


DARP: Divide Areas Algorithm for Optimal Multi-Robot Coverage Path Planning

 medium.com/@athanasios.kapoutsis/darp-divide-areas-algorithm-for-optimal-multi-robot-coverage-path-planning-2fed77b990a3

Athanasios Kapoutsis

September 1, 2021



One of the fundamental problems in robotics is to determine an optimal path involving all points of a given area of interest while avoiding sub-areas with specific characteristics (e.g., obstacles, no-fly zones, etc.). In the literature, this problem is often referred to as coverage path planning (CPP), but can also be found as sweeping, exhaustive geographical search, area patrolling, etc. This problem is directly related to a plethora of robotic applications, such as vacuum cleaning robots, seabed mapping, demining robots, automated harvesters, planetary exploration, and search and rescue operations.

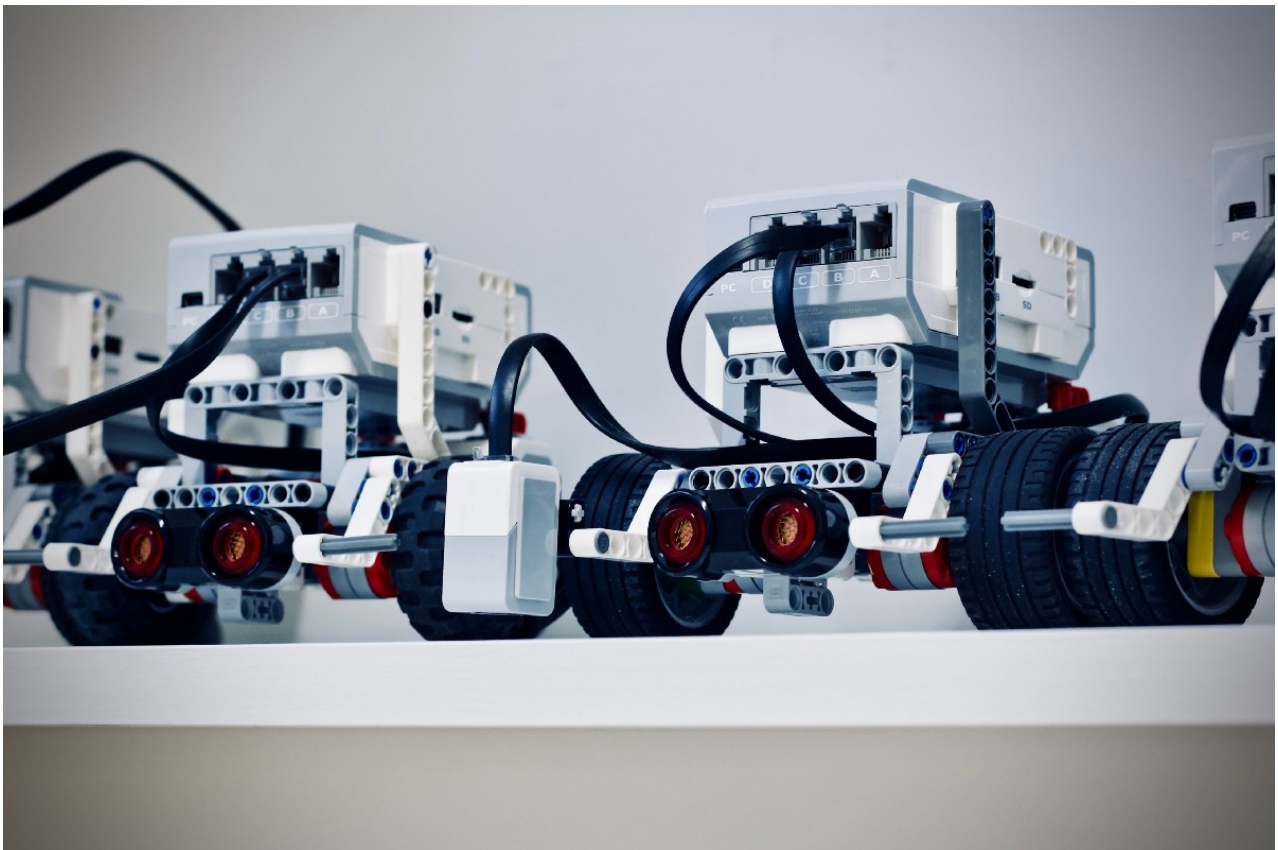
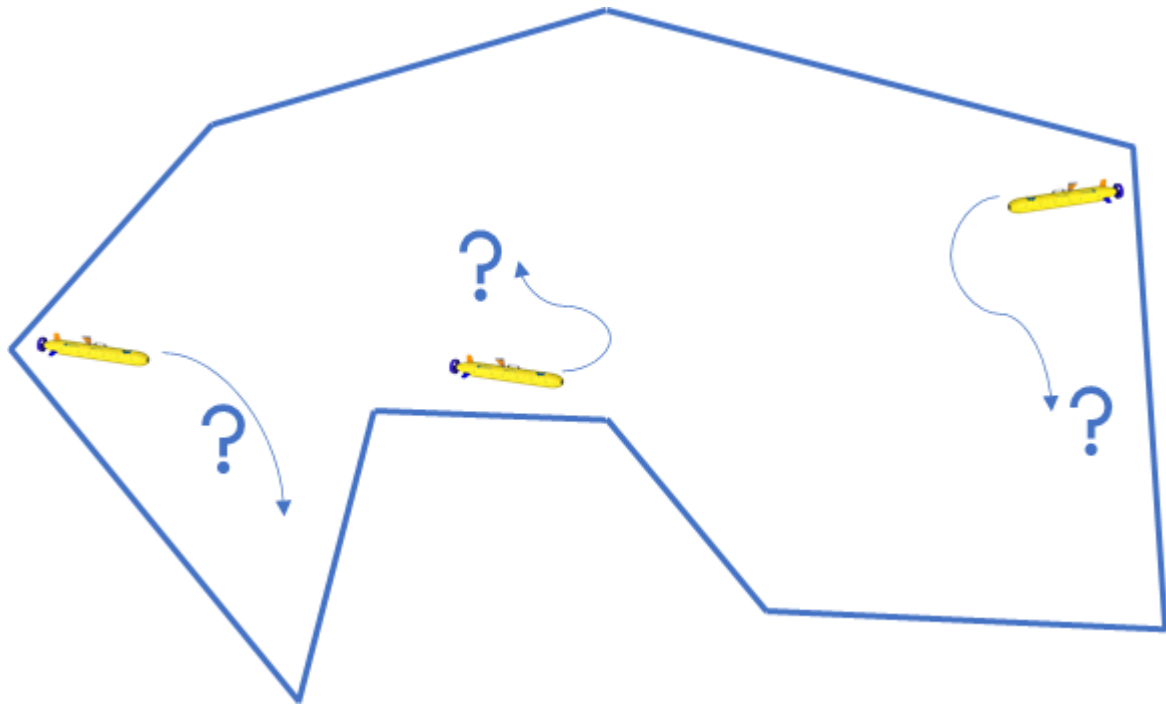


Photo by on

Multi-Robot Coverage Path Planning (mCPP) | Problem Definition

Such a problem is usually defined as follows:



Coverage Path Planning (CPP) problem formulation

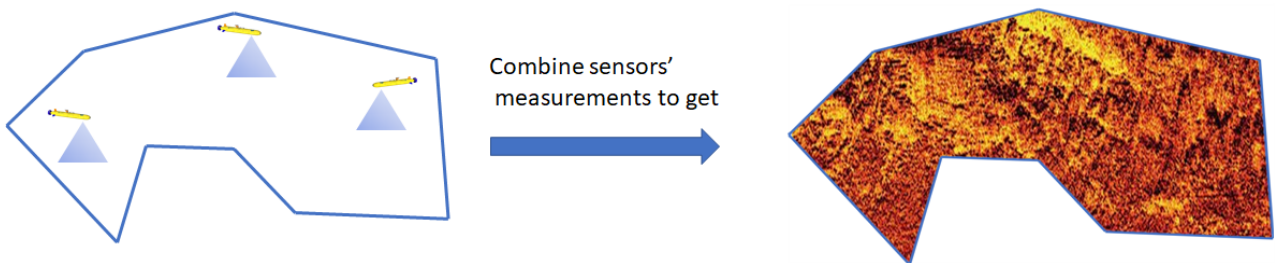
:

- 1) An area of interest
- 2) Robots' initial positions

:

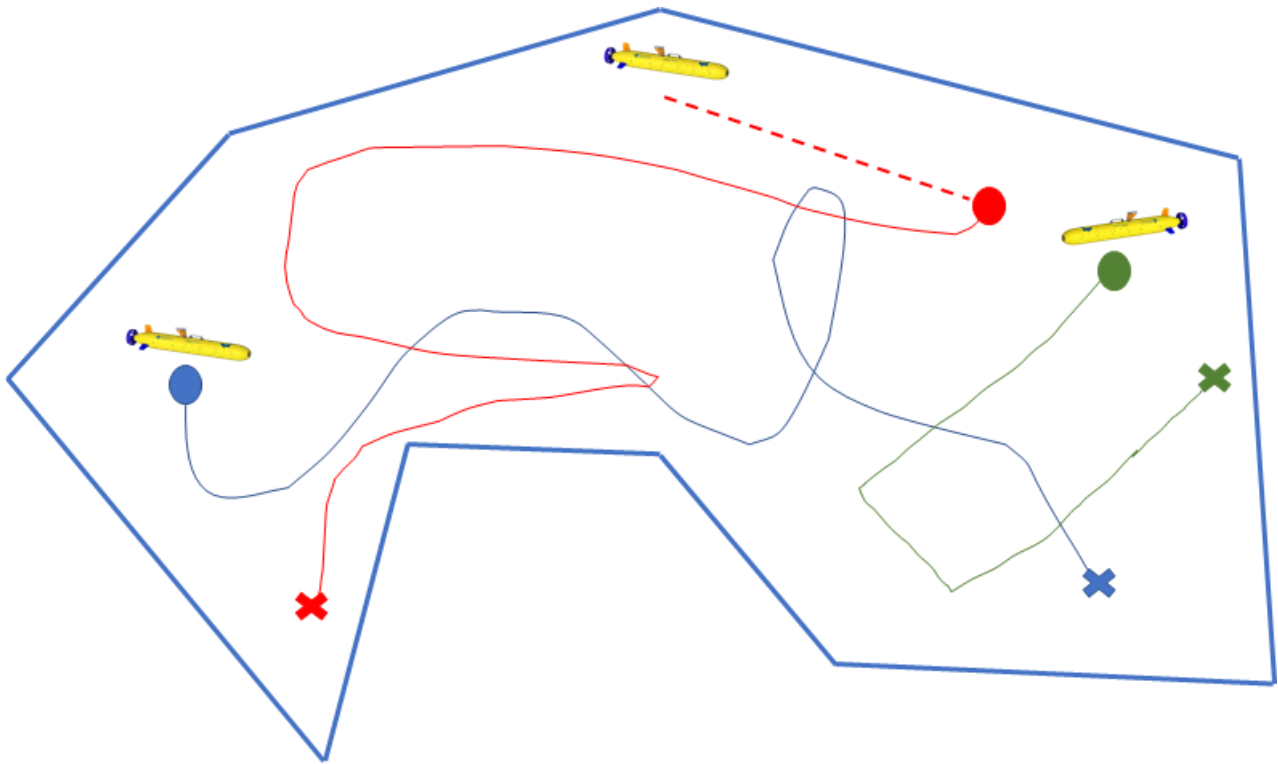
Design robot paths that **completely cover** this area of interest in the **minimum possible time**

Usually, complete coverage is needed to reconstruct the underlying environment (e.g., seabed, ground morphology, cm-level mapping, etc.).



Combine robots' measurements to generate accurate representations of the environment

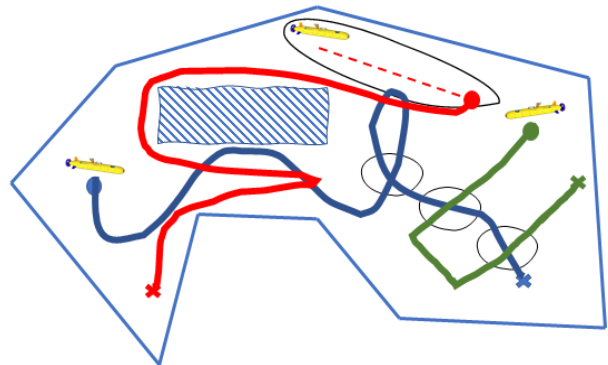
Of course, this can even be achieved by a single robot moving randomly in the space after a (probably) long period of time. However, the limited battery capabilities of autonomous vehicles, in addition to the constantly increased areas that need to be covered/monitored, have motivated the deployment of several autonomous devices with advanced path planning mechanisms.



Suboptimal path generation.

What are the key attributes that affect the performance?

- *Eliminate backtracking*
- *Leave no area uncovered*
- *Equal robots' trajectories*
- *Avoid mismatch between current position and starting location*

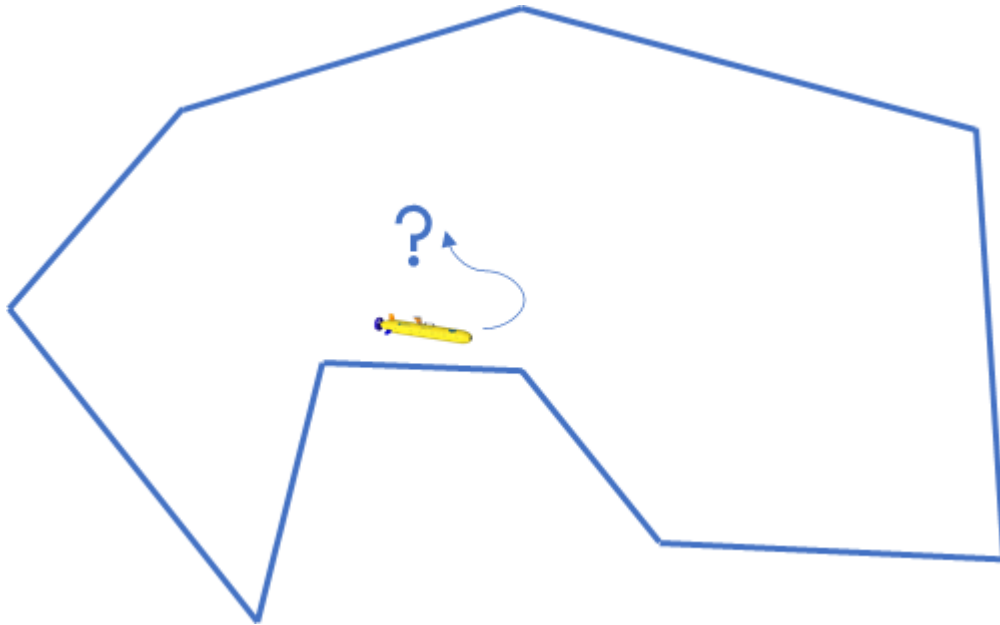


mCPP key attributes

The is that the mCPP setup is at least NP-hard [J]! Ok, mCPP is hard, but **what about single robot CPP?**

Single robot Coverage Path Planning Problem

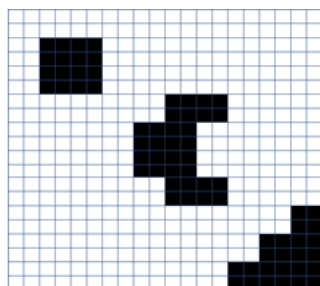
Design the trajectory for a single robot to cover the whole operational area in the minimum possible time:



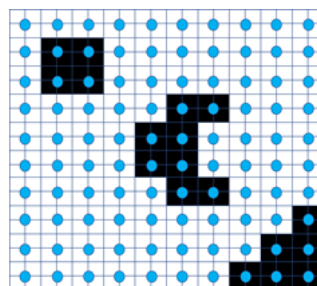
Single robot Coverage Path Planning setup

Good news at last! There exists a dedicated $O(n)$ algorithm, called **Spanning-Tree Coverage (STC)** [2], that computes the optimal robot path under some mild discretization assumptions, where n denotes the size of the area.

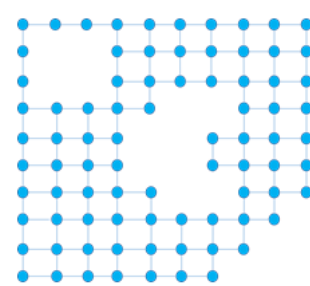
Spanning-Tree Coverage (STC) Algorithm in a nutshell



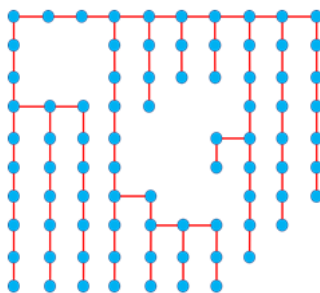
(1) Area discretization



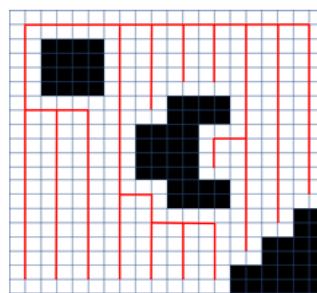
(2) Convert cells to nodes



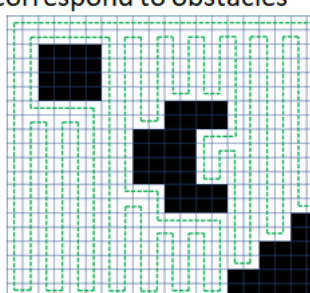
(3) Discard nodes and edges that correspond to obstacles



(4) Construct a Minimum Spanning Tree



(5) The resulting MST is "our guide" for the navigation



(6) The final path that circumnavigates the MST

STC algorithm in a nutshell

Apart from being able to tackle the CPP setup in linear time with respect to the size of the grid, STC also designs a coverage path that

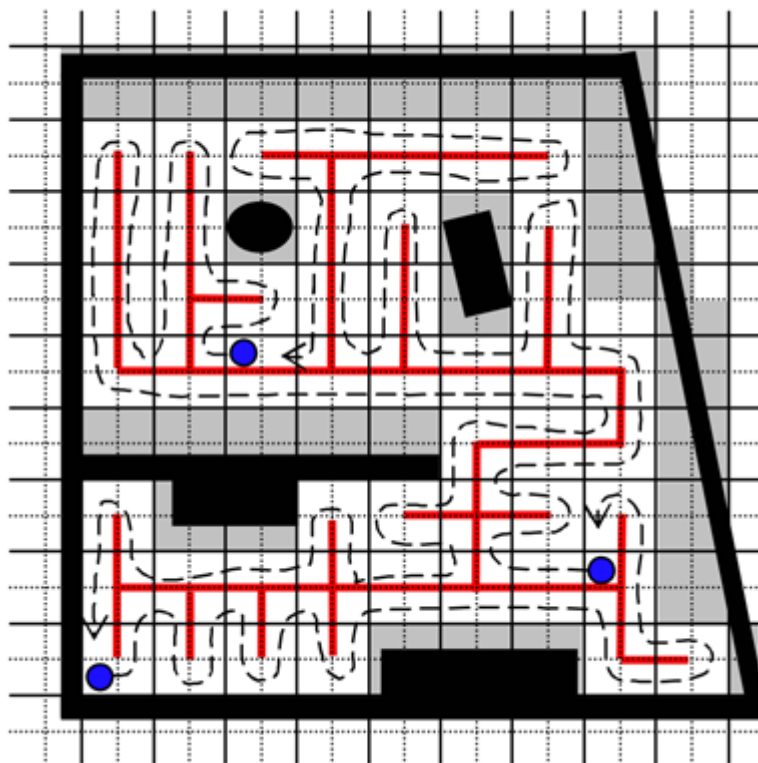
1. finishes in the starting position
2. is actually valid from any starting point inside the operational area

Both features are of paramount importance. The first one eases the utilization of the generated trajectory as a patrolling path that can be executed continuously, enabling efficient persistent coverage. The latter enables the deployment of the mobile robot in different locations without having to change a thing in the mission plan!

Yes! → Opt_MSTC Algorithm [3]

Opt_MSTC Algorithm [3]

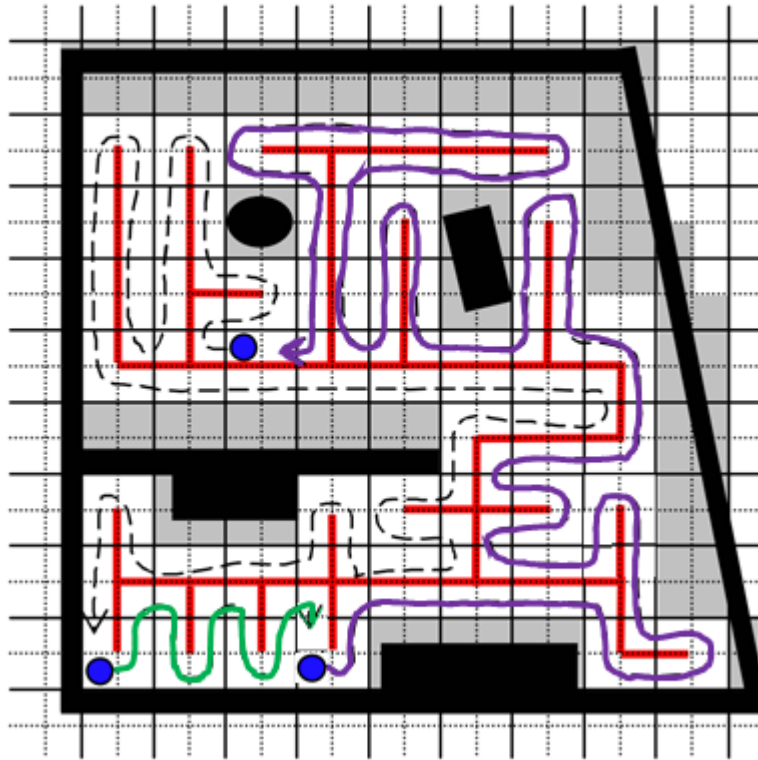
1. Extract a single path using STC algorithm
2. Each robot follows the segment that starts from its initial position till it reaches the starting position of any other robot



Opt_MSTC Algorithm

- + $O(n)$ independent of the number of robots
- +non-backtracking solution
- +complete coverage
- +no mismatch between the current position and starting location
- Uneven robot trajectories. Highly dependent on the initial positions of the robots

The following illustration depicts the influence of the initial positions of the robots on their resulting paths:

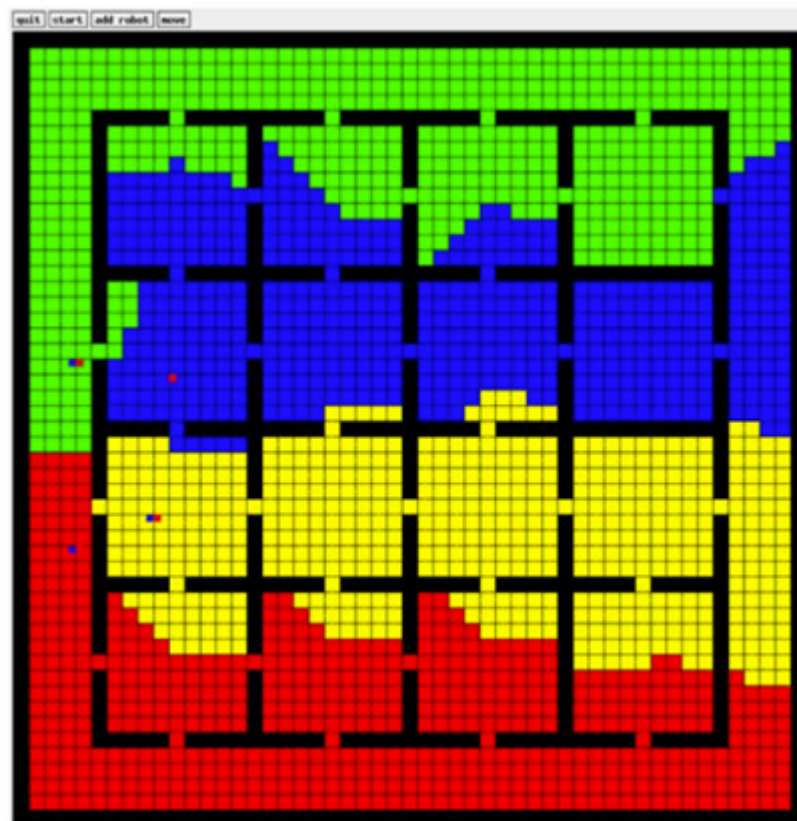


Opt_MSTC Algorithm with different initial positions for the robots

It is worth mentioning that the structure of the spanning tree can be tuned to favor a multi-robot setup with specific initial positions. However, there is no analytical methodology to completely remove the effect of the initial position of the robots on the resulted quality of the solution.

Relax backtracking constrain

Recognizing such an inefficient task allocation, many algorithms have been proposed to mitigate the effect of robots' initial positions by relaxing the backtracking constrain of the original mCPP setup. The most representative approach is **Multi-robot Forest Coverage (MFC)** [4], a polynomial-time heuristic algorithm capable of constructing spanning trees for each robot.



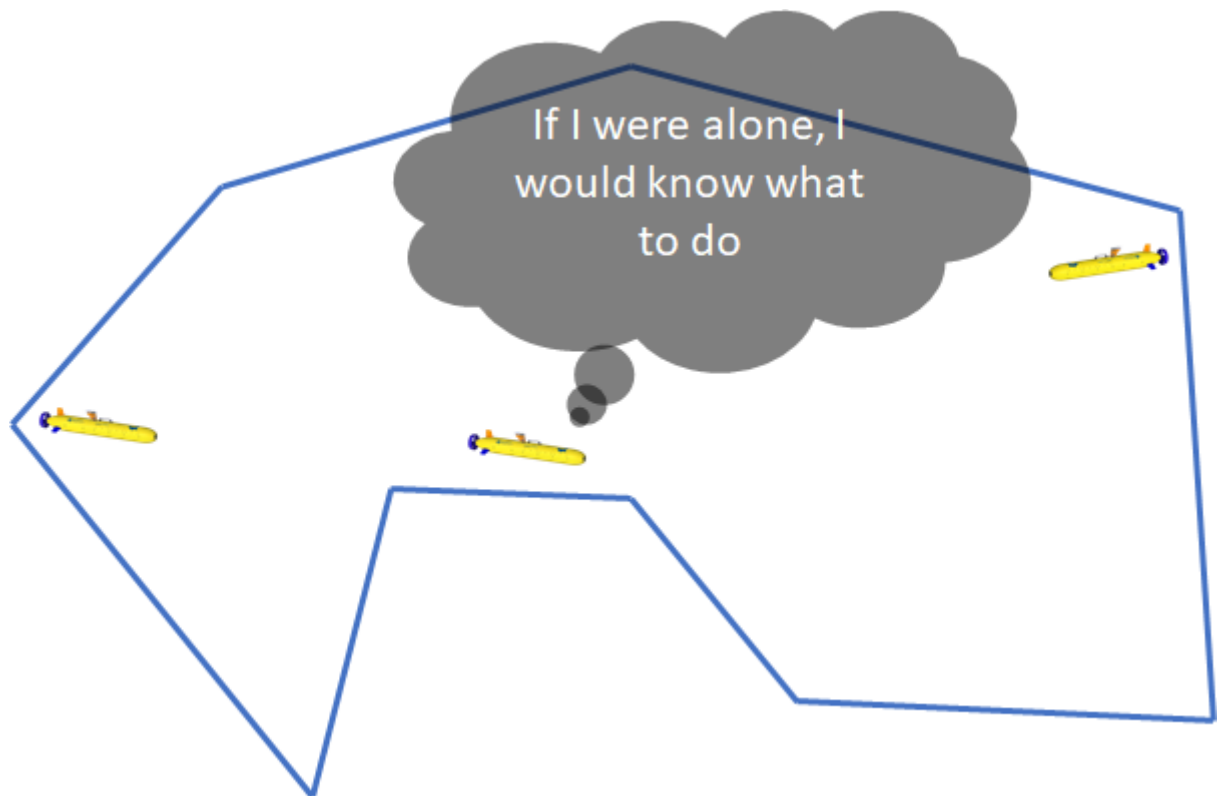
Multirobot Forest Coverage (MFC) Algorithm in action

- +complete coverage
- +no mismatch between the current position and starting location
- +significant improvement over Opt_MSTC
- +improved fairness among robots' paths
- produced solution can be at most 16 times greater than the optimal one
- backtracking solution

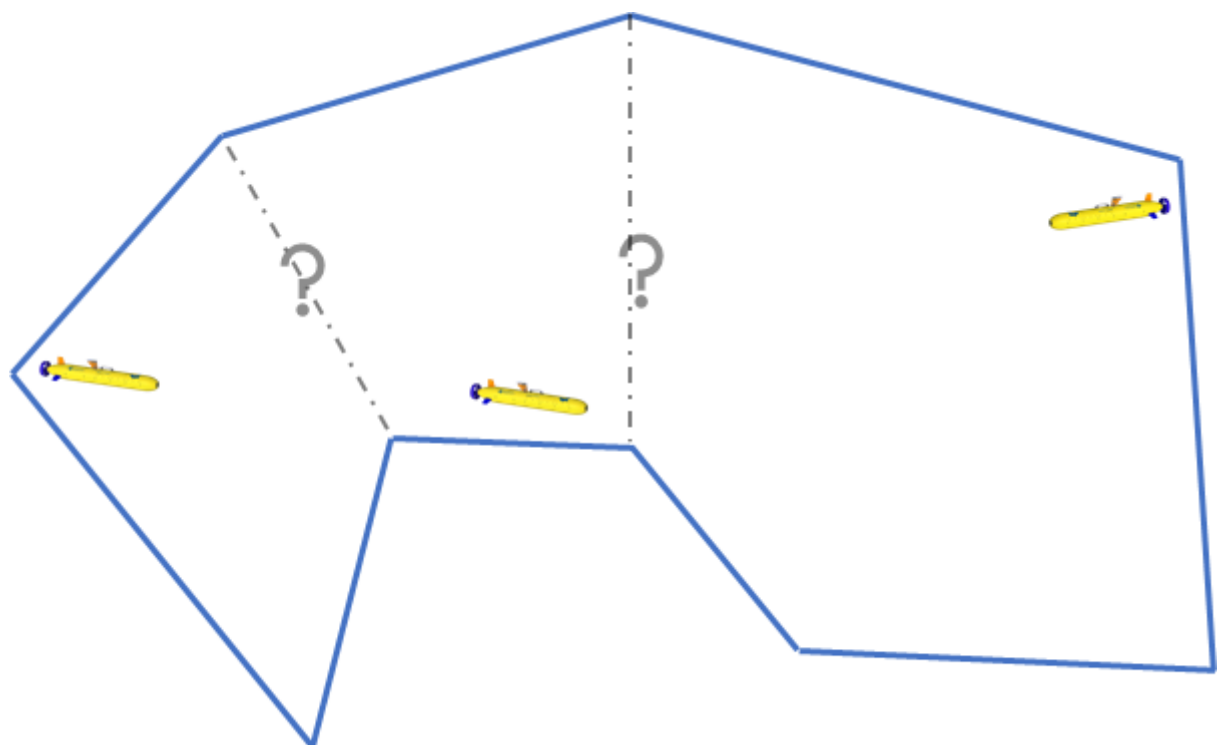
In simple words, this family of approaches demonstrated that it was beneficial to sacrifice the non-backtracking property in exchange for more balanced robot paths.

Exploit STC in mCPP domain with area division

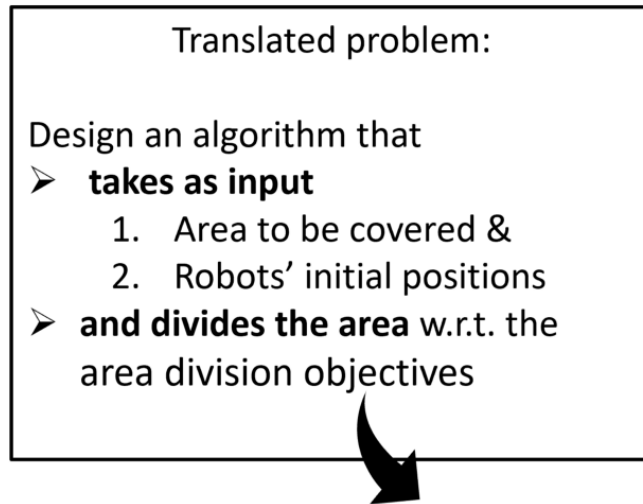
STC algorithm can optimally tackle the single-robot CPP problem under some mild discretization assumptions.



| : Define areas for each robot, and then apply STC algorithm to each one of them.



By design, the STC paths inside these areas are going to be optimal, with respect to the optimality conditions defined earlier. But the catch here is that these paths are only optimal for the divided subparts, and the overall mCPP problem might not be appropriately tackled. Therefore, any such division should meet the following to constitute an optimal solution after the appliance of the STC algorithm inside the separate robot regions:



mCPP optimality conditions	Area division objectives
Eliminate backtracking	Exclusive, spatially-connected robot regions
Leave no area uncovered	The union of all sub-areas should reconstruct the original area
Equal robots' trajectories	Equal sub-areas for each robot
Avoid mismatch between current position and starting location	Each robot's initial position should be inside its exclusive region

From the original mCPP to area division objectives

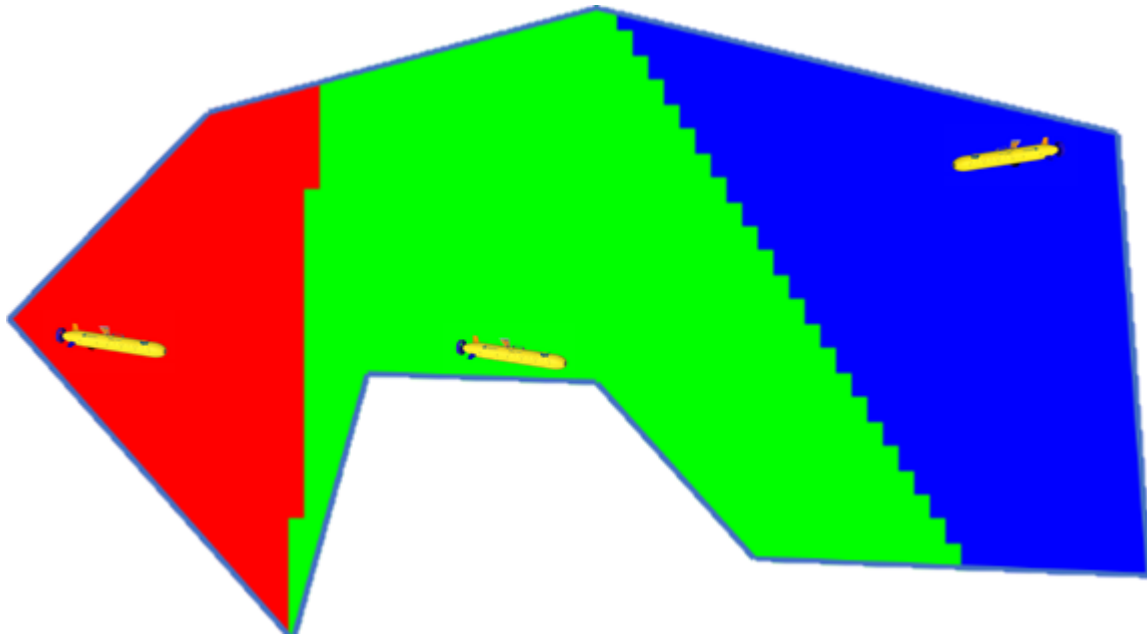
Having translated the original mCPP setup to an equivalent area division setup, we should now shift our attention to the algorithms/methodologies that could address such a setup.

Voronoi space partitioning

Probably the first approach that comes to mind is the Voronoi space partition:

Voronoi partitioning algorithm for grid space:

1. Calculate the Euclidean distance between each robot and every cell of the terrain
2. Each cell is assigned to the closest robot



Voronoi partition for grd space

How many of the area division objectives have been met?

Area division objectives	Voronoi space partitioning
1 Exclusive, spatially-connected robot regions	✓
2 The union of all sub-areas should reconstruct the original area	✓
3 Equal sub-areas for each robot	✗
4 Each robot's initial position should be inside its exclusive region	✓

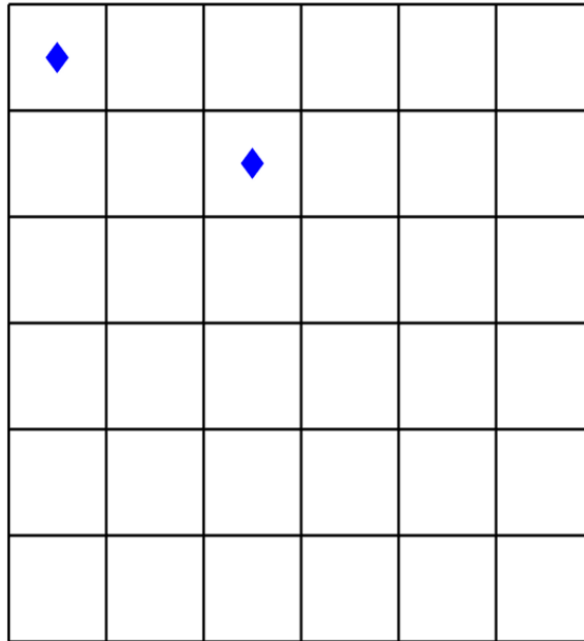
By applying Voronoi space partition, several conditions are already met, however,

the failure to construct equal regions, independently of the robots' initial positions (3rd objective) is a big deal!

Technically, such a methodology would be pretty similar to Opt_MSTC [3], and the overall quality of the solution would strongly depend on the initial positions of the robots.

Let's see this with an example:

Robots' initial positions



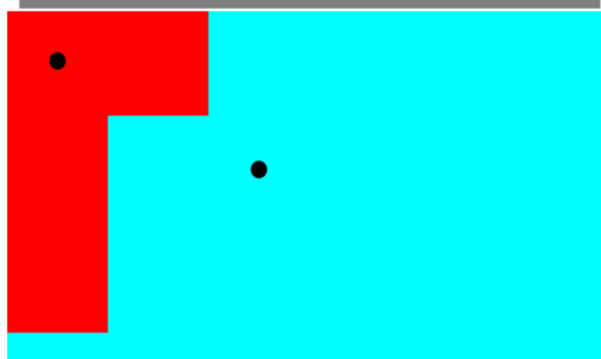
Distance Matrix for Robot #1

0	1	2	3	4	5
1	1.4142	2.2361	3.1623	4.1231	5.099
2	2.2361	2.8284	3.6056	4.4721	5.3852
3	3.1623	3.6056	4.2426	5	5.831
4	4.1231	4.4721	5	5.6569	6.4031
5	5.099	5.3852	5.831	6.4031	7.0711

Distance Matrix for Robot #2

2.2361	1.4142	1	1.4142	2.2361	3.1623
2	1	0	1	2	3
2.2361	1.4142	1	1.4142	2.2361	3.1623
2.8284	2.2361	2	2.2361	2.8284	3.6056
3.6056	3.1623	3	3.1623	3.6056	4.2426
4.4721	4.1231	4	4.1231	4.4721	5

Voronoi partition based on the distance matrices





The final image graphically illustrates the Voronoi space partition based on the Euclidean distance matrices. The cyan robot will cover 32 cells, while the red robot will only cover 4 cells. Assuming that both robots have the same moving and sensing capabilities and can operate simultaneously, the above grid will be covered in $32 \times$ [the time need to scan one grid cell] (completion time for the robot with the longer task).

| Let's now assume that we have (not the Euclidean one), as it is depicted below:

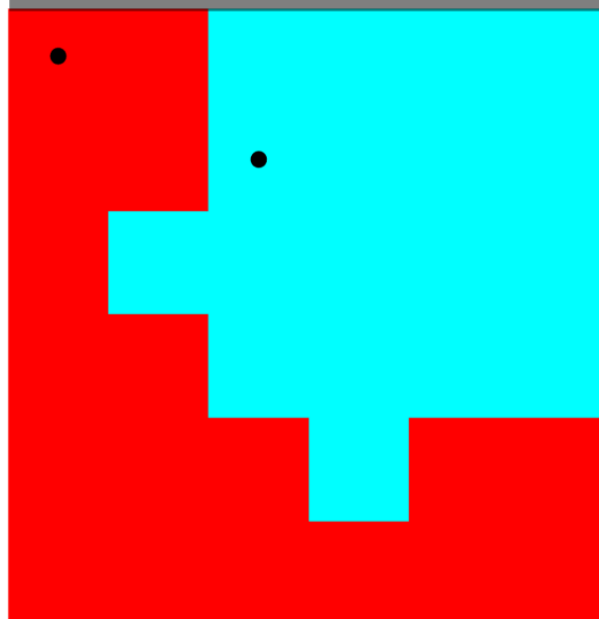
Distance Matrix for Robot #1

0	0.92982	1.867	2.8114	3.7632	4.7223
0.92836	1.3147	2.0868	2.9625	3.8768	4.7972
1.856	2.0777	2.638	3.375	4.1896	5.0467
2.7822	2.9362	3.3598	3.957	4.6663	5.4431
3.7066	3.8247	4.1524	4.646	5.2589	5.9538
4.6288	4.725	4.9942	5.411	5.9444	6.5655

Distance Matrix for Robot #2

3.2631	2.0637	1.4593	2.0637	3.2631	4.6146
2.9186	1.4593	0	1.4593	2.9186	4.3778
3.2631	2.0637	1.4593	2.0637	3.2631	4.6147
4.1275	3.2631	2.9186	3.2631	4.1275	5.2615
5.2615	4.6147	4.3779	4.6147	5.2616	6.1913
6.5261	6.0168	5.8371	6.0168	6.5261	7.2965

Voronoi partition based on the distance matrices



Using this evaluation metric instead of the Euclidean distance results in a fair assignment of 18 cells per robot, reducing the overall completion time by 43%. With the above assignment matrix, we may now apply “blindly” the STC algorithm in each robot territory and ensure that we reached the optimal solution for the original mCPP setup.

The takeaway message is that coupled with can satisfy the previously defined area-division objectives and therefore tackle the original mCPP problem.

But how can we find such a metric function that strongly depends on the area morphology and the initial positions of the robots?

Divide Areas based on Robots' Initial Positions (DARP)

That's exactly the goal of Divide Areas based on Robot's Initial Positions (DARP) algorithm [5].

Core idea:

- > Retain the **assignment process** from **Voronoi partitioning** and
- > **Tune each robot's metric function** to meet the 3rd (**equal division**) objective without violating the others.

#	Area division objectives	
1	Exclusive, spatially-connected robot regions	✓
2	The union of all sub-areas should reconstruct the original area	✓
3	Equal sub-areas for each robot	
4	Each robot's initial position should be inside its exclusive region	✓

DARP algorithm stems from 2 elements:

:

$$J = \sum_{i=1}^N (k_i - f)^2,$$

where k_i denotes the number of assigned cells in the i^{th} robot, f denotes the fair share, i.e., $f = \frac{\#cells}{N}$

> [The number of cells that should be assigned to each robot is **known apriori**.

2. Update robots' metric tables using a multiplier:

$$E_i \leftarrow m_i E_i$$

Combining these two elements, we can construct DARP algorithm version 0.5:

DARP Algorithm version 0.5:

Until *convergence* do

1. Every (x, y) cell is assigned to a robot according to $A(x, y) = \underset{i \in \{1, \dots, N\}}{\operatorname{argmin}} E_i(x, y)$
 - a. Calculate k_i
 - b. $m_i \leftarrow m_i + \eta(k_i - f)$
 - c. Update distance table to be $E_i \leftarrow m_i E_i$

In essence, the DARP algorithm follows a optimization scheme updating each robots' territory separately but towards achieving the overall mCPP objectives.

+convergence guarantees from the coordinate descent algorithms

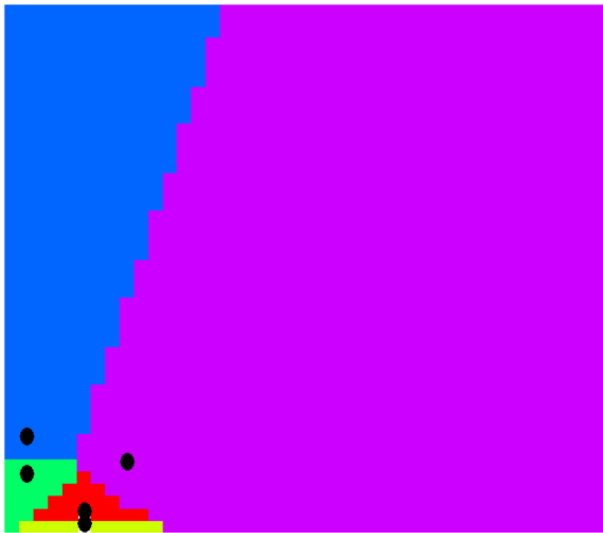
+fast optimization procedure

+very easy to implement

-Spatial connectivity inside each robot' territory is not guaranteed

Deal with spatial disconnectivity

Although in the original Voronoi partitioning, spatial connectivity was guaranteed by design, now, in some cases is violated. This phenomenon is due to the fact that now , over which the assignment process is implemented, . For example, the following figure illustrates the initial partitioning (left-hand side) based on the Euclidean distance, and the subfigure on the right-hand side illustrates the assignment after some iterations.



Initial Euclidean partitioning



Spatially disconnected partition after applying $E_i \leftarrow m_i E_i, \forall i$ several times

Spatially disconnected partitions can now be formed

It is evident that the purple region is disconnected, and therefore with such an assignment, the non-backtracking guarantee is now at risk. To tackle this issue, an extra connectivity matrix is introduced. The rationale behind the calculation of this connectivity matrix is as follows:

C_i calculation methodology

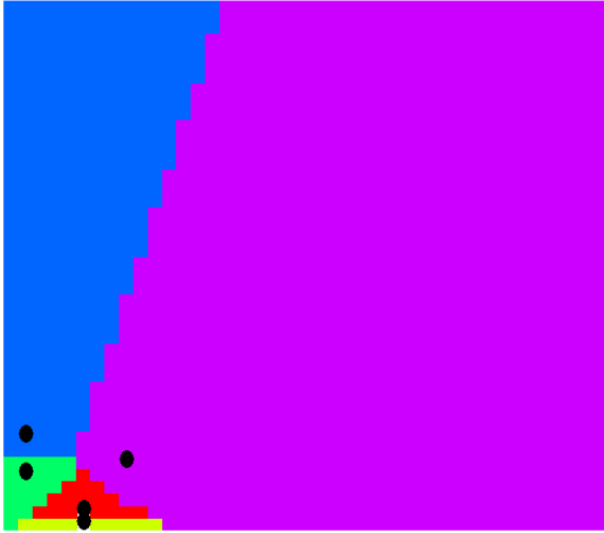
If i^{th} robot contains more than one closed shape regions

- Reward closed shape around the i^{th} robot initial position
- Penalize all other i^{th} closed shapes

Otherwise

C_i is an all-one matrix

In essence, the DARP algorithm incentivizes closed shape regions around the robot's initial position and attempts to eliminate all the others, as is graphically illustrated in the following image:



Initial Euclidean partitioning



Spatially disconnected partition after applying $E_i \leftarrow m_i E_i, \forall i$ several times

Reward closed shape regions around the robot's initial positions and penalize all the others.

The following pseudocode presents the DARP algorithm [5], which also encapsulates the connectivity matrix:

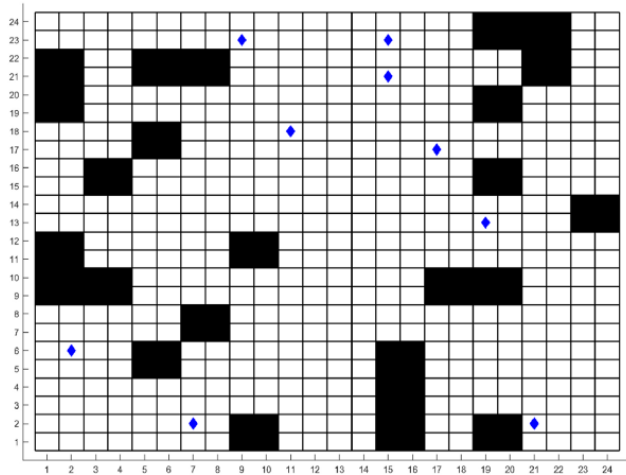
DARP Algorithm version 1.0:

Until ***all area division objectives are met*** do

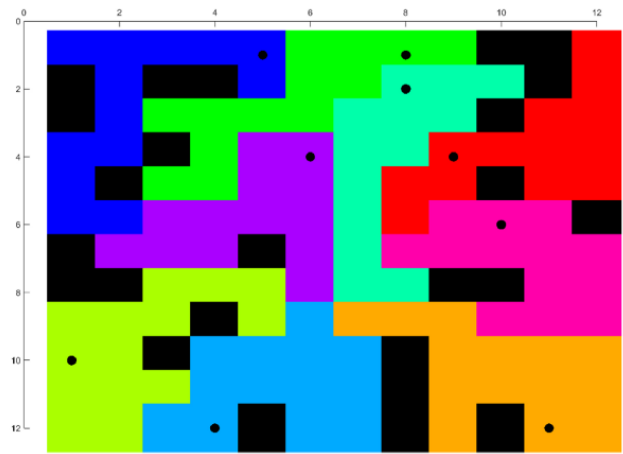
1. Every (x, y) cell is assigned to a robot according to $A(x, y) = \underset{i \in \{1, \dots, N\}}{\operatorname{argmin}} E_i(x, y)$
2. For each i^{th} robot do
 - a. Calculate k_i
 - b. $m_i \leftarrow m_i + \eta(k_i - f)$
 - c. **Calculate connectivity matrix C_i**
 - d. $E_i \leftarrow C_i \odot m_i E_i$

DARP algorithm [5] pseudocode

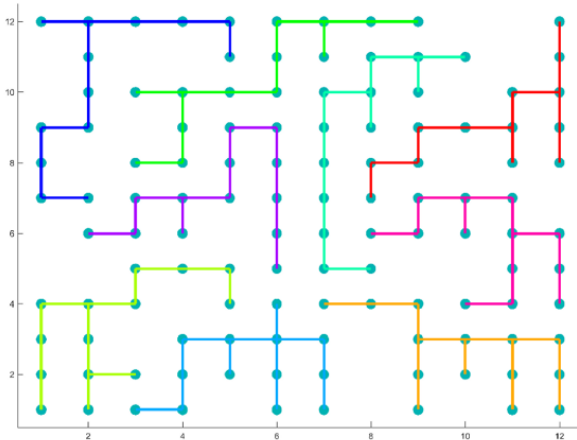
At a glance, the operation of the DARP algorithm coupled with STC for each robot region is illustrated in the following figure:



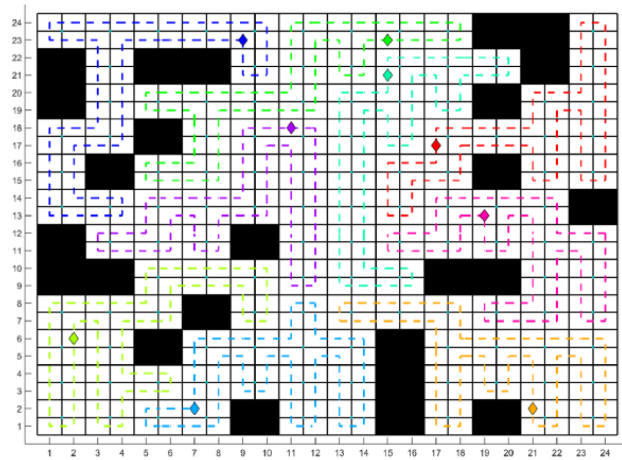
Initial Robots and Obstacles Placement



Divide area into N exclusive sub-areas, utilizing DARP



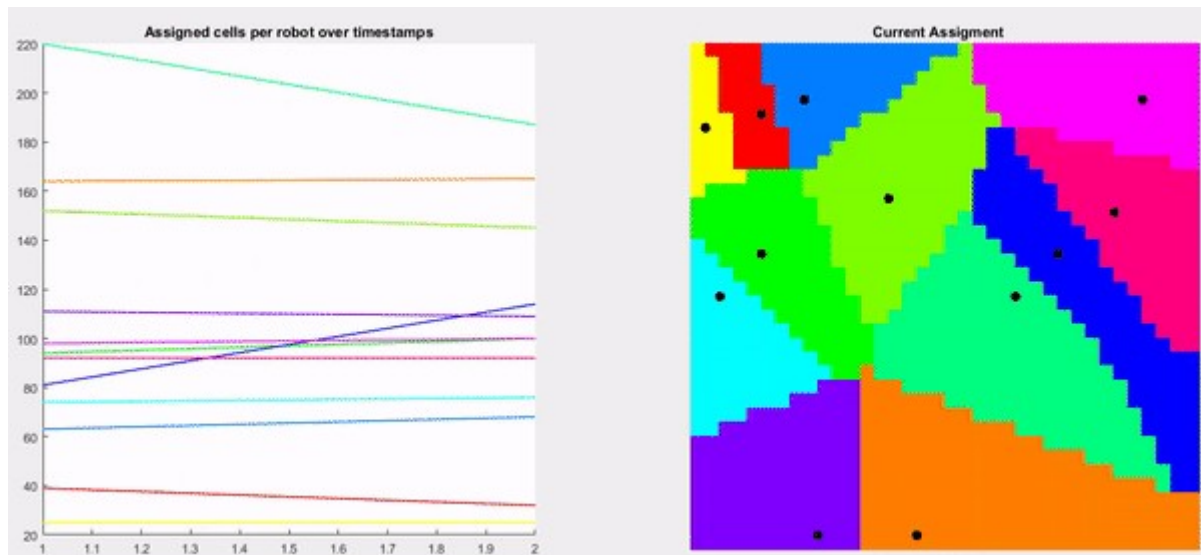
Calculate N MSTs inside each exclusive robot's sub-area



Circumnavigate each robot around the corresponding MST

DARP+STC fundamental steps

During the optimization process and depending on the difficulty of the mCPP setup, robots' territories are expected to change formation several times till the convergence to a fair and spatially connected division:



Assigned cells per robot vs. Current assignment matrix

Beyond fair division

The DARP algorithm, as defined in version 1.0 pseudocode, is tuned to provide equal-area division among the operational robots. However, there are cases where a custom-defined percentage of coverage is more preferable. Such cases could include scenarios with heterogeneous robots, different battery levels, or other human-imposed reasons (e.g., underutilize a specific robot to be readily available for a follow-up mission). To cope with such custom-defined percentages of coverage, the following change in the objective function should take place:

Instead of $J = \sum_{i=1}^N (k_i - f)^2$ define objective function to be

$$J = \sum_{i=1}^N (k_i - f_i)^2$$

Where f_i denotes the number of cells that have to be covered by the i^{th} robot.

For consistency reasons: $\sum_{i=1}^N f_i = \#cells$

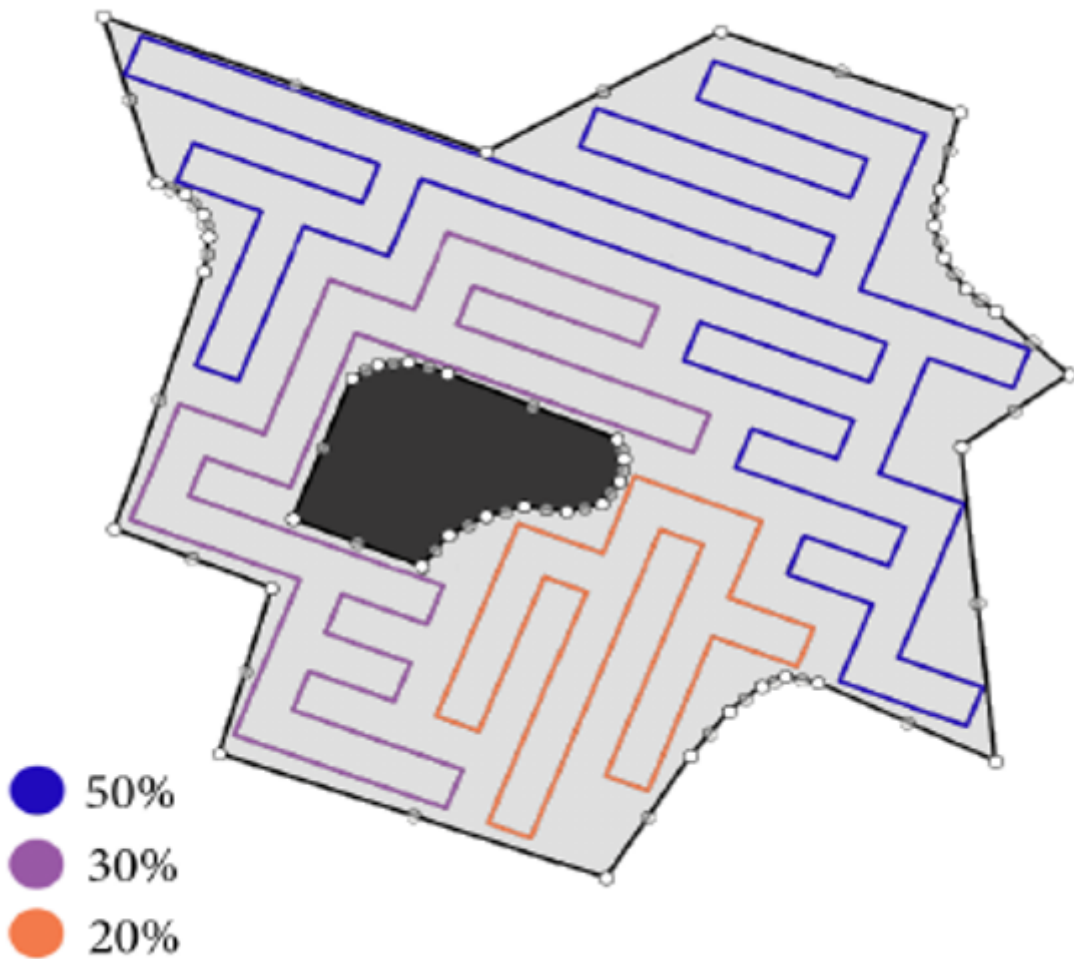
In a nutshell, compared to DARP version 1.0, only the update rule for the i th robot is going to change to encompass the robot-specific percentage:

DARP Algorithm:

Until *all area division objectives are met* do

1. Every (x, y) cell is assigned to a robot according to $A(x, y) = \underset{i \in \{1, \dots, N\}}{\operatorname{argmin}} E_i(x, y)$
2. For each i^{th} robot do
 - a. Calculate k_i
 - b. $m_i \leftarrow m_i + \eta(k_i - f_i)$
 - c. Calculate connectivity matrix C_i
 - d. $E_i \leftarrow C_i \odot m_i E_i$

An example of such a path-planning with 3 robots with 50%, 30%, and 20% requested percentages of coverage is illustrated in the following figure:



Proportional division in mCPP setup

Material

Paper: [Zenodo](#)

GitHub repositories: [Java](#), [Python](#)

GUI demo: [YouTube](#)

ROS integration: [Wiki](#)

Acknowledgments

Big thanks to Savvas Apostolidis and Alik Stefanopoulou for their helpful comments on the original draft of this article.

References

- [1] Rekleitis, I., New, A.P., Rankin, E.S. and Choset, H., 2008. *Efficient boustrophedon multi-robot coverage: an algorithmic approach*. *Annals of Mathematics and Artificial Intelligence*
- [2] Gabriely, Y. and Rimon, E., 2001. *Spanning-tree based coverage of continuous areas by a mobile robot*. *Annals of mathematics and artificial intelligence*
- [3] Agmon, N., Hazon, N. and Kaminka, G.A., 2006, May. *Constructing spanning trees for efficient multi-robot coverage*. *ICRA 2006*
- [4] Zheng, X., Koenig, S., Kempe, D. and Jain, S., 2010. *Multirobot forest coverage for weighted and unweighted terrain*. *IEEE Transactions on Robotics*
- [5] Kapoutsis, A.C., Chatzichristofis, S.A. and Kosmatopoulos, E.B., 2017. *DARP: divide areas algorithm for optimal multi-robot coverage path planning*. *Journal of Intelligent & Robotic Systems*