# Report: Multiagent Dynamic Task Assignment Based on Forest Fire Point Model

Ratnangshu Das, Mohd Faizuddin Faruqui and Sawarkar Shubham Ganesh

***Abstract*— The paper studies how to assign tasks efficiently during forest fires, aiming to minimize completion time. It considers fire spread, agent abilities, and deployment numbers. It proves that optimal task assignment leads to equal completion times, establishes deployment strategies, and proposes a dynamic assignment method. MATLAB simulations validate its effectiveness.**

## I. Introduction

The paper [1] delves into the complexities of multiagent dynamic task assignment in forest fire management. It explores the optimization problem considering various factors like fire spread, firefighting agent characteristics and movement speeds, and the number of deployed agents. The primary aim is to minimize task completion time. The research establishes a model for fire spread and task assignments, proving that an optimal static task assignment ensures uniform task completion times under specific assumptions. Additionally, it calculates the optimal solution for static task assignment, laying the groundwork for both initial and dynamic deployment strategies. A dynamic task assignment scheme is proposed, ensuring reduced completion times with each reassignment and balancing task completion times. MATLAB simulations validate the effectiveness of this dynamic scheme against an auction algorithm. The study aims to guide decision-makers in designing strategic assignment approaches and solving optimization problems in diverse scenarios.

## II. Problem Formulation

### A. Fire Model

Assuming that the perimeter of the burning region forms a perfect circle with a radius of $r$, we represent the fire point area as $s$:

$$s = \pi r^2$$

$s$ expands with a radial displacement of $\Delta r$, representing the increase in radius within a specific time period $\Delta t$ as illustrated in figure 1. The spread in fire point's area $\Delta s_1$ is thus calculated as:

$$\begin{aligned}(s + \Delta s_1) &= \pi(r + \Delta r)^2,\\ \Delta s_1 &= \pi(r + \Delta r)^2 - s = \pi(r + \Delta r)^2 - \pi r^2\\ &= \pi(2r + \Delta r)\Delta r = 2\pi r\Delta r + O(\Delta r^2)\\ &\approx 2\pi r\Delta r = l(s)\Delta r,\end{aligned}$$

where $l(s)$ is the fire point perimeter.

Let the execution of an agent be $\beta$, which represents the area the agent can extinguish within a unit of time. Hence, the area extinguished by the agent $\Delta s_2$ within $\Delta t$

$$\Delta s_2 = \beta\Delta t.$$

Therefore, the net change in fire point area is:

$$\Delta s = \Delta s_1 - \Delta s_2 = l(s)\Delta r - \beta\Delta t.$$

Subsequently, the rate of change of fire point area is:

$$\dot{s} = l(s)\alpha - \beta,$$

where $\alpha = \lim_{\Delta t\to 0} \frac{\Delta r}{\Delta t}$ is the radial velocity.

We further assume $k = \frac{l(s)}{\sqrt{s}}$ to be a constant. Thus, the fire point model boils down to:

$$\dot{s} = k\alpha\sqrt{s} - \beta. \tag{1}$$

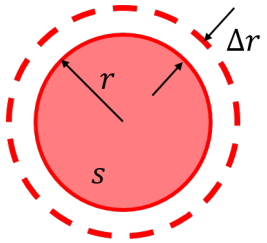


Fig. 1: Fire Spread Model.

### B. Problem Statement

Consider an environment as shown in figure 2, with $N_A$ agents and $N_T$ fire points. $\Lambda_A = \{1, \ldots, N_A\}$ is the set of agents and $\Lambda_T = \{1, \ldots, N_T\}$ is the set of tasks or fire points. For the $j$th fire point, $s_j$ is the area, $\alpha_j$ is the radial velocity, $\Lambda_j$ is the set of agents deployed on it with a total execution capacity of:

$$\beta_j^{\Sigma} := \sum_{i\Lambda_j} \beta_i,$$

where $\beta_i$ is the execution capacity of the $i$th agent. Thus, the multiple fire point model is:

$$\dot{s}_j = k\alpha_j\sqrt{s_j} - \beta_j^{\Sigma}, \forall j \in \Lambda_T. \tag{2}$$

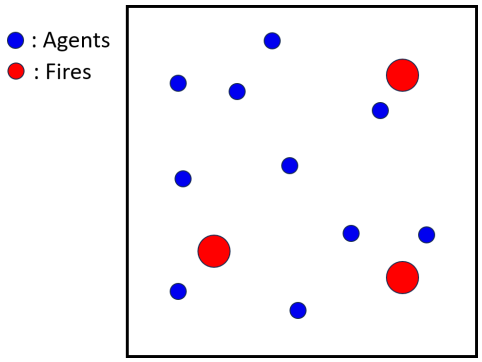


Fig. 2: Environment.

## III. Dynamic Task Deployment

### A. Initial Deployment

*1) Step 1:* The travel time from agent $i$ to fire point $j$ is:

$$t_{ij} = \frac{d_{ij}}{v_i}, \tag{3}$$

where $d_{ij}$ is the distance between agent $i$ to fire point $j$ and $v_i$ is the average speed of agent $i$. Given initial position of the agents and fire point locations, we compute $t_{ij}$, for all $i \in \Lambda_A, j \in \Lambda_T$, and calculate the initial time matrix

$$\begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1N_T} \\ t_{21} & t_{22} & \cdots & t_{2N_T} \\ \vdots & \vdots & \ddots & \vdots \\ t_{N_A1} & t_{N_A2} & \cdots & t_{N_AN_T} \end{bmatrix} \tag{4}$$

*2) Step 2:* The course of action undertaken in Step 2 to allocate agents to the biggest and nearest fire points, ensuring all the fire points are assigned an adequate execution capacity, is summarised in the flowchart in figure 3.

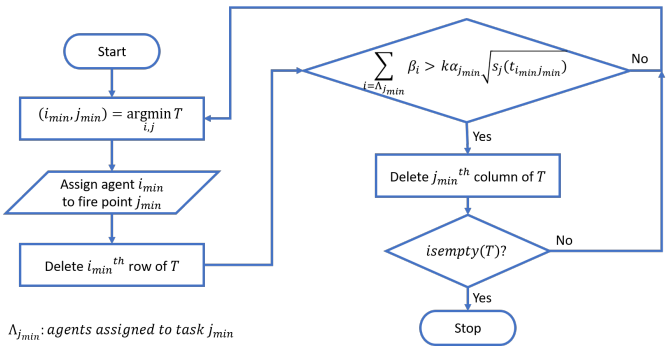


Fig. 3: Initial Deployment: Step 2.

*3) Step 3:* After Step 2, the unallocated agents are assigned to the biggest and nearest fire points through Step 3, summarised in the flowchart in figure 4.

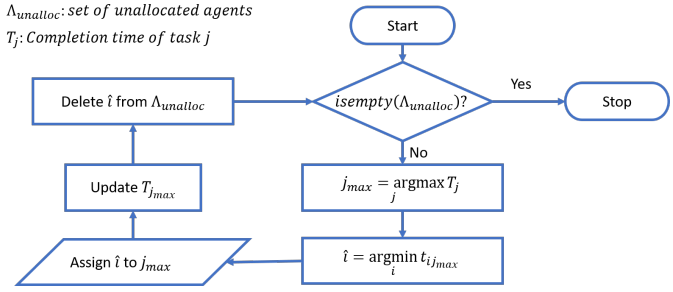


Fig. 4: Initial Deployment: Step 2.

### B. Dynamic Deployment

The initial deployment is a feasible scheme, but not optimal. Dynamic deployment adds to the efficiency of the operation, taking into consideration the time-varying nature of the fire point model.

*1) Step 1::* At each time step $t_k$, we calculate the optimal deployment $\beta_j^*(t_k), \forall j \in \Lambda_T$ and the corresponding time of completion $T^*(t_k)$, solving the following set of equations:

$$T^*(t_k) = \frac{2\beta_j^*(t_k)}{(k\alpha_j)^2} ln\left(\frac{\beta_j^*(t_k)}{\beta_j^*(t_k) - k\alpha_j\sqrt{s_j(t_k)}}\right) - \frac{2}{k\alpha_j}\sqrt{s_j(t_k)}, \quad \forall j \in \Lambda_T, \tag{5}$$

$$\sum_{j\in\Lambda_T} \beta_j^*(t_k) = Q = constant. \tag{6}$$

*2) Step 2::* We first create a list of fire points with sub-optimal execution capacity assigned to them and need reinforcement:

$$\Omega_N(t_k) = \{j | \beta_j^{\Sigma}(t_k) < \beta_j^*(t_k)\}.$$

We also create another list of fire points working at execution capacity over the optimal solution and can provide reinforcement:

$$\Omega_R(t_k) = \{j | \beta_j^{\Sigma}(t_k) \geq \beta_j^*(t_k)\}.$$

*3) Step 3::* From the list of fire points that can provide reinforcement $\Omega_R(t_k)$, we prepare a candidate agent set $\mathcal{A}_R$ using the algorithm represented through the flowchart in figure 5

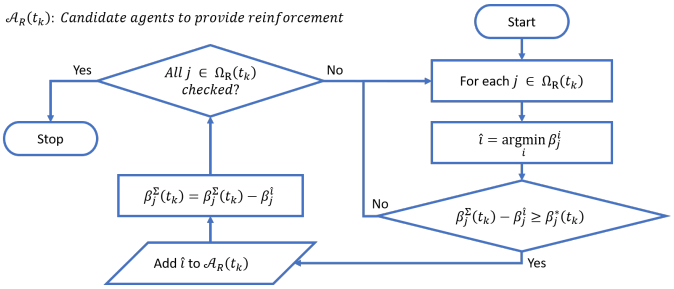


Fig. 5: Dynamic Deployment: Candidate Agents.

*4) Step 4::* We assign the first agent $i_1$ of $\mathcal{A}_R$ with minimum execution capacity from its current fire point $j_{i_1}$ to biggest fire $j_{N_T}$. Note that, while the completion time $T_{j_{i_1}}$ of task $j_{N_T}$ might go down, the completion time $T_{j_{N_T}}$ of task $j_{i_1}$ will simultaneously go up. Further, we have to take into account the time taken by agent $i_1$ to travel from $j_{i_1}$ to $j_{N_T}$. To determine whether the redeployment is more efficient, we compute the completion time of tasks $j_{i_1}$ and $j_{N_T}$ respectively:

$$T_{j_{i_1}} = \frac{2(\beta^{\Sigma_1}_{j_{i_1}}(t_k))}{(k\alpha_{j_{i_1}})^2} \ln\left(\frac{\beta^{\Sigma_1}_{j_{i_1}}(t_k)}{\beta^{\Sigma_1}_{j_{i_1}}(t_k) - k\alpha_{j_{i_1}}\sqrt{s_{j_{i_1}}(t_k)}}\right) - \frac{2}{k\alpha_{j_{i_1}}}\sqrt{s_{j_{i_1}}(t_k)},$$

$$T_{j_{N_T}} = t_{i_1,j_{i_1}\to j_{N_T}} + \frac{2(\beta^{\Sigma_1}_{j_{N_T}}(t_k))}{(k\alpha_{j_{N_T}})^2} \ln\left(\frac{\beta^{\Sigma_1}_{j_{N_T}}(t_k)}{\beta^{\Sigma_1}_{j_{N_T}}(t_k) - k\alpha_{j_{N_T}}\sqrt{s_{j_{N_T}}(t_k + t_{i_1,j_{i_1}\to j_{N_T}})}}\right) - \frac{2}{k\alpha_{j_{N_T}}}\sqrt{s_{j_{N_T}}(t_k + t_{i_1,j_{i_1}\to j_{N_T}})},$$

where $\beta^{\Sigma_1}_{j_{i_1}}(t_k) = \beta^{\Sigma}_{j_{i_1}}(t_k) - \beta_{i_1}$ and $\beta^{\Sigma_1}_{j_{N_T}}(t_k) = \beta^{\Sigma}_{j_{N_T}}(t_k) + \beta_{i_1}$ are the new execution capacities deployed on tasks $j_{i_1}$ and $j_{N_T}$ respectively, and $t_{i_1,j_{i_1}\to j_{N_T}} = d_{j_{i_1}j_{N_T}}/v_{i_1}$ is the travel time of agent $i_1$ from $j_{i_1}$ to $j_{N_T}$.

The total time needed to complete all tasks after redeployment is:

$$T_{i_1,j_{i_1}\to j_{N_T}} = \max\{T_{j_{i_1}}(\beta^{\Sigma}_{j_{i_1}}(t_k) - \beta_{i_1}), T_{j_{N_T}}(\beta^{\Sigma}_{j_{N_T}}(t_k) + \beta_{i_1}), T_{j_{N_T-1}}(\beta^{\Sigma}_{j_{N_T-1}}(t_k))\}.$$

If redeployment reduces total completion time, i.e.,

$$\min_{i_1\in\mathcal{A}_R(t_k)} T_{i_1,j_{i_1}\to j_{N_T}} \geq T_{j_{N_T}}\left(\beta^{\Sigma}_{j_{N_T}}(t_k)\right)$$

reassign the agents and move to Step 1 for next time step $t_{k+1}$. If not, then no agent should be redeployed. In this case, the actual deployment is already almost optimal.

## IV. Greedy Sparse Control

In [2], authors present various strategies for finding sparse control strategies in order to optimally drive towards a desired outcome. One approach towards sparse control is the greedy sparse control that optimizes instantaneously to:

1) minimize, the number of switchings in control, and
2) maximize, at each switching, the rate of decay of the objective function.

This approach models the scenario where the external policymaker lacks the ability to make future predictions and has to make optimal decisions based on instantaneous configurations. Further, note that, in an effort to achieve a minimal amount of switchings, we also reduced the control effort which has not been taken into consideration so far.

Applications of greedy sparse control can be found in society and nature. In society, several government strategies are modeled according to this. Even in nature, we can draw similarities between how a shepherd dog controls a herd of sheep and the greedy sparse control. Hence, another name for this control strategy is the "shepherd-dog strategy".

The basic idea boils down to focusing the entire control effort on the most "stubborn", and once that agent reaches the objective, act on the next most "stubborn". In our case, it implies assigning all the drones to the biggest fire, controlling it in minimal time, and then moving on to the next. While this control strategy seems trivial, note that it does not rely on any fire model to predict the future development of the fire point area and works based on only instantaneous feedback.

## V. Auction Algorithm

This study delves into an auction algorithm employed for solving the multiagent dynamic task allocation problem. Notably, this algorithm demonstrates both low computational complexity and practicality [3]. Its widespread usage across various optimization problems prompts us to juxtapose it with the proposed dynamic task assignment algorithm. The auction algorithm, functioning as a market mechanism governed by specific rules, involves buyers bidding to effectively allocate resources . In this context, auction goods correspond to target tasks, while the agents issuing task information and executing the tasks act as sellers and buyers respectively. An auctioneer facilitates the process by disseminating auction information, while bidding agents use a designated strategy (or revenue function) to bid .

Given the time-varying nature of the fire point model established in this article, employing existing auction algorithms might result in certain tasks not meeting the required execution capacity. Consequently, these tasks may not be completed by all agents. To ensure the completion of all tasks and maximize the utilization of agent resources, a multistage global auction algorithm is devised. This involves the design of an appropriate revenue function tailored to the model, with optimized parameters aimed at minimizing the overall completion time.

The revenue function as follows

$$E_{\text{ij}}(t) = k_1\alpha_j\beta_i\sqrt{s_j(t+t_{\text{ij}})} - 0.1k_2t_{\text{ij}} = E^1_{\text{ij}}(t) - E^2_{\text{ij}}(t)$$

where, $E_{ij}(t_k)$ is the agent i to obtain the revenue when it selects task j as the target at t , k1 and k2 are the weight coefficients, and $E_{1ij}(t) = k_{1ji}(s_j(t + t_{ij}))1/2$ and $E_{2ij}(t) = 0.1k_2t_{ij}$ are the income function and cost function, respectively.

### A. Algorithm

The five steps of the initial auction are described as follows.

1) The auctioneer assumes responsibility for disseminating task-specific information (including parameters such as task load $s_j$, radial velocity $\alpha_j$, and coordinates $(x_j, y_j)$) pertaining to tasks that do not fulfill the

requisite execution capacity, directing this information to the bidding agents.

2) Each unallocated bidding agent undertakes the computation of the projected revenue $E_{ij}$ associated with the execution of individual tasks. Subsequently, these agents strategically select tasks expected to yield the highest revenue, informing their bidding choices.
3) The auctioneer systematically accumulates the bidding information and iteratively identifies the winning bid agent for each task, based on the highest revenue generated from task execution.
4) Evaluate whether all tasks satisfy the necessary execution capacity; if affirmative, proceed to Step 5); if not, revert to Step 1 to reassess and redistribute task information.
5) The remaining unallocated agents, including the auctioneer, strategically prioritize tasks based on their potential to generate the highest revenue.

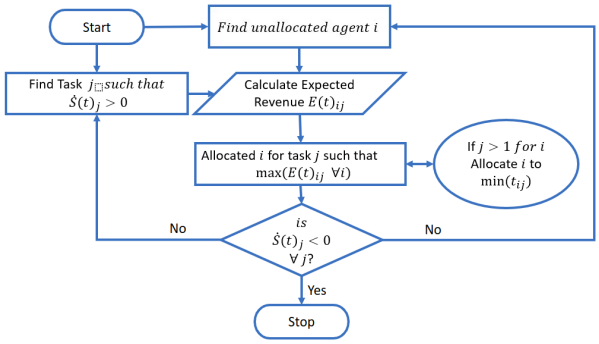


Fig. 6: Initial Static Auction Flow chart

As the output generated by the auction algorithm lacks a guarantee of optimality, it necessitates dynamic adjustments. To address this requirement, a dynamic auction system is formulated specifically for real-time execution. The dynamic auction encompasses five sequential steps.

1) At each time step, the auctioneer computes $T^*(t_k)$ as per equation (33) and populates the set $AR(t_k)$ with selected candidate agents. The selection method for these candidate agents mirrors that of the proposed dynamic task assignment algorithm.
2) The auctioneer calculates the time $T_j(\beta\Sigma_j(t_k))$ required to complete task $j(\forall j \in \Lambda T)$ at $t_k$ based on the assigned agents. Subsequently, the auctioneer disseminates the information of task $j_{NT}$ (the task projected to take the longest time) to the bidding agent $i$, where $i \in AR(t_k)$.
3) Each unallocated bidding agent $i$, where $i \in AR(t_k)$, computes the revenues $E_{ij_{NT}}(t_k)$ and $E_{iji}(t_k)$ (with agent $i$ working on task $j_i(t_k)$ at $t_k$) and compares their magnitudes. If $E_{ij_{NT}}(t_k) > E_{iji}(t_k)$, agent $i$ opts to bid for task $j_{NT}$; otherwise, it remains in $AR(t_k)$.
4) The auctioneer collects the bidding information and selects the winning bid agent that offers the largest reduction in total completion time.
5) The auctioneer recalculates the time $\tilde{T}_j(\beta\Sigma_j(t_k))$ and compares it with $T_j(\beta\Sigma_j(t_k))$. If $\tilde{T}_j(\beta\Sigma_j(t_k)) < T_j(\beta\Sigma_j(t_k))$, the process reverts to Step 2. Otherwise, it proceeds to Step 6.
6) The remaining unallocated agents in $AR(t_k)$ continue their work at $t_k$.

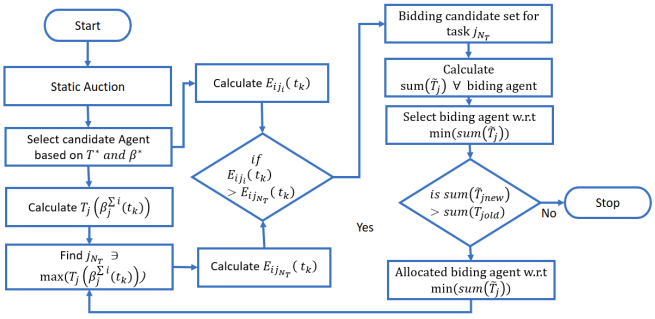


Fig. 7: Dynamic auction with Initial Static Auction Flow chart

## VI. Simulation Results

### A. Example 1

There are two fire points located at $(80, 100)$ units and $(120, 100)$ units respectively, characterized by $s_1(0) = 2.1$ unit$^2$, $s_2(0) = 1.2$ unit$^2$, $\alpha_1 = 1.2$ unit/sec and $\alpha_2 = 1.5$ unit/sec. Parameter $k$ is set as 0.6 and the sampling time is taken to be 0.01 sec. To extinguish these fires 10 agents are deployed in the arena with the following characteristics ($(x_i, y_i)$ are in units, $v_i$ is in unit/sec and $\beta_i$ is in unit$^2$/sec):

TABLE I: Agent characteristics for Example 1

| $i$ | $(x_i, y_i)$ | $\beta_i$ | $v_i$ | $i$ | $(x_i, y_i)$ | $\beta_i$ | $v_i$ |
|---|---|---|---|---|---|---|---|
| 1 | (80, 90) | 0.20 | 60 | 6 | (90, 110) | 0.20 | 85 |
| 2 | (70, 90) | 0.20 | 85 | 7 | (90, 100) | 0.60 | 60 |
| 3 | (70, 100) | 0.20 | 60 | 8 | (120, 90) | 0.55 | 80 |
| 4 | (70, 110) | 0.20 | 85 | 9 | (130, 100) | 1.00 | 10 |
| 5 | (80, 110) | 0.20 | 60 | 10 | (120, 110) | 0.70 | 60 |

*1) Task Allocation Without Dynamic Deployment:* The initial deployment is:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

which implies that agents 1–7 are assigned to fire point 1, and agents 8–10 to fire point 2. The execution capacity working on the tasks at each time step and the evolution of the fire point areas with time are shown in figure 8.

*2) Task Allocation With Dynamic Deployment:* After the initial deployment, the algorithm at each time step compares the current execution capacity deployed at each fire point with the optimal solution and redeploys agents as and when needed. The execution capacity working on the tasks at each time step and the evolution of the fire point areas with time are shown in figure 9.

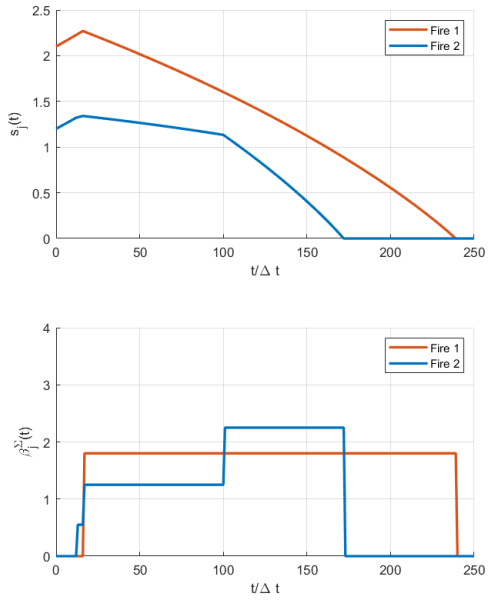


Fig. 8: Example 1: Without Dynamic Deployment.

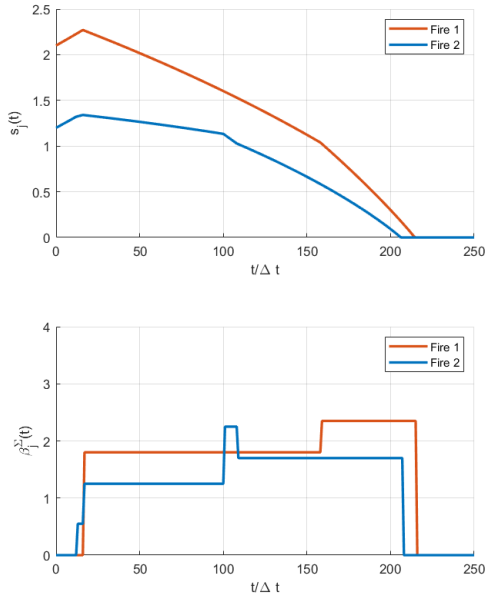


Fig. 9: Example 1: With Dynamic Deployment.

*3) Greedy Sparse Control:* The initial deployment is:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which implies all the agents are assigned to fire point 1. After $t/\Delta t = 102$, the deployment is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which implies all the agents are assigned to fire point 2. The execution capacity working on the tasks at each time step and the evolution of the fire point areas with time are shown in figure 10.

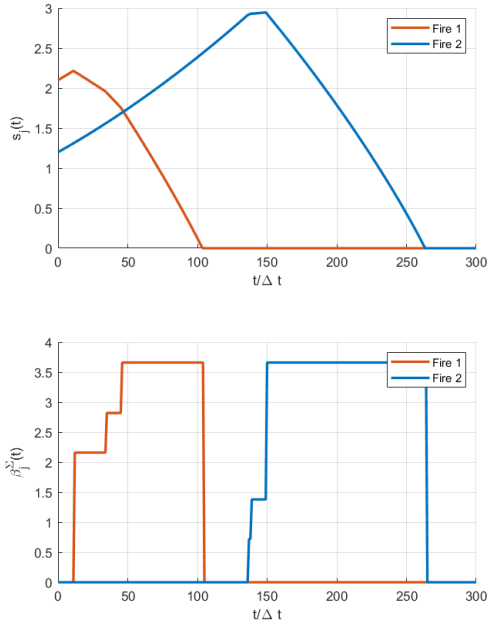


Fig. 10: Example 1: Greedy Sparse Control.

## B. *Example 2*

There are two fire points located at $(80, 100)$ units and $(120, 100)$ units respectively, characterized by $s_1(0) = 2.1$ unit$^2$, $s_2(0) = 1.2$ unit$^2$, $\alpha_1 = 1.5$ unit/sec and $\alpha_2 = 1.8$ unit/sec. Parameter $k$ is set as 0.6 and the sampling time is taken to be $0.01$ secs. In this example, we assume that all the 10 agents start in the same area, such as a fire station. The characteristics of the agents are ($(x_i, y_i)$ are in units, $v_i$ is in unit/sec and $\beta_i$ is in unit$^2$/sec):

TABLE II: Agent characteristics for Example 2

| $i$ | $(x_i, y_i)$ | $\beta_i$ | $v_i$ | $i$ | $(x_i, y_i)$ | $\beta_i$ | $v_i$ |
|---|---|---|---|---|---|---|---|
| 1 | (90, 100) | 0.45 | 100 | 6 | (90, 100) | 0.40 | 80 |
| 2 | (90, 100) | 0.40 | 100 | 7 | (90, 100) | 0.40 | 80 |
| 3 | (90, 100) | 0.40 | 80 | 8 | (90, 100) | 0.40 | 80 |
| 4 | (90, 100) | 0.40 | 80 | 9 | (90, 100) | 0.40 | 80 |
| 5 | (90, 100) | 0.40 | 80 | 10 | (90, 100) | 0.40 | 80 |

*1) Task Allocation Without Dynamic Deployment:* The initial deployment is:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

which implies that agents 1–4 and 10 are assigned to fire point 1, and agents 5–9 to fire point 2. The execution capacity working on the tasks at each time step and the evolution of the fire point areas with time are shown in figure 11.

*2) Task Allocation With Dynamic Deployment:* After the initial deployment, the algorithm at each time step compares the current execution capacity deployed at each fire point with the optimal solution and redeploys agents as and when needed. The execution capacity working on the tasks at each time step and the evolution of the fire point areas with time are shown in figure 12

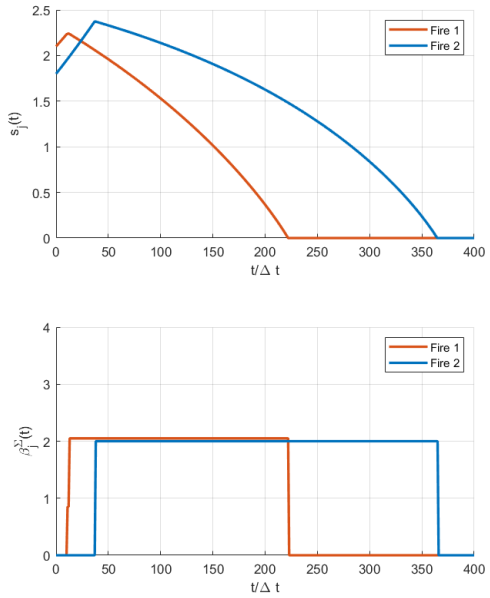


Fig. 11: Example 2: Without Dynamic Deployment.

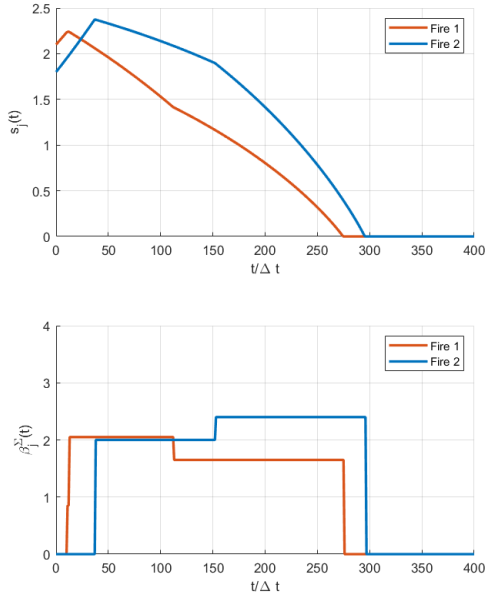


Fig. 12: Example 2: With Dynamic Deployment.

## C. Example 3

The number of agents and tasks are 18 and 3, respectively.Parameter $k$ is set to be 0.6. The initial positions of the three fire points are $(80, 120)$, $(40, 40)$, and $(80, 40)$, respectively; they are characterized by $s_1(0) = 1.9$, $s_2(0) = 1.8$, $s_3(0) = 1.15$, $\alpha_1 = 1.1$, $\alpha_2 = 1.2$, and $\alpha_3 = 1.2$. We denote by $\Delta t$ and $T_s$ the sampling period of simulation and the time step of dynamic deployment, respectively. $\Delta t$ is set to be 0.001 and $T_s = 10\Delta t$. We set $k_1 = 0.2$, $k_2 = 0.1$ .following table presents the characteristics of the 18 agents.

| $i$ | $(x_i, y_i)$ | $\beta_i$ | $v_i$ | $i$ | $(x_i, y_i)$ | $\beta_i$ | $v_i$ |
|---|---|---|---|---|---|---|---|
| 1 | $(100, 140)$ | 0.25 | 170 | 10 | $(50, 40)$ | 0.3 | 60 |
| 2 | $(80, 140)$ | 0.2 | 120 | 11 | $(70, 20)$ | 0.2 | 216 |
| 3 | $(100, 70)$ | 0.2 | 324 | 12 | $(30, 20)$ | 0.2 | 135 |
| 4 | $(80, 70)$ | 0.2 | 300 | 13 | $(30, 40)$ | 0.3 | 60 |
| 5 | $(60, 120)$ | 0.2 | 120 | 14 | $(20, 60)$ | 0.2 | 170 |
| 6 | $(60, 130)$ | 0.2 | 135 | 15 | $(80, 50)$ | 0.6 | 60 |
| 7 | $(90, 120)$ | 0.5 | 60 | 16 | $(80, 30)$ | 0.6 | 80 |
| 8 | $(40, 60)$ | 0.4 | 120 | 17 | $(90, 40)$ | 1.0 | 10 |
| 9 | $(50, 50)$ | 0.2 | 85 | 18 | $(90, 50)$ | 0.2 | 85 |

Fig. 13: Agent Parameters.

The results are as follows

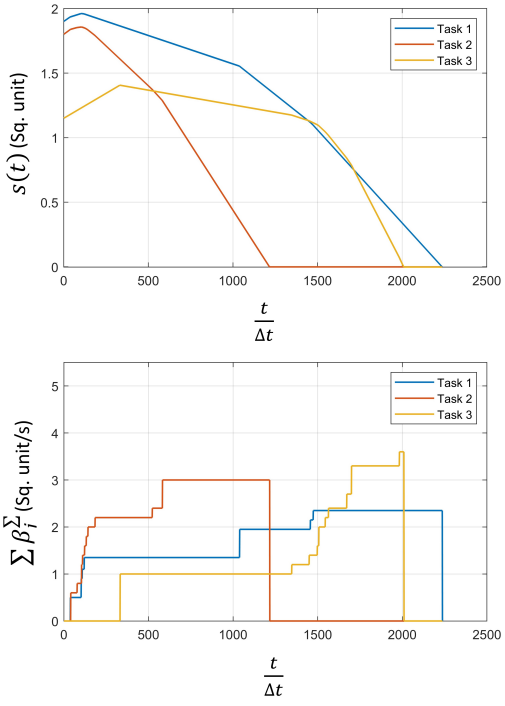


Fig. 14: Static Auction.

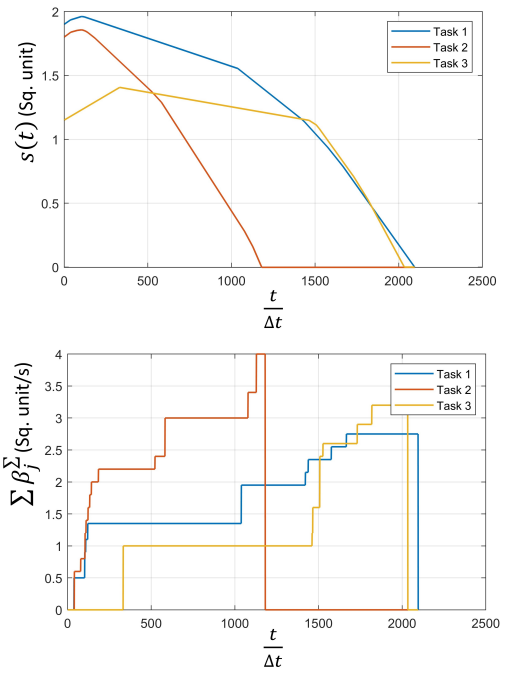


Fig. 15: Static Auction initialization with dynamic auction

A) **Static Auction:** The task 1, task 2 and task 3 complete the tasks in $2236\Delta t$, $1215\Delta t$ and $2008\Delta t$, respectively. The total completion time is $2236\Delta t$.

B) **Dynamic Auction with static initialization:** The task 1, task 2 and task 3 complete the tasks in $2109\Delta t$, $1181\Delta t$ and $2033\Delta t$, respectively. The total completion time is $2109\Delta t$ sec. The dynamic allocation takes less time for fire execution compared to static allocation but at a higher computational cost. The computational cost increases exponentially with the increase in the number of agents and the number of tasks. The weight parameters $k_1$ and $k_2$ play an important role in calculating the optimal solution. The radius of the fire also makes a difference in the completion time, which was not taken into account.

## VII. Quadrotor Trajectory Simulation

The simulation of trajectory of the quadrotor was done using *MATLAB-Simulink* UAV Toolbox. The dynamic deployment explained in the previous sections gives the task matrix which varies with time based on the optimal deployment required. Guidance model from*Simulink* was used to generate trajectories for the quadrotor from their initial locations to the deployed locations. In this simulation we have considered two fire points and ten agents.

### A. Waypoint Design

The trajectory simulation for fire fighting in this project was done using waypoint follower tool from Simulink. To make the quadrotor follow a polygonal trajectory over a fire point, we considered vertices of a square as the waypoint which the quadrotor is expected to follow, the construct of the same is shown in Figure 16 where $(x, y)_i$ are the locations of the fire points $i \in [1, N_T]$, d is the offset distance and h is the reference altitude over the fire point. The waypoints are designed such that d and h are unique for each agent to have higher efficiency in fire fighting and minimize chances of collision, it can be noted that no separate collision avoidance method is used see

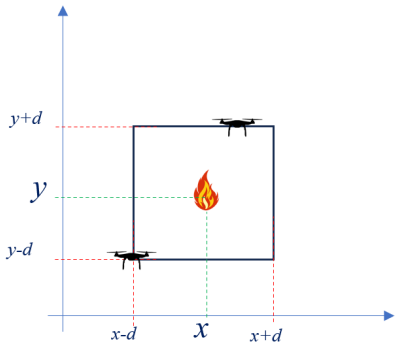


Fig. 16: Waypoint reference for agent i

### B. Simulink Quadrotor Guidance Block

All the quadrotor dynamics are simulated using the simulink quadrotor dynamics block available in UAV-toolbox. This block is called as *Guidance Model* which is a reduced order model of the UAV, quadrotor in our case. The inputs to the block are the control commands *viz.* Roll, pitch, yawrate and thrust commands and the environment which for the case of a quadrotor is gravity. The outputs of the block are listed in Table III. The block is as shown in Figure 17.

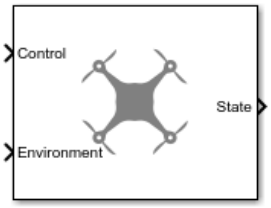


Fig. 17: Simulink - Guidance model of Quadrotor

TABLE III: Output states- Quadrotor block

| State | Element |
|---|---|
| WorldPosition | [x,y,z]] |
| WorldVelocity | [$v_x, v_y, v_z$] |
| EulerZYX | [psi phi theta] |
| BodyAngularRateRPY | [r p q] |
| Thrust | F |

### C. Simulink Waypoint Follower

The waypoint follower in simulink UAV toolbox generates commands for the quadrotor to follow the desired set of waypoints, Figure 18 shows the Simulink waypoint follower block. The inputs to the block are - current position of the quadrotor, the set of waypoints and lookahead distance. Based on the lookahead distance input the lookahead point

output is calculated, another output is the desired course which gives the heading direction command. The block also has a status flag which returns 1 when the quadrotor has navigated all the waypoints. In this project we have used a state machine as shown in Figure 19 which gives a new set of waypoints minimally offset in height but identical planar co-ordinates once set-one is completely navigated by the quadrotor. This way the quadrotor continuously keeps getting waypoints to trace.

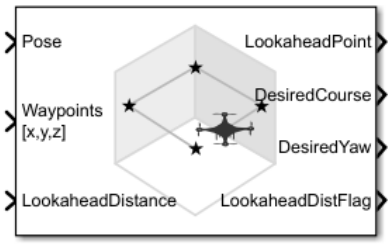


Fig. 18: Simulink- Waypoint Follower

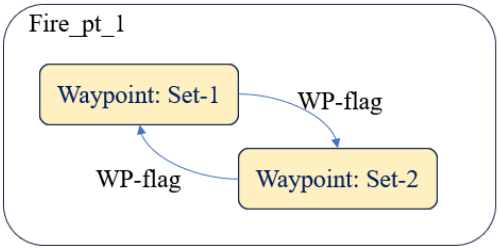


Fig. 19: Waypoint-state machine

### D. Tracking redeployment of agents

The task matrix keeps track of the redeployment of the agents in case any of the agents based on optimal assignment of agents to particular task are redeployed. In this project we take series stack of the task matrix generated from the dynamic deployment algorithm explained in previous sections and keep on assigning the respective agents to the respective fire point. To realize this we have used a state machine construct which starts from the initial deployment and then keeps on updating the task matrix every 10 seconds to check for occurrence of redeployment in the task matrix series. The implemented state machine is shown in Figure 20, where B is the task matrix.

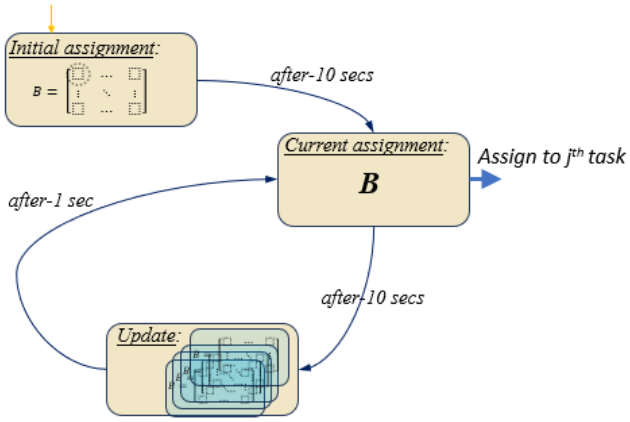


Fig. 20: State-machine for dynamic task allocation

### E. Trajectories of the quadrotors

The quadrotors start at the initial locations as stated in Table I and the initial deployment matrix as:

$$B = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Figure 21 shows the trajectories of the quadrotors from initial positions to the deployed targets and Figure 22 and Figure 23 shows the top and side view of the trajectories for the entire series of task matrix respectively. It can be observed that agent-8 get redeployed to another fire point. Also, unique setting for altitude h and offset distance d for each agent creates manageable separation between all the trajectories.

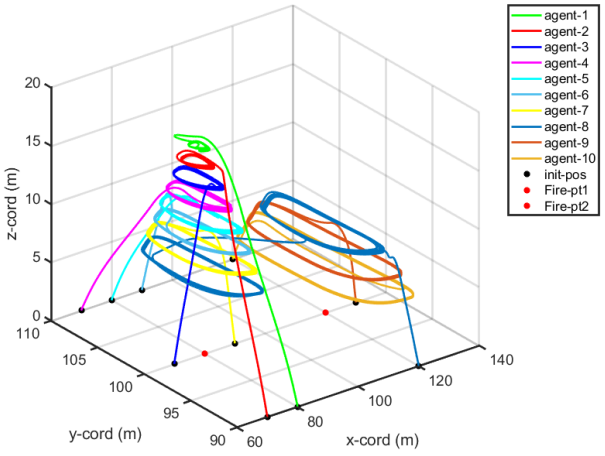


Fig. 21: Plot of quadrotor trajectories

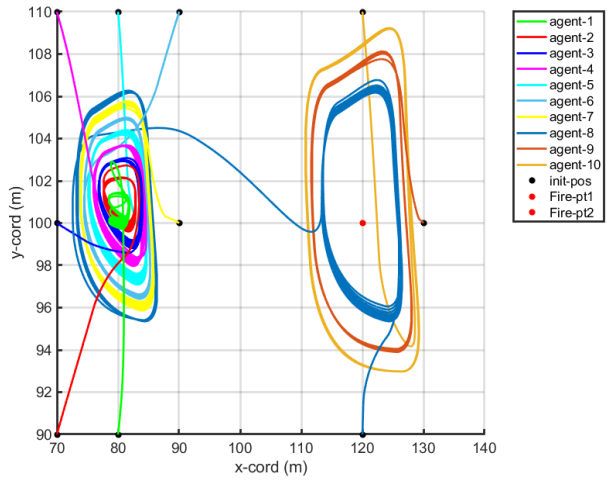


Fig. 22: Top-view plot of quadrotor trajectories

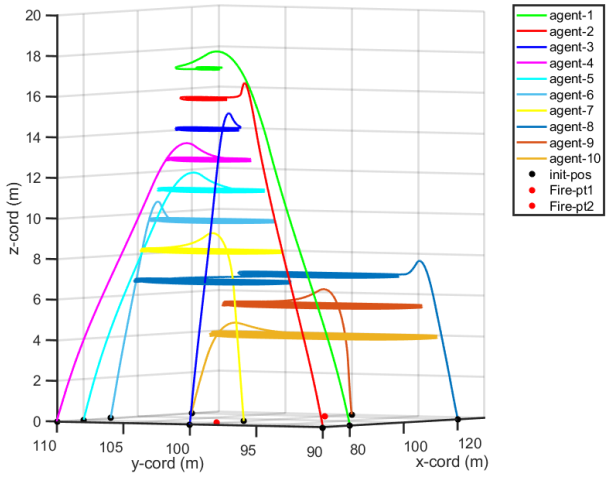


Fig. 23: Side-view plot of quadrotor trajectories

## VIII. Discussion

### A. Execution Capacity

We compare the execution capacity deployed at each fire point vs the minimum necessary execution capacity such that the fire point area does not grow. The results are illustrated in figures 24 and 25 which compare the execution capacities in Examples 1 and 2 respectively. Note that as long as the deployed execution capacity is lower than the necessary execution capacity, the fire point area grows, and as the difference between them increases, the area decays faster. This is in line with the fire model used discussed above.

### B. Greedy Sparse Control

Despite the advantage of not relying on any fire model for future predictions, it can be seen from figures 9 and 10 that the Dynamic deployment strategy clearly has the upper hand. Both in terms of time of task completion and number of agents redeployed, dynamic deployment strategy is optimal.

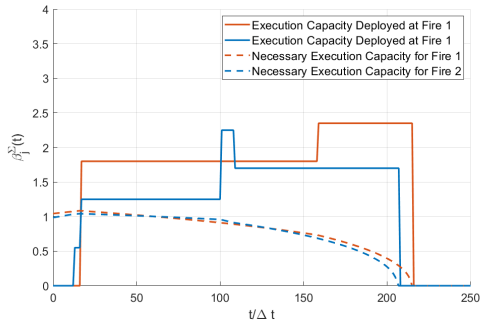


Fig. 24: Execution capacity comparison for Example 1.

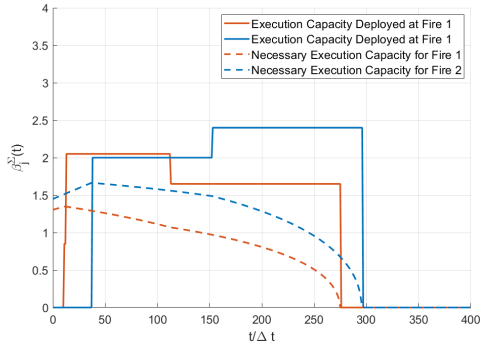


Fig. 25: Execution capacity comparison for Example 2.

However, remember that the inspiration for greedy sparse control came from the shepherd-dog strategy and has been readily used in government policies, where the travel speed between tasks is much higher compared to the growth rate of the tasks. Hence, we investigated the dependence of the task completion time on the average drone speeds. It can be seen from figure 26, that as the average drone speed increases, the task completion time goes down more rapidly for greedy sparse control and eventually at a much higher speed ($>$ 23.7 unit/sec), it even shows better results than dynamic deployment.

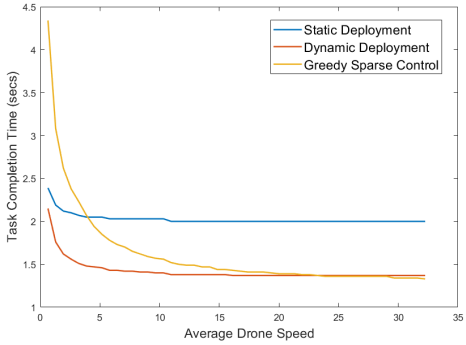


Fig. 26: Task completion time vs Average drone speed.

## IX. Conclusion

In this project we have simulated a task assignment problem for mitigation of forest firefighting. The main aim of the project is to optimize the time required for extinguishing the fire by optimally allocating the fire fighting agents among the tasks. To arrive at optimal allocation we used initial deployment, dynamic redeployment, greedy sparse control and auction control methods. Initial deployment assigns agents based on the size of the fire and agent's distance from the fire hence cannot be optimal as all the agents have different extinguishing capacity. Dynamic redeployment scheme provide optimal task assignment policy as based on the state of the fire and availability of extra capacity for a particular task, on the other hand greedy sparse control demands less redeployment of agents and does not rely on the fire model which is more practical. Based on the requirement in terms of sensing of fire locations or priority of optimization of time or number of redeployments, we can select appropriate algorithm.

## References


[1] J. Chen, Y. Guo, Z. Qiu, B. Xin, Q.-S. Jia, and W. Gui, "Multiagent dynamic task assignment based on forest fire point model," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 2, pp. 833–849, 2022.

[2] M. Caponigro, M. Fornasier, B. Piccoli, and E. Trélat, "Sparse stabilization and control of alignment models," 2014.