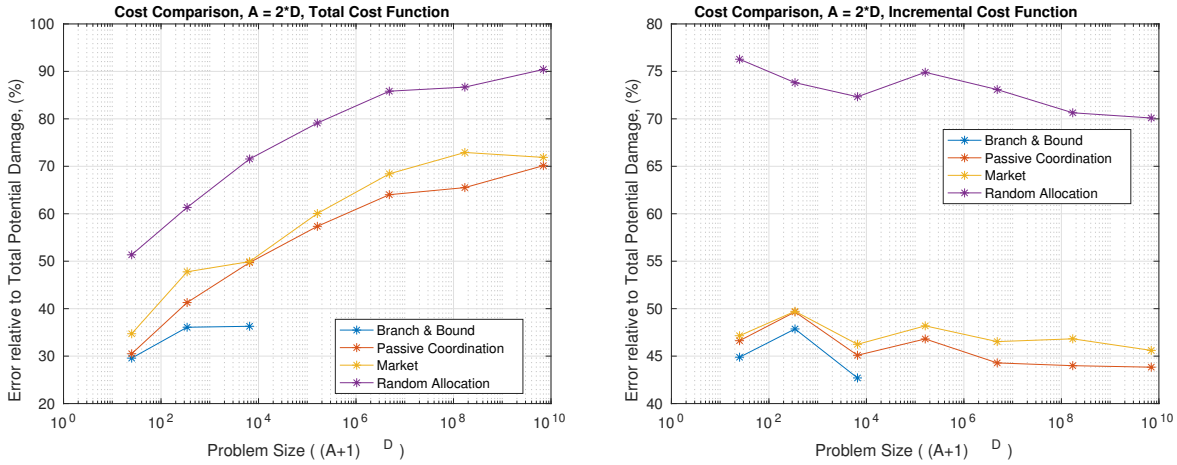

Figure 5: Cost Comparison for Total and Incremental Damage Scenarios, $A = 2D$

In Figure 5, the number of attackers is twice the number of defenders. When considering the scenario of total resource loss, the greedy algorithm outperforms the market by a 0-5% margin. When considering incremental damage accrual, the greedy outperforms the market by a 0-5% margin.

6 Discussion

We also performed simulations of the system following the dynamics discussed above, and with reallocation after events occurred (an event is either resource loss or attacker elimination). The results of these simulations provided some additional insight into the behaviors of the algorithms, in addition to validating the expected cost functions.

The first insight is that the greedy algorithm had a tendency to form coalitions against attackers when planning with the total damage cost function, but uses a dispersed allocation when using the incremental damage cost function. This observation from the simulation was key to allowing us to understand why the greedy algorithm performed more poorly than the market algorithm for the total damage scenario.

We hypothesize that dispersing the defenders would minimize the expected damage, such as is done in the market algorithm, and is done for the greedy algorithm with the incremental damage scenario. But the greedy algorithm has a flaw in dealing with the total damage scenario because the defenders choose in sequential order and only based on what the defenders before them have chosen. For example, consider the case with two resources, three attackers, and four defenders, and you have attacker A1 going for resource M and attackers A2 and A3 going to resource N. Under the greedy algorithm, the first defender chooses its own best allocation, lets say it goes for A1 (that is headed to resource M). Now let's consider how the greedy algorithm will act when allocating the second defender. Because there are two attackers going for resource N, the second defender sees it as futile to go for A2 or A3 because the other would still destroy the resource. Therefore it will choose to help the other defender. This could potentially continue for the third defender, the fourth defender, and so on. This is an example of a lack of coordination that is common in greedy algorithms. We would expect this to be a more common occurrence with large ratio of attackers to resources.

The market algorithm does not run into this problem because in the subcontract round the marginal profit gained by helping another defender diminishes quickly with more defenders, which discourages coalition forming. In Figures 6 and 7 we can see for the total damage scenario the greedy algorithm forming coalitions to go after some attackers while in the market uses a dispersed allocation.

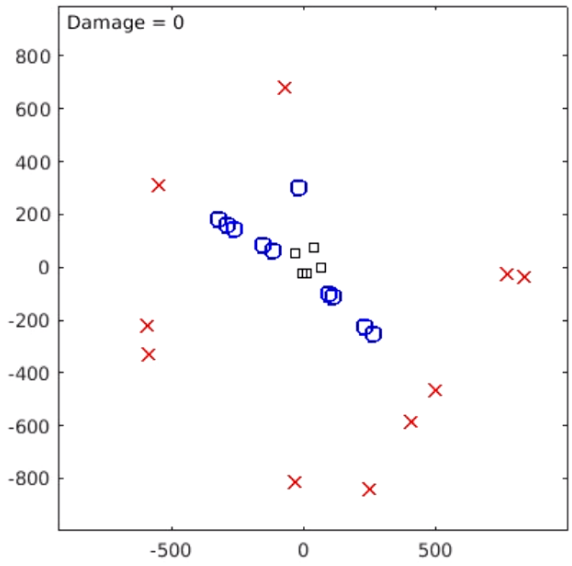

Figure 6: Total Damage Simulation: Greedy

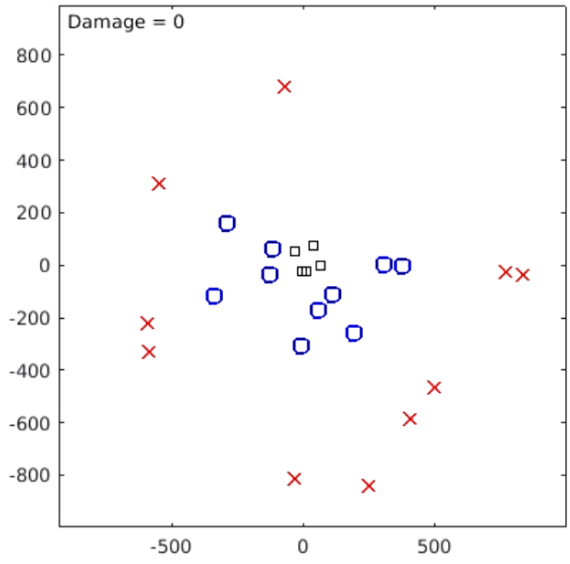

Figure 7: Total Damage Simulation: Market

7 Conclusion

In this work, we presented the problem of defending high value targets, like the buildings of an airport, from a group of attacker drones by using another set of defender drones. We first developed a probabilistic model of the problem, which we modeled as a multi-robot task allocation (MRTA) problem, also known as a combinatorial optimization problem. Next, we developed three algorithms to solve the proposed problem. The first one, an implementation of the well-known branch and bound with a lower bound we derived, solves

the problem exactly but its computational time scales poorly with respect to the number of attackers and defenders. Next, we turned to approximate solutions and developed a greedy algorithm that scales better than branch and bound, but that suffers from being susceptible to local minima. Finally, we proposed a market-based algorithm that seems to offer a good compromise between performance and robustness to local minima. We also point out that scalability allowed us to integrate the market-based algorithm inside of a model-predictive control (MPC) framework, which in turn enables reallocations at runtime and leads to overall better performance (albeit still sub-optimal). This would also be critical for the ability to handle a more dynamic system, such as if more attackers entered at random times. Finally, we presented a series of simulation results that substantiate our claims. In future work, we would like to characterize the problem better, perhaps by exploring the possibility of this problem being NP-hard. We would also like to more fully understand the behavior of our market-based task-allocation algorithm by proving some of its important properties.

References

Amador, S., Okamoto, S., and Zivan, R. (2014). Dynamic multi-agent task allocation with spatial and temporal constraints. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 1495–1496, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Earl, M. G. and D'Andrea, R. (2002). Modeling and control of a multi-agent system using mixed integer linear programming. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 1, pages 107–111. IEEE.

Gerkey, B. P. and Mataric, M. J. (2002). Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768.

Gerkey, B. P. and Mataric, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *I. J. Robotics Res.*, 23(9):939–954.

Huang, H., Ding, J., Zhang, W., and Tomlin, C. J. (2015). Automation-assisted capture-the-flag: a differential game approach. *IEEE Transactions on Control Systems Technology*, 23(3):1014–1028.

Kalra, N., Ferguson, D., and Stentz, A. (2005). Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1170–1177. IEEE.

Kaminka, G. A., Erusalimchik, D., and Kraus, S. (2010). Adaptive multi-robot coordination: A game-theoretic perspective. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 328–334. IEEE.

Khamis, A., Hussein, A., and Elmogy, A. (2015). Multi-robot task allocation: A review of the state-of-the-art. In *Cooperative Robots and Sensor Networks 2015*, pages 31–51. Springer.

Lawler, E. L. and Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719.

Lin, W., Qu, Z., and Simaan, M. A. (2013). Multi-pursuer single-evader differential games with limited observations. In *American Control Conference (ACC), 2013*, pages 2711–2716. IEEE.

Pierson, A., Wang, Z., and Schwager, M. (2016). Intercepting rogue robots: An algorithm for capturing multiple evaders with multiple pursuers. *IEEE Robotics and Automation Letters*.

Raboin, E., Svec, P., Nau, D., and Gupta, S. K. (2013). Model-predictive target defense by team of unmanned surface vehicles operating in uncertain environments. In *2013 IEEE International Conference on Robotics and Automation*, pages 3517–3522.

Sheridan, P. K., Kosicki, P., Liu, C., Nejat, G., and Benhabib, B. (2010). On-line task allocation for the robotic interception of multiple targets in dynamic settings. In *Advanced Intelligent Mechatronics (AIM), 2010 IEEE/ASME International Conference on*, pages 291–296. IEEE.

Vail, D. and Veloso, M. (2003). Multi-robot dynamic role assignment and coordination through shared potential fields. In Schultz, A., Parker, L., and Schneider, F., editors, *Multi-Robot Systems*. Kluwer.

A Appendix

A.1 Expected Future Incremental Damage

The expected system damage given in Eq. 4 only accounts for attackers that are known. An important extension to this is to consider expected damage due to attackers that are on their way but not within detectable range (phantoms, ζ). This could be accounted for in different ways, but no matter what it is based on some assumptions.

We could account for this by creating a cost to allocate a defender to an attacker. This would essentially be saying that by allocating a defender to a known attacker you are expecting to incur some damage from a phantom attacker. This models the prospect of phantoms existing without having to make any assumptions about the phantoms. In this case the cost function would be augmented with:

$$\sum_{1}^{j} \sum_{i \in m_{a_j}} c_{i,allocation}$$

Alternatively, we could decide to make some assumptions about the phantoms that will hopefully provide a better model.

1. Phantoms are more likely to go after valuable targets.

2. Phantoms are more likely to come soon after current attackers.

To address the first assumption we can assume that the probability a phantom ζ goes after resource r_k is given by its fractional system value, $p_{k,target} = \frac{c_k}{\sum_{n=1}^{R} c_n}$ and then the expected damage a phantom would do if it succeeded is given as a constant $C = \sum_{k=1}^{R} c_k p_{k,target}$.

The expected damage to the system is then equal to the expected damage from a phantom times the expected number of phantoms. Determining the expected number of phantoms is difficult, so we just create an approximate calculation. A simple way to do this is to assume the number of phantom attackers N_ζ at a given time after the most recent attacker group entered is given by a inverse relationship

$$N_\zeta = \frac{\#}{(2\# - 1)\frac{t}{t^\star} + 1}$$

where # is the number of attackers that entered since the last allocation, and $t^\star$ is the time after the last attacker entered that you want to assume the chance of phantoms existing is small. A good approximation for this value could just be the system natural period because for any time beyond that you will have some defenders freed up after already handling existing attackers.

Therefore the addition to the expected system damage is

$$\mathbb{E}(damage_{phantoms}) = N_\zeta C \prod_{i \in m_\zeta} \left(1 - p_{d,i/\zeta}\right)$$

A.2 Cost Function Derivations

To develop the cost function we define some parameters:

$$p(d\ eliminates\ a) = p_{d/a}$$

Each defender d_i is allocated only one attacker a_j, but each attacker may have multiple defenders. With $m_{a_j} = \{..,i,..\}$ defining the set of defenders d_i that are allocated to an attacker a_j, we can define the probability that an attacker a_j succeeds by recognizing that attacker success occurs in the event when all defenders fail.

$$p(d\ fails\ a) = 1 - p_{d/a}$$

$$p_{a_j,success} = \prod_{i \in m_{a_j}} 1 - p_{d_i/a_j}$$

Similar to before, each attacker a_j is assigned to only one resource r_k, but each resource may have multiple attackers. The set of attackers a_j that are heading for resource r_k is given by $\gamma_{r_k} = \{..,j,..\}$.

A.2.1 Total Damage

When an attacker reaches a resource we say that the resource is completely destroyed and that no further damage can be done to it. In this case we define the probability that a resource r_k is destroyed as $p_{r_k,lost}$. This probability is the probability of the event that any one attacker succeeds.

$$p_{r_k,lost} = 1 - \prod_{j \in \gamma_{r_k}} 1 - p_{a_j,success}$$

From this setup the expected damage to the entire system is the sum of the expected damage to each resource.

$$\mathbb{E}(damage) = \sum_{k=1}^{R} c_k p_{r_k,lost}$$

Therefore the cost function for the system is the total expected damage, which we wish to minimize. The control is the allocation of defenders to attackers, $m = \{m_{a_j} | \forall j\}$

$$\mathbb{E}(damage) = \sum_{k=1}^{R} c_k \left(1 - \prod_{j \in \gamma_{r_k}} 1 - \left(\prod_{i \in m_{a_j}} 1 - p_{d_i/a_j}\right)\right) \tag{3}$$

A.2.2 Incremental Damage

Alternatively we could model the damage as being incremental. This means that when an attacker hits a resource a defined amount of damage is incurred to the resource. Further we assume that the resource can

incur this incremental damage an infinite number of times. In this case we still have the same probability an attacker succeeds, $p_{a_j,success}$, but the expected damage is different.

The expected damage to a resource r_k by an attacker a_j is simply the incremental damage multiplied by the probability of a_j succeeding.

$$\mathbb{E}(damage\,to\,r_k) = \sum_{j \in \gamma_{r_k}} c_{k,inc} p_{a_j,success}$$

Therefore the total expected damage to the system is given by

$$\mathbb{E}(damage) = \sum_{k=1}^{R} \sum_{j \in \gamma_{r_k}} c_{k,inc} \prod_{i \in m_{a_j}} \left(1 - p_{d_i/a_j}\right) \tag{4}$$

In this case, the incremental damage could be assumed constant or it could vary with the number of successful attacks.

A.2.3 Matrix Form

We can rewrite the cost functions in matrix form. The cost vector is defined as $c \in \mathbb{R}^{1xR}$ where each element corresponds to the total or incremental cost of a particular resource. $M \in \mathbb{R}^{DxA}$ is a binary matrix that maps each defender to an attacker. The j^{th} column of M gives all of the defenders allocated to a_j, (m_{a_j}). $\Omega \in \mathbb{R}^{RxA}$ is also a binary matrix that maps each attacker to a resource. The k^{th} row of Ω gives all attackers allocated to r_k. $\Psi \in \mathbb{R}^{DxA}$ is a matrix that gives the probability of defender success against attackers. Element $\Psi_{i,j} = p_{d_i/a_j}$.