Theorem 1. The UMCP problem is \(\mathcal{NP}\)-Hard.

Proof. Clearly, this problem is in \(\mathcal{NP}\), since one can easily guess the coverage path of the robot and then verify its probability of surviving it in polynomial time. To prove its \(\mathcal{NP}\)-hardness, we use a reduction from the Hamiltonian path problem on grid graphs. A grid graph is a finite node-induced subgraph of the infinite two-dimensional integer grid (see Figure 1 for an example of a general grid graph). The Hamiltonian path problem on grid graphs (i.e., the construction of a path that visits every node of the grid graph precisely once) has been proven to be \(\mathcal{NP}\)-complete in [2].

![Fig. 1. An example of a general grid graph](image)

Given an instance of the Hamiltonian path problem, we construct an instance of the uniform-threat safest coverage problem as follows. Let us denote the grid graph in the Hamiltonian path problem instance by \(G\). We now create a target area for the safest coverage problem and denote it by \(T\). For each node in the grid graph \(G\) we create a cell with a threat point of probability \(p\) in \(T\). All the
other cells in T will contain obstacles. This completes the construction, which can be done in polynomial time.

We claim that there exists a Hamiltonian path in G, if and only if the safest coverage path in T visits each threat point exactly once.

First direction - if there exists a Hamiltonian path in G, then there exists a path in T that visits each threat point exactly once. This path covers the target area T completely (since T contains only threat points and obstacles). Clearly, this is also the safest coverage path (or one of the safest coverage paths if there exist more than one). Thus, the safest coverage path in T visits each threat point exactly once.

Second direction - if the safest coverage path in T visits each threat point exactly once, then there exists a path in G that visits each node in the grid graph exactly once, i.e. there exists a Hamiltonian path in G.

Therefore, we can find if there exists a Hamiltonian path in a given grid graph G, by constructing T, finding the safest coverage path in T and then checking if it covers each threat point exactly once. Thus, the uniform-threat maximize coverage completeness coverage problem is \( \mathcal{NP} \)-hard.

**Theorem 2.** The UMEC problem is \( \mathcal{NP} \)-Hard.

*Proof.* The same construction from the previous proof can be used to prove the \( \mathcal{NP} \)-Hardness of the UMEC problem, since the target area T that was constructed there contains only threat points and obstacles. Thus, a coverage path of T with maximum expected coverage percentage is also a coverage path that contains minimal number of threat points.

**Theorem 3.** Let \( d \) be the total number of dangerous cells in the accessible grid. Let \( b \leq d \) be the total number of dangerous boundary cells and \( c \leq d \) the number of dangerous connecting cells. Then STAC covers the given grid using a path that contains \( x \leq d + b + 2c \) dangerous cells.

*Proof.* First, we should note that dangerous cells are added to the coverage path if they reside on the connecting route between different areas (either safe or dangerous areas), or when they are part of the coverage of a dangerous area.

Let us begin by counting the number of dangerous cells the robot needs to pass through when moving between the safe areas. The connecting route between the safe areas is found by the Christofides approximation algorithm to TSP. This algorithm is based on creating an MST (Minimum Spanning Tree) of the graph and then adding some edges to the graph in order to create an Euler circuit. Just by traversing the MST twice, the tour will be within a factor of 2 of the optimal TSP solution. Thus, the route produced by the Christofides algorithm cannot traverse the connecting dangerous cells more than twice.

Let \( k \) be the number of safe areas, and let \( S_1, ..., S_k \) be the order of their visit as determined by STAC. Now let us denote the number of connecting dangerous cells between two safe areas \( S_i \) and \( S_{i+1} \) by \( s_i \) (\( 1 \leq i \leq k - 1 \)). In the worst case scenario, the robot has to traverse all the dangerous cells in the separating area between \( S_i \) and \( S_{i+1} \). Such a scenario is illustrated in Figure 2. In this map
the separating area between the upper and the lower safe areas has an S-shape form, composed of alternating rows of dangerous cells and obstacles. In this case the robot must traverse all the dangerous cells in the separating area in order to move between the safe areas.

![Diagram of an environment where the robot must traverse all the dangerous cells between the safe areas.](image)

**Fig. 2.** An example for an environment where the robot must traverse all the dangerous cells between the safe areas.

Thus, the total number of dangerous cells the robot needs to traverse between safe areas is bounded by:

\[
c_S = \sum_{i=1}^{k-1} s_i \leq d
\]

(1)

where \(c_S\) is the number of dangerous cells that connect safe areas.

Next, we count the number of visits to dangerous cells during the coverage of the dangerous areas. Note that the dangerous cells that have already been visited during the coverage of the safe areas are not covered again. Thus the number of dangerous cells that need to be covered in this stage is \(d - c\).

Now let us denote the number of dangerous areas by \(r\), and let \(D_1, \ldots, D_r\) be the order of their visit as determined by STAC. Let us denote the number of dangerous boundary cells in each dangerous area \(D_i\) by \(b_i\) (\(1 \leq i \leq r\)). By theorem 1 in [1], the total number of repetitive coverages in Spiral-STC is bounded by the number of boundary subcells in the work-area grid. Since STAC runs Spiral-STC on each dangerous area separately, the total number of visits to dangerous cells during the coverage of the dangerous areas is:

\[
\sum_{i=1}^{r} (|D_i| + b_i) \leq d - c_S + b
\]

(2)

The inequality is due to the fact that some of the boundary dangerous cells may already have been visited during the coverage of the safe areas.

Finally, the robot might need to pass through already visited dangerous cells in order to move between different dangerous areas. In addition, it might have to pass through dangerous cells when moving from the last safe area to the first dangerous area. Let us denote the number of connecting dangerous cells between two dangerous areas \(D_i\) and \(D_{i+1}\) by \(d_i\) (\(1 \leq i \leq r - 1\)) and the number
of connecting dangerous cells between the last safe area and the first dangerous area with \( d_0 \). Again, in the worst-case scenario, the robot will have to traverse the entire separating dangerous area before it can move to the next unvisited dangerous area. Thus, the total number of dangerous cells the robot needs to traverse between dangerous areas is bounded by:

\[
c_D = \sum_{i=0}^{r-1} d_i \leq d
\]

where \( c_D \) is the number of dangerous cells that connect dangerous areas. As before, Christofides algorithm cannot traverse these connecting dangerous cells more than twice.

Thus, the total number of repetitive coverages of dangerous cells is at most

\[
2c_S + d - c_S + b + 2c_D = d + b + c_S + 2c_D
\]

If we denote by \( c = c_S + c_D \) the number of connecting dangerous cells between different areas, then the total number of dangerous cells along the coverage path generated by STAC satisfies \( x \leq d + b + 2c \).

**Theorem 4.** Let \( n \) be the total number of cells in the accessible grid. Let \( m \leq n \) be the total number of boundary cells and \( r \leq n \) the number of connecting cells. Then STAC covers the given grid using a path that contains \( y \leq n + m + 2r \) cells.

**Proof.** The proof is similar to Theorem 4. By theorem 1 in [1], the total number of repetitive coverages in Spiral-STC is bounded by the number of boundary cells in the work-area grid. Thus, the total number of visits to cells during the coverage of the connected areas is at most \( n + m \). In addition, cells may be revisited along the connecting route between different areas (either safe or dangerous areas). The connecting route is found by the Christofides approximation algorithm to TSP. This algorithm is based on creating an MST (Minimum Spanning Tree) of the graph and then adding some edges to the graph in order to create an Euler circuit. Just by traversing the MST twice, the tour will be within a factor of 2 of the optimal TSP solution. Thus, the route produced by Christofides cannot traverse the connecting cells more than twice. Thus, the coverage path generated by STAC contains at most \( n + m + 2r \) cells, where \( m \leq n \) and \( r \leq n \).

**Theorem 5.** Let \( d \) be the total number of dangerous cells in the accessible grid. Then GSAC covers the given grid using a path that contains at most \( 2d \) dangerous cells.

**Proof.** Consider the outgoing edges from a dangerous cell \( c \). The first time the algorithm traverses such an edge, it is guaranteed that it will cover all the cells that are accessible from that edge before it returns back to \( c \), since all the other cells in the grid will have longer paths from the current position of the robot. Thus, each edge connected to \( c \) is traversed at most once in each direction. Consequently, there are at most \( 2d \) edges that touch dangerous cells along the coverage path. Hence, the coverage path contains at most \( 2d \) dangerous cells.
Theorem 6. Let $n$ be the total number of cells in the accessible grid. Then GSAC covers the given grid using a path that contains at most $4n$ cells.

Proof. The outline of the proof is similar to the proof of Theorem 7. Let us examine the outgoing edges from a given cell $c$. The first time the algorithm traverses one of these edges, it is guaranteed that it will cover all the safe cells that are accessible from that edge before it returns back to $c$, since all the other cells in the grid will have longer paths from the current position of the robot. However, the dangerous cells which are accessible from that edge will be covered only after all the safe cells, which are accessible via the other edges that connect to $c$, are covered. Thus, the robot may traverse the same edge again in order to cover all the dangerous cells which are accessible from it. After traversing the same edge for the second time, it is guaranteed that the robot will cover all the cells accessible from that edge which have not been visited yet, before returning back to $c$. Thus, each edge connected to $c$ is traversed at most twice in each direction. Consequently, the number of edges along the coverage path is at most $4n$. Hence, the coverage path generated by GSAC contains at most $4n$ cells.

We now provide the proofs of the lemmas mentioned in the paper.

Lemma 4.1 (completeness) Algorithm STAC creates a path that covers every free cell accessible from the starting cell $s$.

Proof. We will first prove that CSCP is complete, i.e., that it covers all the cells in its input $C$. In line 3 of the procedure, CSCP splits the cells in $C$ into $k$ connected areas $R_1, \ldots, R_k$. Each of these areas is covered by Spiral-STC (line 6), which is known to be complete [1]. Since all cells in $C$ are reachable from the starting cell (this is a pre-condition on the input), there must be a connecting route between all the areas $R_i$, which is found in line 14 of the procedure. Thus, CSCP creates a path that covers all the cells in its input $C$.

Subsequently, STAC is complete, since every free cell accessible from the starting cell $s$ is included in either the safe cells group $S$ (line 3) or in the group of dangerous cells $D$ (line 6), which are both covered by CSCP.

Lemma 4.2 Let $n$ be the total number of free cells accessible from the starting cell $S$, and $a$ be the number of connected areas. Then STAC covers the given area in $O(a^2 n \log n + a^3)$ time.

Proof. We will first analyze the running time of the CSCP procedure. Let us assume that its given group of cells $C$ contains $n$ cells and $a$ connected areas. In the first stage, CSCP runs DFS on the graph induced from $C$. The time complexity of DFS is linear in the size of the cells graph, which in this case is $O(|E|) = O(n)$, since the graph is sparse. In the second stage, it finds a coverage path for each connected area by running Spiral-STC on it. By lemma 4.1 in [1], Spiral-STC covers a region of $n$ cells in $O(n)$ time. Since Spiral-STC is executed on each connected area separately, and the total number of cells in all these areas is $n$, the time complexity of this stage is $O(n)$. In the third
stage, CSCP runs Dijkstra for each pair of connected areas. The time complexity of Dijkstra in this case is $O(n \log n)$, since the induced graph is sparse. Thus, the total time complexity of this stage is $O(a^2 n \log n)$. In the last stage, we run the Christofides algorithm on the graph induced by the connected areas, which has a time complexity of $O(|V|^3) = O(a^3)$. Hence, the entire complexity of the CSCP procedure is $O(a^2 n \log n + a^3)$.

The STAC algorithm runs CSCP twice - once for each type of cells. In addition, it executes Dijkstra one more time on the induced graph of the grid cells. Hence, the entire complexity of STAC is $O(a^2 n \log n + a^3)$.

**Lemma 5.1** (completeness) Algorithm GSAC creates a path that covers every free cell accessible from the starting cell $s$.

*Proof.* All the reachable cells from the starting cell $s$ have a path that connects them. Thus, all of them will eventually become the node $v$ in line 7 of the algorithm, which is added to the coverage path returned by the algorithm.

**Lemma 5.2** Let $n$ be the total number of free cells accessible from the starting cell $S$. Then GSAC covers the given area in $O(n^2 \log n)$ time.

*Proof.* Since the graph is sparse ($|E| = O(|V|)$), running Dijkstra on the entire graph takes $O(n \log n)$ time, and the GSAC algorithm runs Dijkstra $n$ times.

We now provide the detailed pseudo-code of the CSCP procedure described in section IV of the paper.
procedure Create_Safe_Coverage_Path(G, C)
Input: a grid G and a group of reachable cells C
Output: a path P that covers all the cells in C
1: Create a new coverage path P
2: Build a graph $G_C$ whose nodes represent the cells in C and its edges connect neighboring cells in G
3: Find the connected components of $G_C$ using DFS and denote them by $R_1, ..., R_k$
4: for every area $R_i$, $1 \leq i \leq k$
5: $s_i \leftarrow$ the upper-left corner of $R_i$
6: Compute a coverage path for the cells in $R_i$ by running Spiral-STC from $s_i$ and denote it by $\pi_i$
7: $f_i \leftarrow$ the last node in $\pi_i$
8: Build a graph $G_W$ whose nodes represent all the cells in the grid G and its edges connect neighboring cells in G with the weights as defined by formula (3)
9: for every node $f_i$, $1 \leq i \leq k$
10: Run Dijkstra on the graph $G_W$ starting from $f_i$
11: for every node $s_j$, $j \neq i$ do
12: Find the safest path $f_i \rightarrow s_j$ by traversing the shortest paths tree created by Dijkstra's algorithm
13: Build the graph $G_R$ whose nodes represent the areas $R_i$ and its edge weights are determined by the lengths of the paths connecting them
14: Run a TSP approximation algorithm on $G_R$ to find the safest possible route that connects the areas $R_i$, and arrange them by the order of their visit $R'_1, ..., R'_k$
15: for every area $R'_i$, $1 \leq i \leq k$
16: Add the coverage path $\pi'_i$ of $R'_i$ to P
17: if $i < k$ then
18: Add the connecting path $f'_i \rightarrow s'_{i+1}$ to P
19: return P
References