La Sapienza University

Department of Computer Science

Autonomous Networking

Report HomeWork 3

Authors:
Giordano Dionisi
1834919
Mattia Lisi 1709782
Michele Spina 1711821

Supervisor:
Prof. Andrea Coletta
Co-Supervisor:
Prof.ssa Gaia Maselli

January 8, 2022

Dipartimento
di Informatica

Sapienza
Università di Roma

Contents

1 Formalize The Problem

We have this scenario:

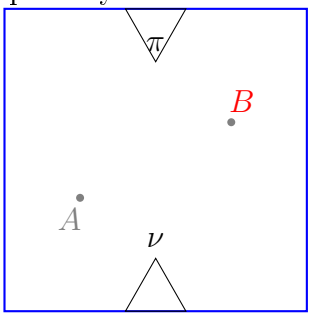
- Two Depots:

$$\nu = Depot - 1$$
$$\pi = Depot - 2$$

- η drones
- Each drone θ has $\Gamma(\theta)$ packets

We want build a routing algorithm to allow drones to communicate each other for the purpose to get arrive as much as possible Γ to the ν and π.

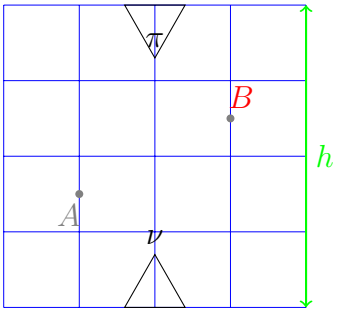
Graphically the Scenario is:



Where:
A and B are two *drones.*

2 Solution

First of all we divide the scenario in *grids:*



Where:

- $h \Rightarrow$ Lenght of one side of the square of grid.

After we need to define States2.1, Actions2.2 and Reward2.2.2

2.1 States

Each cell χ of the grid built is a state for the system. So each θ isn't a state, but it takes the state of the particular χ where it stays.

Formally we have:

$$\omega = 4$$

$$\Lambda = 4$$

Where:

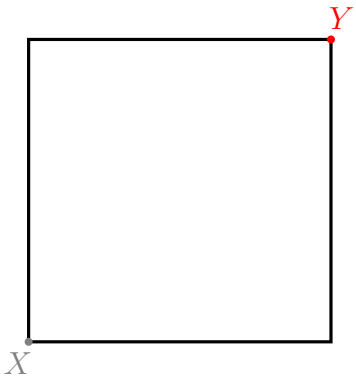
- ω: Number of cells on X *axis*
- Λ: Number of cells on Y *axis*

Ω is the total amount of cells in the grid:

$$\Omega = \omega * \Lambda$$

Let's define:

- Each χ is defined by a left bottom point X and a right upper point Y, so each χ is defined by a bounding-box.



Each X and Y for each χ are defined by:

```
for i=0 to h:

    for j=0 to h:

        χ[X] = [i; j]
        χ[Y] = [i + (h/ω); j + (h/ω)]
        j = j + h/ω

    i = i + h/ω
```

Where:

- $\chi[X] \Rightarrow$ coordinate X of χ
- $\chi[Y] \Rightarrow$ coordinate Y of χ

Each State of System $\equiv$ Each χ

Brief Recap

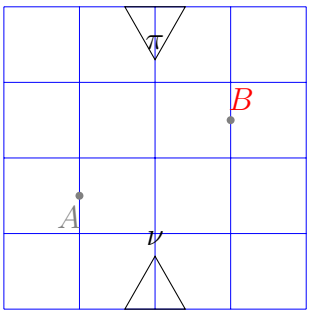
We divide the area (h,h) of grid with a regular tessellation[1]. Each χ of the grid delimits an internal space in the grid that we will use as one state for the θ.

When a θ stays in a particular χ, it takes its particular state.

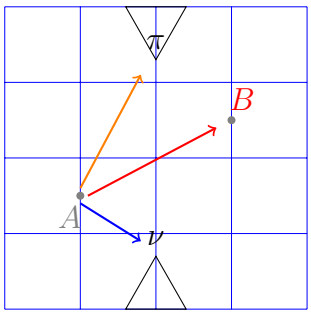
[1] Ω cells.

2.2 Actions

The following:



Becomes this:



Let's assume we are A[2]
Let's define:

- $N(A)$ is the set of neighbors for A[3].

In each moment if A has a packet ψ, then A can perform different actions:

- A can mantain ψ;
- A can go to ν;
- A can go to π;
- A can pass ψ to a $B \in N(A)$.

The particular action β chosen depends on:

$$\beta = \begin{cases} \text{Q-Learning}, & \text{if } \ni \leq 1 - \epsilon \\ \text{Modified MGEO}, & \text{otherwise} \end{cases}$$

Where $\ni$ is a random value[4].

We based our algorithm on *Epsilon-Greedy Approach.*

First of all let's define ϵ.

[2]This does not make you lose generality.
[3]Nodes in its communication range.
[4]This is explained in 2.3.

2.2.1 Epsilon

ϵ is defined by the following function:

$$\epsilon[\theta] = \frac{1}{\kappa}$$

- ϵ is computed locally for each θ.
- κ is variable that represents the number of times that θ has updated its Q-Sets.

$$\kappa = 1, 2, 3, \ldots, +\infty$$

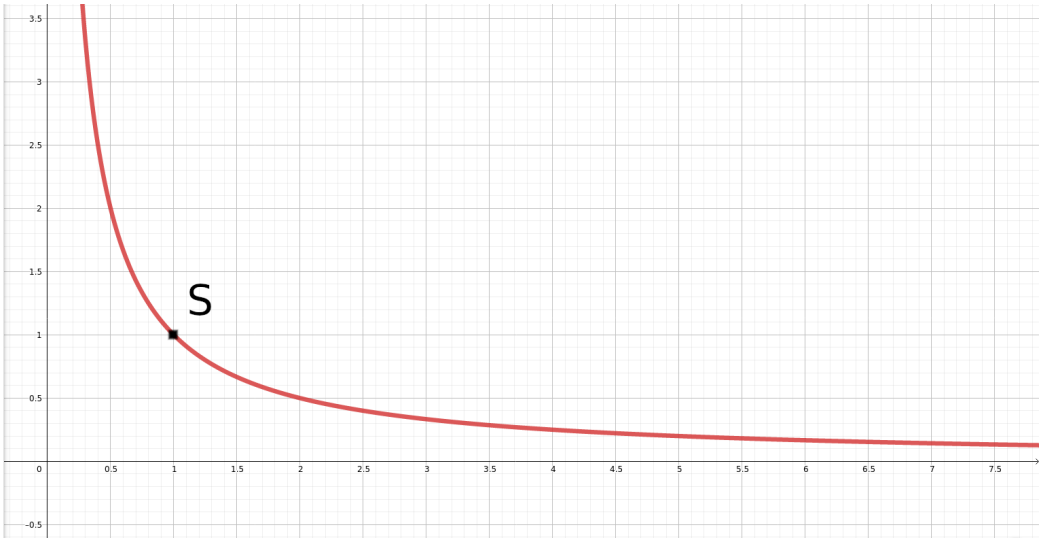


Figure 1: Plot $\frac{1}{\kappa}$

As can be seen in Figure 1, ϵ function, defined in $[1, +\infty]$, is a continuous decreasing function with initial value $S = 1$ and it tends to zero for high value of κ.

In this way the probability of using *Q-learning* approach will be directly proportional to the amount of information received (κ), so using it more when it's more effective. At the same time when the algorithm haven't some information, then the use of *Modified MGEO* approach is almost a certain event.

Just for Completeness

Algorithm is based on **Optimistic Initial Value** μ**.** Sperimentally best value is:

$$\mu = 10$$

Indeed ***initially*** it's privileged *higher* ***exploration*** *than* ***exploitation,*** so μ must be enough higher than upper value of Reward2.2.2.

2.2.2 Reward

The Reward λ is related to the delayed outcome ι of the specific packet ψ:

$$\lambda = \begin{cases} -1, & \text{if } \iota = -1 \\ 1 + \frac{\sigma - \xi}{\sigma}, & \text{otherwise} \end{cases}$$

Where:

- σ: *Time-To-Live* for ψ
- ξ: Delay of ψ, so:

$$\xi \in [0, \sigma]$$

So:

$$\Rightarrow \lambda \in [-1] \cup [1, \ldots, 2]$$

Explanation Idea:

- If ψ arrive to ν or π with $\xi > \sigma$, then λ, for every ***responsible*** θ, is a fixed negative value;
- Otherwise: fewer ξ and higher λ.

Remember: It's better a small ξ than large ξ.

The contribute of ξ is:

$$\frac{\sigma - \xi}{\sigma} \in [0, 1]$$

Remember: λ is a function that returns a value that it gives for ψ to **EVERY** responsible θ.

2.3 Continue: Actions

- First of all[5]:

  1. If $\Gamma(\theta) \geq 1$, then $\forall \psi \in \Gamma(\theta)$[6]:

$$\begin{cases} \psi \text{ to } \theta', & \text{if } \exists \theta' \in N(\theta) \mid \theta' \text{ is going to } \nu \textbf{ or } \pi \\ \psi \text{ to } \theta', & \text{otherwise if } \exists \theta' \in N(\theta) \mid mustGoBack(\theta') = True \\ GotoDepot, & \text{otherwise if } mustGoBack(\theta') = True \\ \psi \text{ to } \theta', & \text{otherwise if } \exists \theta' \in N(\theta) \mid Q[None](\theta') = \varphi \end{cases}$$

     Where:

     - ψ *to* θ' means that θ passes ψ to θ'
     - $mustGoBack(\theta') = True$ means that θ' is going to deliver $\Gamma(\theta')$ to π or $\nu \rightarrow$ It's an internal state for the drone:

$$mustGoBack = \begin{cases} True, & \text{if } \theta' \textit{ is approaching to } \pi \textit{ or } \nu \ \wedge \ \Gamma(\theta') \geq 1 \\ False, & \text{otherwise} \end{cases}$$

     - $Q[\varpi](\theta')$: Value for Q-Learning to do action ϖ for θ'
     - $\varphi = \theta' \mid \theta' \in N(\theta) \wedge Q[None](\theta') \geq Q[None](\theta''), \forall \theta'' \in N(\theta) \wedge \Gamma(\theta') \geq 1 \wedge \Gamma(\theta'') \geq 1$

     By adding the third control it's possible to make packets converge on a single θ, reducing the number of returns to depot required and saving battery without affecting performance.

- In case 1 - ϵ:

  Let's Define:

  $$\Phi = \{x | x \in \{Depot, None, Pass\} \wedge \forall y \in \{Depot, None, Pass\}, Q[x](\theta) \geq Q[y](\theta)\}$$

  Where Φ is the best choice action to do for *Q-Learning*

[5] This part was "Future Developments" in HW2

[6] **Remember:** Γ is the amount of packets for a drone θ

1. If $N(\theta) = \emptyset$:

$$\beta = \begin{cases} None, & \text{if } \Phi = \{Pass\} \\ None, & \text{otherwise if } \Phi = \{None\} \\ Depot, & \text{otherwise if } \Phi = \{Depot\} \\ Depot, & \text{otherwise if } \Phi = \{Pass, Depot\} \\ None, & \text{otherwise} \end{cases}$$

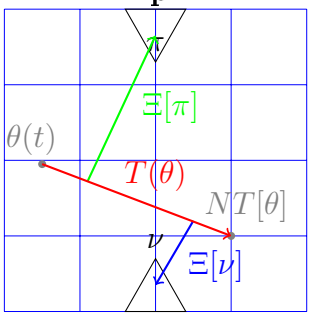
If $\Phi = \{Depot\}$ then:

$$\rho = \begin{cases} \nu, & \text{if } Q(\nu) > Q(\pi) \\ \pi, & \text{otherwise if } Q(\pi) > Q(\nu) \\ \nu, & \text{otherwise if } \Xi[\nu] \leq \Xi[\pi] \\ \pi, & \text{otherwise} \end{cases}$$

Where:

- $Q(\rho)$: Another Q-Set where the Q-Value represents the learning choice of ν **or** π to understand which is best $\rightarrow$ Respectively we have $Q(\nu)$ and $Q(\pi)$.
- $\Xi[x]$ is the euclidian distance between the $T(\theta)$ to depot x, ***Where:***
  - $T(\theta)$ is the trajectory from $\theta(t)$ and NT$[\theta]$
  - $\theta(t)$ is the position of θ at time t
  - $NT[\theta]$ is the position of the next target of θ

  For Example:



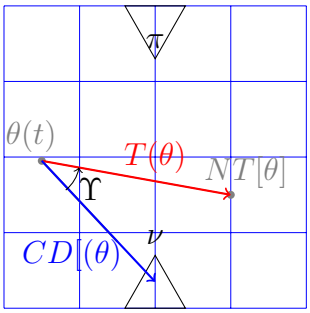
We update the *mustGoBack* value:

$$mustGoBack = \begin{cases} \text{True}, & \text{if } \Upsilon \leq 90° \\ \text{False}. & \text{otherwise} \end{cases}$$

Where:

- Υ is the angle from $T(\theta)$ and $CD[\theta]$
- $CD[\theta]$ is the segment between θ and ν

For Example **Υ is:**



And:

$$\beta = \begin{cases} Check\ NextIteration\ mustGoBack, & \text{if } mustGoBack = True \\ Depot, & \text{otherwise} \end{cases}$$

2. If $|N(\theta)| \geq 1$

$$\beta = \begin{cases} None, & \text{if } \Phi = None \\ \psi \text{ to } \theta', & \text{otherwise if } \Phi = Pass \\ Depot, & \text{otherwise if } \Phi = Depot \\ Depot, & \text{otherwise if } \Phi = \{Pass, Depot\} \\ None, & \text{otherwise} \end{cases}$$

Where[7]**:**

- $\theta' \mid \theta' \in N(\theta) \wedge \forall \theta'' \in N(\theta),\ v^*(\theta') = \mathbf{max}(v^*(\theta''))$

[7] Q-Learning purpose.

If $\Phi = \{Depot\}$ then:

$$\rho = \begin{cases} \nu, & \text{if } Q(\nu) > Q(\pi) \\ \pi, & \text{otherwise if } Q(\pi) > Q(\nu) \\ \nu, & \text{otherwise if } \Xi[\nu] \leq \Xi[\pi] \\ \pi, & \text{otherwise} \end{cases}$$

We update the usually mustGoBack value:

$$mustGoBack = \begin{cases} \text{True}, & \text{if } \Upsilon \leq 90° \\ \text{False}. & \text{otherwise} \end{cases}$$

And:

$$\beta = \begin{cases} Check\ NextIteration\ mustGoBack, & \text{if } mustGoBack = True \\ Depot, & \text{otherwise} \end{cases}$$

- In the ϵ case we have *modified* ***MGEO*** *algorithm:*

1. If $N(\theta) = \emptyset$:

$$\rho = \begin{cases} \nu, & \text{if } \Xi[\nu] \leq \Xi[\pi] \\ \pi, & \text{otherwise if } \Xi[\pi] > \Xi[\nu] \end{cases}$$

We choose a ρ and set:

$$mustGoBack = \begin{cases} \text{True}, & \text{if } \Upsilon \leq 90° \\ \text{False}. & \text{otherwise} \end{cases}$$

And:

$$\beta = \begin{cases} Check\ NextIteration\ mustGoBack, & \text{if } mustGoBack = True \\ Depot, & \text{otherwise} \end{cases}$$

2. If $|N(\theta)| \geq 1$:
   $\forall \theta' \in N(\theta) \Rightarrow$

$$\rho[\theta'] = \begin{cases} \nu, & \text{if } \mathbf{dist}(NT[\theta'], \nu) \leq \mathbf{dist}(NT[\theta'], \pi) \\ \pi, & \text{otherwise} \end{cases}$$

   Where:
   - $\rho[\theta'] \Rightarrow$ Nearest ρ for θ'

   θ passes ψ to ϑ, where:

   $\forall \theta' \in N(\theta), \exists \vartheta \in N(\theta) \mid \mathbf{dist}(NT[\vartheta], \rho[\vartheta]) = \mathbf{min}(\mathbf{dist}(NT[\theta'], \rho[\theta']))$

   In this case we only update Q-Sets, NO EXPLOITATION

Obviously we always update Q-Sets for any action that we do.

2.4 Alpha/Gamma

Important things are Q-Learning's parameters:

$$\alpha = 0.5$$

$$\gamma = 0.2$$

Where:

- α is the learning-rate. If λ function[8] or transition function is stochastic (random), then α should change over time. α should $\approx$ zero at ∞.
- γ represents future λ. It can affect learning quite a bit. If $\gamma = 1$, the agent values future reward JUST AS MUCH as current reward. If an agent does something good this is JUST AS VALUABLE as doing this action directly. So learning doesn't work at that well at high γ values.

  Conversely, $\gamma = 0$ will cause the agent to only value immediate rewards, which only works with very detailed λ functions.

Sperimentally, for our scenarios, we want that α is balanced between the past and the future and we also want the value of γ that it is small, because the scenario change frequently. So we need to do an high weight to the rewards that we will receive compared to the past learnings.

[8] **Remember:** It is the Reward Function.

3 Conclusions: Performances in the Two Possible Scenarios

We have tested two possible scenarios[9]:

- Repeated Routes *Rep*;
- Randomic Routes *Rnd*.

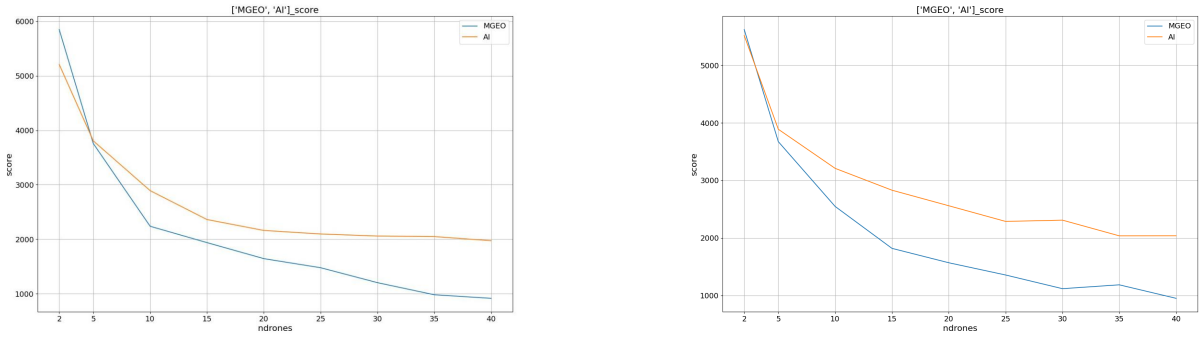


Figure 2: Plot Score *Rep* vs *Rnd*

Figure 2: Score in *Rep* is less than *Rnd*, but both of them have same convergence.

[9] To see differences with learning.

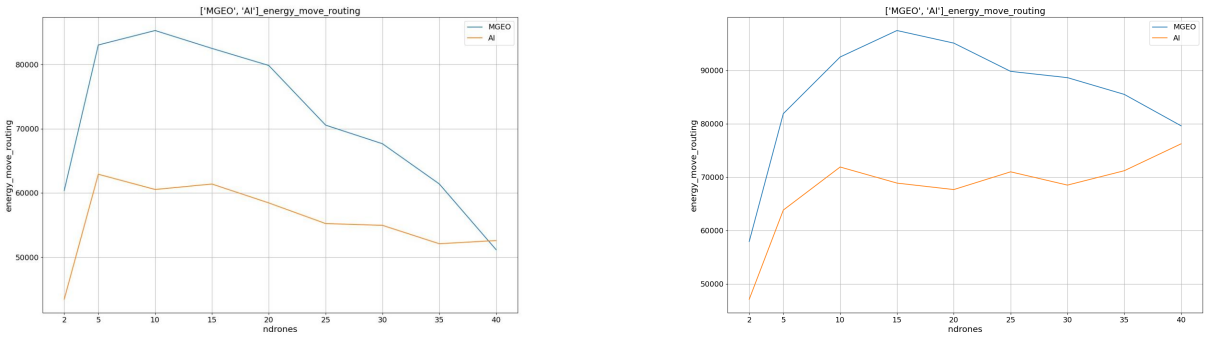


Figure 3: Plot Energy Consumed *Rep* vs *Rnd*

Figure 3: In *Rep* we exploit more the learning → So energy consumed is less, because it is avoided many times to go to π or ν.

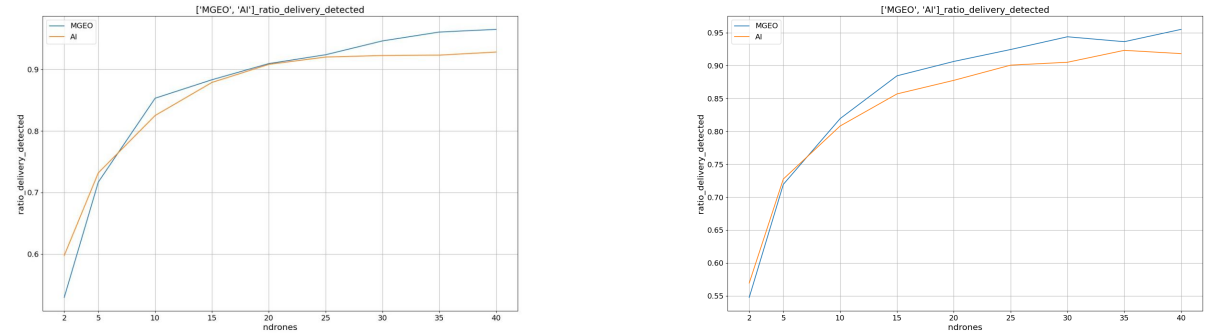


Figure 4: Plot Ratio for Delivery Detected *Rep* vs *Rnd*

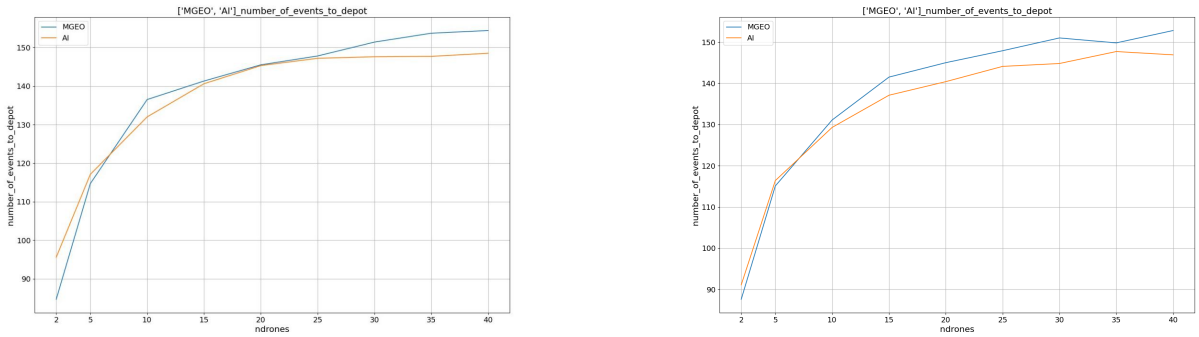


Figure 5: Plot Number of Events to Depot *Rep* vs *Rnd*

Figure 4 & Figure 5::Ratio of Delivery and Number of Events to Depot is pratically the same in both scenarios and also for MGEO and AI_Routing[10].

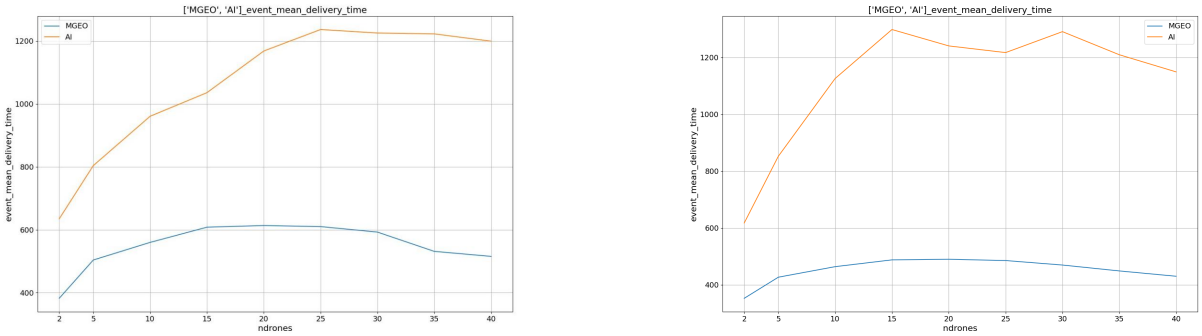


Figure 6: Plot Mean Delivery Time *Rep* vs *Rnd*

Figure 6: **Important:** We deliver a lot of Γ and for this reason the score isn't high, but Mean Time is a high $\rightarrow$ For each ψ we have a wait 3 times higher than MGEO.

[10] 5 is the same of 4, because both of them are dependent values.

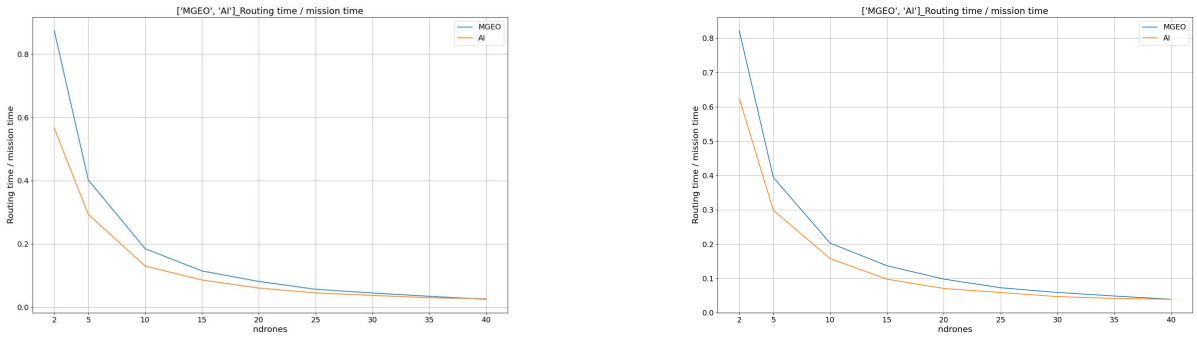


Figure 7: Plot $\frac{RoutingTime}{MissionTime}$ *Rep* vs *Rnd*

Figure 7: With few drones we have good results than MGEO. With more drones MGEO is in advantage.

3.1 Conclusions

MGEO Algorithm is better than us, also with 2 Depot's Scenario. Our approach is enough strong to deliver an high number of Γ, but it delivers each ψ with high ξ, in average: This is the **Failure Point** $\rightarrow$ For a future work it's important work on this aspect.

On other hand, the proposed algorithm has consistently low battery consumption, as expected.

Luckly there aren't too many differencess between *Rep* and *Rnd*. We have same good performance also in *Rnd* $\rightarrow$ With our α and γ parameters algorithm can adapt fastly to changing on the scenario (*Rnd*), without a large exploiting of past knowledges (*Rep*).

4 Components' Contribute

- **Giordano:** With other components he decided what it is better to implement $\rightarrow$ It also briefly fix some code errors $\rightarrow$ Mainly he is the main responsible for the writing and the revision of the Report and for every formalism realized in it.

- **Mattia:** He worked on the design and implementation of the future development algorithms and at plotting.
- **Michele:** He took part in the design of the algorithm and its implementation, of both things he is the main author. He also collaborated in drafting the report, writing part of it, revising it and collaborating in writing the formulas.

THANKS FOR YOUR ATTENTION :)