Multi-Robot Weighted Coverage Path Planning: a Solution based on the DARP Algorithm

Olivier Idir, Alessandro Renzaglia

Multi-Robot Weighted Coverage Path Planning: a Solution based on the DARP Algorithm

Olivier Idir[1] and Alessandro Renzaglia[1]

***Abstract*— Covering a given area with a team of mobile robots in a minimum time is a well-studied problem with many real-world applications. A rarely studied subject, however, is the case of a weighted plane: due to the necessity of taking time-consuming measurements or having to traverse different kinds of terrains, the coverage time may vary over the environment and the path planning needs to be adapted accordingly. In this paper, we present an adapted version of a state-of-the-art mCPP (multi-robot coverage path planning) approach, the DARP algorithm, to make it suitable to deal with weighted environments. In particular, we propose several modifications to DARP that allow overcoming some of its limitations and, as a result, obtain an increased convergence rate and decreased convergence time with respect to the original version. Furthermore, as proved by extensive simulations, these improvements are also noticed in the unweighted version of the problem.**

I. Introduction

In the last decades, teams of cooperative autonomous robots, and especially UAVs (unmanned aerial vehicles), have been frequently deployed to achieve missions such as inspection, search and rescue, and more generally data gathering in large environments ([1], [2], [3]). A fundamental task behind these numerous applications is the ability to cover the entire area of interest in a minimum time. The path-generation problem in such situations is known as CPP (Coverage Path Planning), or mCPP in the multi-robot case. Many algorithms have been designed in order to approximate a solution for this NP-hard problem [4], [5]. A first solution of reference to this problem was presented by Hazon and Kaminka in [6]. In their work, they started from the spanning tree coverage (STC) algorithm [7], and adapted it to the multi-robot situation, proposing an algorithm robust to robot failures. This algorithm, however, is highly dependent on the robots' initial position, and a poorly chosen spanning tree may end up with a total exploration time nearly equal to the time that would have sufficed to a single robot. In order to solve this issue, these same authors presented in [8] a new version, relying on a heuristic to build a spanning tree that will maximize the distance between the robots' initial positions. Instead of generating a single spanning tree for the whole instance, Dong et al. chose in [9] to build simultaneously a spanning tree for each robot, growing towards the center of inertia of the uncovered area. Its performances neighbor the ones of more recent centralized approaches, such as MFC presented in [4], although with the computational advantages of distributed algorithms. More recently, Kapoutsis et al. proposed a new approach, called the DARP algorithm [10], that reduces the multi-robot case into n single-robot problems by partitioning the environment in n regions and assigning each of them to a different robot to finally obtain a complete coverage with no backtracking and minimum coverage path.

[1]Univ Lyon, Inria, INSA Lyon, CITI, 69621 Villeurbanne, France
Emails: olivier.idir@ens-lyon.org, alessandro.renzaglia@inria.fr

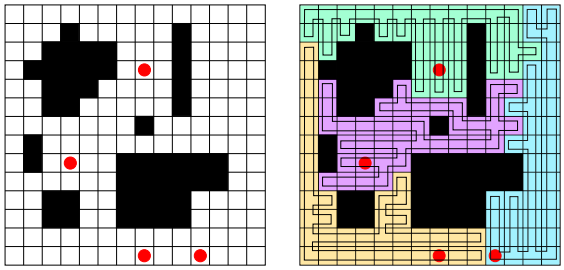


Fig. 1: An example of mCPP instance of size 14×14 and a solution obtained via the DARP algorithm.

In this paper, we study a variation of the classic mCPP problem where some areas require a bigger effort or are longer to cover than others. This can be due to the need to physically take some samples, have a closer look at specific zones, or just traverse more difficult terrain. The path planning problem has thus to take into account not only the total distance traveled by the robots to cover the environment but also this additional workload to optimally distribute the effort among the robots and obtain the trajectories that minimize the mission time.

The contribution of this paper is twofold: firstly, we show that the DARP algorithm can be adapted to deal with this new problem formulation and able to solve the mwCPP in most instances; secondly, we identify some important limitations of the original version, especially when used for mwCPP, and we propose some modifications to increase its performance in terms of convergence rate and convergence time. Extensive results considering several 2D environments show not only the capability of the proposed approach to provide a solution to the mwCPP problem, but also a significant improvement in the unweighted case with respect to the initial DARP.

II. Problem formulation

An instance of mCPP consists of a set of n_r robots, each equipped with an identical sensing tool (camera,

radar, etc.), allowing them to cover a region of diameter at least their size. The environment is then defined by a grid G of identical cells. Some of these cells are defined as "obstacles", and are impassable: they are said *occupied* and form a set $\mathcal{B}$. The other cells are thus passable, and need to be covered at some point by at least one of the robots: they form a set $\mathcal{L}$ (respectively, black and white cells in figure 1). The instance also defines the number of robots and their different starting points, represented as red circles in the figure. It is worth noticing that these grids are only an approximation of truly arbitrary environments, denominated as *approximate cellular decomposition*. Such approximation techniques are commonly used to tackle mCPPs [11].

A solution consists in a set of paths rooted in the initial positions, one for each robot, such that each cell of $\mathcal{L}$ belongs to at least one of these paths, and that the size of the longest path is minimal (that is, we minimize the time needed to cover all cells).

In this paper, we focus on the weighted version of the mCPP, called mwCPP, where each cell c is assigned a weight w_c corresponding to the time necessary to cover this cell. We thus define $W := \sum_{v \in \mathcal{L}} w_v$, corresponding to the total *work time* required to cover the space.

III. The DARP algorithm

The solution proposed in this paper is an extension and adaptation of the DARP (Divide Areas based on Robots initial Positions) algorithm, initially presented by Katpousis et al. in [10] to solve mCPP. DARP defines paths of nearly equal lengths for the robots, thus ensuring a near-perfect time efficiency. Its convergence speed (cubic in the size of the input), and its performances — near optimal, thus much better than the precedent standard, the optimized MSTC [8] — have made it a frequently used mCPP algorithm ([12], [13], [14]).

The DARP algorithm proceeds in two steps. The first part aims to split the environment in areas of nearly equal surface, each containing the initial position of exactly one robot. In a second stage, it generates spanning trees in these regions, and has the robots circumnavigate around them, thus defining paths. An instance of DARP is an instance of mCPP, with the additional assumption that the size of each cell corresponds to twice the sensing diameter of the robots.

A. Area division

The first step corresponds to a competition between the robots to claim the cells for themselves. We define, for each iteration step j, an assignation matrix A_j, of size `rows` $\times$ `cols`, that will hold the current distribution of cells among robots. We also define, for each robot $r_i, 1 \leqslant i \leqslant n_r$ and for each step j, a priority matrix $E_{i,j}$ of the same dimension. Its cells are initialized with the airborne distances to each cell from the robot r_i.

At the beginning of iteration j, the matrix A_j becomes $(argmin_{i \leqslant n_r}(E_{i|x,y}))_{x,y}$: for each cell, it takes the value of the robot having the lowest priority in this cell. We thus define, for each $i \leqslant n_r$, the set $L_{i,j}$ of unoccupied cells *assigned to* r_i (or *claimed by* r_i). We have that $\forall j, \bigsqcup_{i \leqslant n_r} L_{i,j} = \mathcal{L}$: each unoccupied cell is assigned to one and only one robot. We will eventually aim to satisfy the following conditions: all robots should have (nearly) the same number of cells, and the said assigned regions are connected, in order to be able to define paths.

1) Regions of nearly equal size: We denote, for each robot $k_{i,j} := |L_{i,j}|$ its current number of assigned cells, and $f := \frac{|\mathcal{L}|}{n_r}$ the ideal number of cells. We then define $J := \sum_{i \leqslant n_r}(k_{i,j} - f)^2$ as the sum of square differences to f, thus at its smallest when each $k_{i,j}$ is nearly equal to f – when each robot covers a nearly equal proportion of cells. In order to minimize J, the option chosen by Katpousis et al. was to implement a gradient descent. At each step, we will define $E_{i,j+1} := m_{j_i} E_{i,j}$ with $(m_j)_{i \leqslant n_r}$ a scalar vector. We can thus view J_{j+1} as a function depending on m (after the update of the assignation matrix A following the changes in the E_i's). We will then proceed to a gradient descent on J, using a cyclic coordinate descent, [15], to minimize J along each coordinate.

The updated value is then defined by $m_{j_i} := 1 - \eta \frac{\partial J}{\partial m_i}$, with η a constant and m_i the scalar in $i-$th position:

$$m_{j_i} := 1 - 2\,\eta\,(k_i - f)\frac{\partial k_i}{\partial m_i} \tag{1}$$

Then, the authors state that all these $\frac{\partial k_i}{\partial m_i}$ are nearly identical, and that if all these partial derivatives get multiplied by a factor α, the obtained $E_{i,j+1}$'s still define the same order relations between coefficients. We can therefore approximate this value by

$$m_{j_i} := 1 + c\,(k_i - f) \tag{2}$$

with c a positive constant, because when $m_{i,j}$ increases, the values of $E_{i,j+1}$ too, thus reducing $|\mathcal{L}_{i,j+1}| = k_{i,l+1}$.

As the evolution of J is convex along every m_i, according to the cyclic coordinate descent, we should ultimately reach a m^* such that $\forall m \in \mathbb{R}^{n_r}, J(m^*) \leqslant J(m)$: at this point, all k_i's are nearly equal.

2) Ensuring the connectivity: The objective, however, is not only to share the cells among the robots: we also want the $L_{i,j}$ to form connected regions. DARP thus adds a corrective multiplier when updating the $E_{i,j}$ to incentivize the regions assigned to the different robots to become connected. We define, for each robot r_i,

$$C_{i,j|x,y} := \min_{r \in \mathcal{R}_{i,j}}(||[x,y] - r||) - \min_{q \in \mathcal{Q}_{i,j}}(||[x,y] - q||) \tag{3}$$

where $R_{i,j}$ is the connected set of cells assigned to r_i where lies the robot's initial position, and $Q_{i,j}$ the union of all other cells assigned to r_i. That is, the value in a given cell is bigger the further it is from the initial connected component (and the closer it gets to the other ones). We can then define the difference matrix C':

$$C'_{i,j} := 1 + \frac{\mu}{\max_{x,y}(C_{i,j|x,y}) - \min_{x,y}(C_{i,j|x,y})} C_{i,j} \tag{4}$$

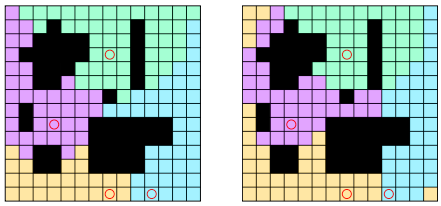


Fig. 2: The DARP temporary assignation matrix on the previous grid, at iteration steps 18 and 48.

That is, these corrective terms are seen as a difference to 1, scaled back to the order of magnitude of a chosen small μ. The idea is that it rewards the cells around $R_{i,j}$, and penalizes the ones around the other connected components. We finally define $E_{i,j+1} = C'_{i,j} \odot (m_{j_i} \cdot E_{i,j})$, where $\odot$ corresponds to the elementwise product.[1] It is to note that we do not forbid disconnected component: we actually want them to appear. However, they will be quickly corrected, due to this aforementioned factor.

Starting from the instance described in figure 1, this process goes, after some intermediary iterations (figure 2), to a final acceptable cell distribution (already seen in figure 1), where all the robots get the same number of cells, except for the pink one who gets 1 more.

B. Generating the paths

Once these regions defined, we generate the actual paths of our robots by finding a spanning tree in each of these regions. Thanks to the hypothesis on the cells' diameter compared to the sensing diameter, circumnavigating that tree thus forms covering paths of minimal length, as in figure 1. These paths also have the advantage of being independent of the robots' starting positions, and to have them return to their starting point at the end of the process.

However, mCPP being a NP-hard problem, DARP cannot possibly work well on all instances. The initial work presented in [10], though, does not linger on those edge cases, which means that such convergence failures have not been clearly documented yet. It is one of the objectives of this paper.

IV. Naive use of DARP for mwCPP

Looking at the different steps of DARP, we can notice that few adaptations are sufficient to incorporate the weight map and thus try to obtain a solution to the mwCPP problem. To take weights into account, we can indeed define $k_{w,i,j} := \sum_{c \in \mathcal{L} | A_{j|c}=i} w_c$, that corresponds to the total weight assigned to the robot r_i, and, instead of J, try to minimize $J_w := \sum_{i \leqslant n_r} (k_{w,i,j} - f)^2$, where f still represents the ideal share but this time corresponds to the desired weight fraction per robot, i.e. $f = \frac{1}{n_r} \sum_{v \in \mathcal{L}} w_v$. It therefore seems logical to try to use DARP to solve mwCPP, as it does not seem much affected in its behavior by this change. An example of solution in a weighted scenario can be seen in figure 3. The question then lies in how such a modification will affect the convergence of the algorithm.

[1] The initial article defines $E_{i,j+1} = C_{i,j} \odot (m_{j_i} \cdot E_{i,j})$, but this is most probably a mistake, and is corrected in their implementation of the algorithm: `https://github.com/alice-st/DARP`

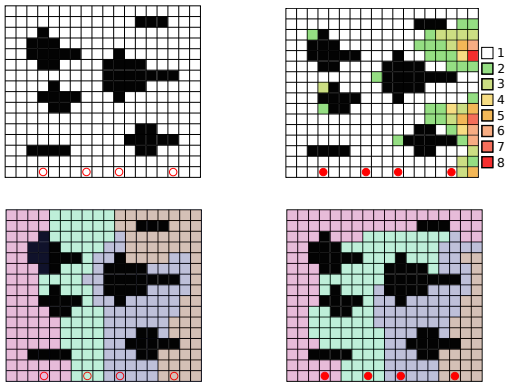


Fig. 3: The impact of weights in the environment partition. On the left an unweighted grid while on the right some clustered weights force the corresponding regions to be smaller to obtain a balance coverage time.

instance	unweighted	clustered weights	spread weights
hills	95	108	50.006
corridor	753	70706 (2)	559
barriers	896	429	1657
cave	30502 (1)	30408 (1)	32870 (1)
konigsberg	71	395	38

TABLE I: Number of iterations required for DARP to converge in both unweighted and weighted cases (with clustered and spread weights) of the same grids. The number between parentheses corresponds to the number of divergences.

We observe that the convergence is generally slower in the weighted case, and some cases of divergences or suboptimal solution can even be observed in this latter case that were not present in the unweighted case. This is evidenced in table I (that uses some test instances described in section VI and shown in figure 6). In particular, the instances with isolated weights are usually a bit longer to solve, probably due to the locality induced by the connectivity multiplier. Nevertheless, this approach, though probably suboptimal, attests that the resolution of mwCPP is possible through DARP. The objective is then to identify the origin of the difference in terms of convergence, see how can we accelerate it, and possibly avoid the divergence in some instances.

A. Practical considerations on DARP

Working with DARP in the attempt to make it more suitable to our problem made us aware of some issues of the initial version, as well as of the practical impact of some of its approximations.

1) Derivation on discrete values: The most critical point in the DARP algorithm is that the gradient descent

relies on the approximation of such a gradient that does not always hold. The computability of DARP relies on the passage from the equation (1) to (2). It allows us to get rid of a partial derivative in order to reach an expression that can be efficiently computed. The issue, however, is that such an approximation is based on an intuition, and is not reliable in all situations. In particular, we have two points of concern as to this approach. First, since the changes in k_i are discrete, the partial derivative $\frac{\partial k_i}{\partial m_i}$ is not well-defined and its meaning is unclear in the general case. It would be null almost everywhere, and amount to infinity (or minus infinity) in a finite set of points[2]. Second, the argument they advance is that these different $\frac{\partial k_i}{\partial m_i}$ are "almost identical for each robot". This assumption, however, is no longer accurate in the weighted case, as cells having more weight will induce bigger partial derivatives.

But even in the unweighted case, this notion raised some issues: through the iterations, one of the E_i can have grown to have cells with values thousands of times bigger than the other robots' in most cells that it does not claim (values ranging from 10^{10} to 10^{60} can actually appear as early as iteration 15,000). It then seems highly unlikely that its partial derivative will be as important as the other ones, as it will take a great number of iterations before any of these cells could possibly be claimed by the other robots. Therefore, assuming that its partial derivative is *almost identical* to the other robots' proves inaccurate in many more cases than expected, and especially in the weighted case. Thus, even though DARP converges quickly in most (unweighted) instances, these assumptions are less reliable in the weighted case and require some additional consideration in order to avoid too frequent divergences.

2) Connectivity correction multiplier: Another issue comes from (3). These correction matrices are completely disjoint from the gradient-like approach: they thus come without any guarantee as to the respect of convergence they may or may not induce. They are a corrective multiplier applied to penalize disconnected solutions, but may actually counter the gradient's efforts.

This can be seen in figure 4. While the left part of the grid gets the expected multiplier (penalizing the parts close to the disconnected component and progressively increasing the bonus as we get closer to the initial connected component), we notice that the right part just extends these multipliers in *trails*, and it proves particularly problematic in the downright corner. Indeed, here, this corner will get a substantial boost in priorities, that we have no reason to desire: it will just cause unnecessary future conflicts with the bottom robots.

3) Weight of the initial decisions: Another concern on the original implementation is that the successive multipliers (m_j and $C_{i|x,y}$) are cumulative. In order

[2]However, we might be tempted to consider these partial derivatives as a kind of averaged derivative. Such a construction has been tried and judged inconclusive.

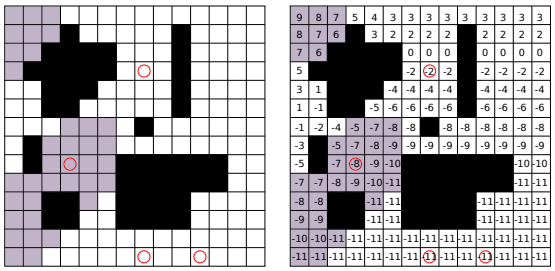


Fig. 4: An example of cells assigned to the leftmost robot, and the ensuing $C_{i,j}$ matrix with horizontal trails of equal $C_{i,j|x,y}$.

to explicit the problem raised, we will introduce the intentionally vague notions of *stable* and *disputed* cells. The formers are such that one of the E_i is notably smaller than the other ones in this cell. That is, this cell will not change attribution before a certain number of iterations. On the contrary, disputed cells are the cells that frequently change assignation: two or more robots are fighting over their control. In the case of an instance with both stable and disputed cells (which happens most of the time, as the different cells take different amounts of time to stabilize), the stable part will often be directly close to the robot, and thus see its coefficient reduced each step by C_i: the priorities of r_i in these cells will quickly become minuscule.

This has two notable downsides. The first one is that these numbers quickly become cumbersome to handle: as we want to compare them to each other, having such orders of magnitude in priorities seems quite futile. It even implies an attention as to the limits to floats' size in our implementation, which we would be glad to ignore.

The second one, much more impacting, is the inertia it gives to the iteration process. It indeed means that in most instances, the implied cells will not ever change assignation, as in the meantime during which the difference to the required threshold may gradually decrease (which will already be quite long due to the values' sizes), other cells (the disputed ones, generally) will change assignation first, thus inflecting the m_i's sign (and by extension the reduction of these first values). An example of such a behavior can be seen in figure 5.

V. Adapting DARP for mwCPP

A. Reducing the inertia through rooting

A solution we found to the issue reported in IV-A.3 is to regularly scale down the values towards 1. The idea is that while keeping the same total order between priorities – thus not interfering with the current assignation – it allows reducing the gap between the different priorities, therefore making the extreme value more accessible to the other robots. This has been done by turning all the coefficients of the E_i's to a power $\beta < 1$ with a fixed period. We do not want to do it at each iteration in order

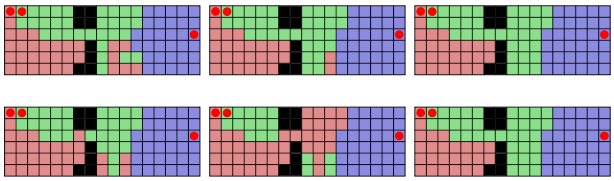


Fig. 5: Six successive iterations of DARP, where due to the bottleneck the right-hand side is contested, while we would want DARP to instead focus on the left-hand side.

not to ruin the gradient's progress, yet act frequently enough to have significance in most instances. Doing so also prevents these coefficients from becoming too big (as turning to this power $\beta < 1$ will eventually overpower the multiplications by 30 multipliers close to 1), which answers another of our remarks. We observed that this change allows getting rid of many divergence cases and accelerates the overall program, as seen in VI-B.1

B. Random stabilization of contested cells

Among the different situations encountered that caused a slower convergence, a recurrent situation is what we denote as a *bottleneck*: some robots are competing over which of them will secure a path through a passage not large enough for all of them to fit, as in figure 5.

We notice that a great number of iterations is then "wasted" on trying to reach a connected configuration, as the evolutions rarely form clear connected paths, but more often random-like distribution of assignation.

In order to counter this behavior, we introduced an additional heuristic: the contested cells now have a small chance at each iteration to see their value in $E_{i_0,j}$ halved (with i_0 their current assignation). That is, we enforce that some cells remain assigned to their robot for some time span, in order to incentivize the formation of such connected paths. The results linked to this modification can be found in VI-B.2.

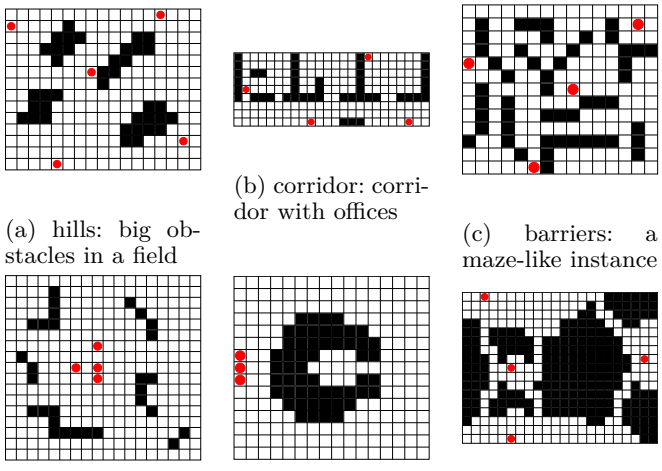
C. Altering the connectivity multiplier

As seen in section IV-A.2, the current way to compute C_i may be improved. However, our attempts to do so yielded some results, but these seemed too situational to infer a more general behavior. A first attempt was made in the optic of reducing the influence of C_i in far-away cells: currently, the influence of the initial connected component is equally strong everywhere in the "trails". We would, however, want this influence to reduce with the distance. Therefore, we define instead

$$C'_{i,j|x,y} := \min_{r\in\mathcal{R}_{i,j}} (||[x,y]-r||)^\alpha - \min_{q\in\mathcal{Q}_{i,j}} (||[x,y]-q||)^\alpha \quad (5)$$

with $\alpha < 1$. The results as to these functions can be found in VI-B.3.

A few other kinds of functions have been tried to define C_i differently, for example to give more weight to the regions in-between the different connected components, but the results have not been deemed satisfying.



(a) hills: big obstacles in a field

(b) corridor: corridor with offices

(c) barriers: a maze-like instance

(d) windmill: big obstacles and clustered robots

(e) cave: a bottleneck without clear assignation

(f) konigsberg: nearly isolated areas

Fig. 6: The most recurrent test layouts, in their unweighted version. Black cells still represent obstacles, and red circles the robots' initial positions. A last recurrent instance is the empty grid, with robots either evenly spread, or only spread in the bottom row.

VI. Results

In this section, we present the results obtained with the proposed approach for both weighted and unweighted scenarios, and their comparison with the standard version of the DARP algorithm. The code used to generate these results can be found at `https://gitlab.aliens-lyon.fr/Idir/darp-re-implementation`.

A. Methodology

1) Test files: A set of tests has been designed, in order to model the different situations in which the robots are usually deployed. The exact layout of some instances can be observed in figure 6.

Most of these layouts have been declined in 3 versions: unweighted, with clustered weights, and randomly spread weights. The full set of test files used in this work can be found at `https://gitlab.aliens-lyon.fr/Idir/darp-re-implementation/-/tree/main/tests_txt`. This head directory, labeled "test set", consists of the set of usually run tests during the initial process.

2) Chosen indicators: When looking at the results obtained with a set of parameters, the statistical indicators chosen to portray the efficiency are the average (A), the standard deviation (SD), and the geometrical average (GA). The reason behind this last choice is that due to the uncertainty of convergence, it is frequent that a slight tweak changes the number of iterations on a single "hard" instance (for example with a bottleneck) from 1000 to 2000, while speeding the other instances (taking around 100 iterations) by a factor 1.2. We would then want to notice such a change as possibly beneficial (as we will

always have little control as to the convergence on hard instances, but an acceleration on most other instances represents a significant improvement). The geometrical average, taking directly into account these acceleration factors, permits a better representation of the desirability of this set of parameters. Will also be indicated the cumulated number of divergences (D) and of suboptimal solutions (S) over the 185 runs for each set of tests.

The standard number of iterations is $X_0 = 50.000$, that is, DARP will try to find an optimal solution for the first X_0 iterations (i.e., the maximum difference d authorized between the $|\mathcal{L}_{i,j}|$ equals $\lceil \frac{W}{n_r} \rceil - \frac{W}{n_r}$), then look for a slightly less-than-optimal solution ($d \leftarrow d+1$) for $X_1 = \frac{X_0}{2}$ iterations, and so forth, up to $d = max(2, max_{c \in \mathcal{L}}(w_c))$. Therefore, the total iteration number will never go above $2X_0$. In case of non-convergence, we chose arbitrarily to set the number of iterations to $3X_0$, to set a clear penalty in the statistical analysis.

Lastly, due to the presence of random noise (both in the initial implementation and in some modifications we made), each specific instance is run five times with different seeds, and the result remembered for this instance will be the average of the required numbers of iterations.

3) Confirmation set: A set of "confirmation tests" has been designed, on which the different combinations have not been tried before the last test phase. Its goal, just as its machine learning counterpart, is to avoid the choice of parameters to have been too biased by the "test set" used when adjusting the parameter combinations. This set can be found at `https://gitlab.aliens-lyon.fr/Idir/darp-re-implementation/-/tree/main/set_confirmation`.

All the results described in the following will consist of the aforementioned indicators over the union of these two sets, described as the "verification set". The verification set is composed of 37 different instances, half of which are weighted.

B. Results of the different modifications

A first minor change made to DARP has been to increase the potency of the connectivity multiplier. That is, the coefficient μ defining how the $C_{i,j|x,y}$ may differ from 1. Initially set to 0.01, we have found it more efficient than to be set at a higher value, despite adding some more instability. In the following, unless said otherwise, we have set $\mu = 0.06$.

1) Scaling down the priorities: Table II presents the impact of the scaling down proposed in V-A. The results show an undeniable impact of this scaling down, as even a β close to 1 allows avoiding many divergences. If β gets too small, however, the added instability prevents us from converging, as most beginnings of solutions are set back every so often. The optimal value of β, at least over this set, seems to lie around 0.8. In the following, we will thus test our other parameters with values of β neighboring 0.8. The additional time such a computation brings is negligible compared to the overall acceleration observed, as we merely do a linear number of operations every 30 iterations: this tweak therefore brings a notable acceleration - in addition to preventing most multiplication overflows.

β	A	GA	SD	D	S
1	18294	1002	40158	15	18
0.95	10841	434	36134	12	1
0.9	6986	420	25886	7	0
0.85	3209	379	8684	1	3
0.8	3052	297	11670	2	1
0.75	2910	377	8525	0	0
0.7	6165	414	25054	6	0
0.65	9527	408	34455	11	0
0.6	9795	449	34405	11	2
0.55	10411	491	34414	10	2

TABLE II: The results in number of iterations over the verification set, with different values of β, and such an operation taking place every 30 iterations.

frequency	β	A	GA	SD	D	S
10^-3	0.9	3142	344	14049	2	1
3.10^-3	0.9	3524	330	14808	2	3
10^-2	0.9	6933	327	27760	7	1
10^-3	0.85	1681	328	4220	0	0
3.10^-3	0.85	4860	338	17185	3	3
10^-2	0.85	6851	315	27725	7	1
10^-3	0.8	1132	269	3693	0	1
3.10^-3	0.8	6015	326	23304	6	2
10^-2	0.8	6416	337	26113	6	1
10^-3	0.75	3682	356	15006	2	0
3.10^-3	0.75	4146	311	19990	5	0
10^-2	0.75	5046	297	24644	5	0
10^-3	0.7	3516	311	17042	3	0
3.10^-3	0.7	5288	320	24848	5	0
10^-2	0.7	6849	344	27674	7	2
10^-3	0.65	4848	283	26436	5	0
3.10^-3	0.65	5640	275	25537	6	1

TABLE III: The results in number of iterations, depending on the frequency and threshold of this stabilization.

2) Stabilizing contested cells: In this section, we will discuss the impact of the random stabilization of contested cells, proposed in V-B. We define a cell as contested if, in the last 10 iterations, its assignation changed at least 6 times. In the table III presenting the corresponding results, frequency relates to the probability for a contested cell to be stabilized in its current assignation.

We observe that the overall number of iterations is greatly reduced, but having too frequent such stabilization seems to bring unwanted general fluctuation, as the different regions fail to be clearly defined. This is most notable when the robots start close to each other, as the different robots struggle longer to define their respective regions, due to these random (and thus possibly unwanted) stabilizations. Therefore, we may want to restrict this change to the situations where the robots do not start too close to each other.

Another issue with this change is that the computation proves more costly than before, mainly in the hardest instances, where many cells are contested every turn. When running the verification set with parameters such

α	frequency	β	A	GA	SD	D	S
0.8	0	0.9	9407	428	29355	1	6
0.75	0	0.85	7332	347	25769	5	3
0.8	0	0.7	8718	316	34268	10	0
0.75	$3 * 10^-3$	0.7	5156	288	24785	5	0
0.75	$3 * 10^-3$	0.65	4920	290	24626	5	0
0.75	10^-3	0.85	5272	309	19684	0	2
0.8	10^-3	0.7	5722	290	25145	5	1
0.75	10^-3	0.65	6183	360	25773	6	0

TABLE IV: The results in number of iterations, depending on the power α in the connectivity multiplier.

that the average iteration number neighbors 2300, the time taken with or without stabilization is more or less the same: 3963 seconds instead of 3584 (in the same computer configuration). However, when the average number of iterations grows around 4500, we reach a computation time of 10,000 seconds. The reduction in the number of iterations still makes this change noteworthy, especially as the increased computation time can be easily dealt with, with some parallelism.

3) Connectivity correction with reduced trail: In this section, we will discuss the impact of the random stabilization of contested cells, proposed in IV-A.2, with α playing the aforementioned role.

We observe that, while this change does not seem to bring major stabilizations, compared to the base connectivity, it still tends to even out the geometrical average. Another element, less visible in this small table, is that such an alteration tends to perform up to twice as slowly in the presence of the bottleneck. Therefore, due to the overall unchanging results, it proves definitely more efficient in instances more sparse in obstacles, which seems logical as to the issue raised with trails: while reducing the gap between values hampers the overall stability, it allows not to arbitrarily favor the regions affected by the trails, which is particularly notable in sparse instances.

VII. Conclusion

In this paper, we tackled the mwCPP, a variation of the classic mCPP where weights are introduced to represent different coverage times for each cell in the environment. To solve this problem, we presented a solution based on a modified version of the DARP algorithm. The proposed improved version cannot only solve mwCPP, a problem for which we did not find any previous solver, but also does so while converging in most natural cases. It is able to converge with a considerable acceleration compared to the initial situation, of respectively a ratio of 6 and 3 for the average and geometrical average. This acceleration, while much less notable, still persists in the unweighted cases. We however need to note that there still remain differences in the convergence speed between the unweighted and weighted case of a same grid, though noticeably reduced. We could consider a more localized approach around the weighted cells to reduce this gap.

This work has also brought some first reflections as to the situations where DARP fails to converge even for standard mCPP scenarios. While we have not been able to characterize them extensively nor settle on a definitive criterion, the presence of bottlenecks or the proximity of the robots' initial positions seem to be reoccurring in all the situations of divergence observed. A more extensive analysis might prove helpful to a more definitive characterization of these edge cases, and how to possibly deal with them.

VIII. Acknowledgment

The authors would like to thank Athanasios Kapoutsis for the fruitful discussions and the help with the implementation and use of the DARP algorithm.

A part of this work has received support from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871260.

References

[1] W. Maes and K. Steppe, "Perspectives for remote sensing with unmanned aerial vehicles in precision agriculture," *Trends in Plant Science*, vol. 24, 12 2018.

[2] W. Jing, D. Deng, Y. Wu, and K. Shimada, "Multi-uav coverage path planning for the inspection of large and complex structures," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020, pp. 1480–1486.

[3] J. Sun, B. Li, Y. Jiang, and C.-Y. Wen, "A camera-based target detection and positioning uav system for search and rescue (sar) purposes," *Sensors*, vol. 16, p. 1778, 10 2016.

[4] X. Zheng, S. Jain, S. Koenig, and D. Kempe, "Multi-robot forest coverage," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 3852–3857.

[5] T. Cabreira, L. Brisolara, and P. Ferreira, "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, no. 1, p. 4, 2019.

[6] N. Hazon and G. Kaminka, "Redundancy, efficiency and robustness in multi-robot coverage," in *Robotics and Automation*, 2005, pp. 735 – 741.

[7] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 77–98, 02 2001.

[8] N. Agmon, N. Hazon, and G. Kaminka, "Constructing spanning trees for efficient multi-robot coverage," in *IEEE International Conference on Robotics and Automation*, 2006.

[9] W. Dong, S. Liu, Y. Ding, X. Sheng, and X. Zhu, "An artificially weighted spanning tree coverage algorithm for decentralized flying robots," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 4, p. 1689–1698, 2020.

[10] A. C. Kapoutsis, S. A. Chatzichristofis, and E. B. Kosmatopoulos, "Darp: Divide areas algorithm for optimal multi-robot coverage path planning," *Journal of Intelligent & Robotic Systems*, no. 86, pp. 1–18, 2017.

[11] H. Choset, "Coverage for robotics – a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1–4, pp. 113–126, 2001.

[12] K. Joshi, "Seafloor mapping and localization for a multi-robot system using route optimization algorithms," Ph.D. dissertation, Madras Institute of Technology, 06 2020.

[13] C. Gao, Y. Kou, Z. Li, A. Xu, Y. Li, and Y. Chang, "Optimal multirobot coverage path planning: Ideal-shaped spanning tree," *Mathematical Problems in Engineering*, vol. 2018, pp. 1–10, 2018.

[14] P. Grippa, A. Renzaglia, A. Rochebois, M. Schranz, and O. Simonin, "Inspection of Ship Hulls with Multiple UAVs: Exploiting Prior Information for Online Path Planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.

[15] S. J. Wright, "Coordinate descent algorithms," *Mathematical programming*, no. 151, pp. 3–34, 2015.