Constructing Optimal Cyclic Tours for Planar Exploration and Obstacle Avoidance : A Graph Theory Approach

Abhishek Tiwari, Harish Chandra, Jacob Yadegar, and Junxian Wang

Utopia Compression Corporation
11150 West Olympic Boulevard, Suite 1020,
Los Angeles, CA-90064
{abhishek, harish, jacob, junxian}@utopiacompression.com

Abstract. We propose a graph theory approach for planar autonomous exploration. We first partition the planar space using Peano-Cesaro triangular tiling and then construct an edge adjacency dual graph of the tiling pattern. The dual graph of the Peano Cesaro triangulation is obtained by defining a vertex for each triangular tile and drawing an edge between two tiles that share an edge. In the presence of obstacles we analyze the subgraph induced by the non-obstacle tiles in the dual graph. We prove the existence of Hamiltonian cycles in this induced subgraph for a certain class of obstacles. We also prove the non-existence of Hamiltonian cycles for certain other obstacle configurations. We present heuristic based algorithms and compare their results for the cases where we have a definitive answer to the existence of Hamiltonian cycles. Examples with figures are included to illustrate the concept.

1 Introduction

Exploratory path planning with obstacle avoidance finds application in areas involving autonomous search/exploration tasks like mine sweeping, rescue operations, locating survivors in a disaster struck area, ocean exploration, monitoring coast lines, protecting borders etc. Some other exciting commercial applications of interest involve autonomous coverage applications for lawn mowing and vacuum cleaning robots.

Autonomous path planning and obstacle avoidance has been studied by numerous researchers and over the years quite a few interesting approaches have been proposed. [7] and [10] present a comprehensive treatment of these approaches. In this chapter, we bring together concepts from the fields of space filling curves, graph theory and path planning and merge them to gain insights into the optimal solutions of a problem of considerable practical interest. We propose a graph theory approach to the exploratory path planning problem. The existing approaches are based mainly on heuristic and there exist no provable guarantees. Graph theory has been applied for path planning problems earlier in [9] by Jun and D'Andrea, the difference is that the authors use hexagonal cells which are not well suited for multi-resolution decomposition and it is not clear whether an irregular decomposition of the exploration space is possible. In [14] and [15], the Savla et al. tile the exploration space with bead tiles. The bead tiles are constructed assuming a Dubin's vehicle model for the UAVs (with a maximum allowable radius of curvature of the planned paths). The bead tiles even though they encode the mobility constraints satisfactorily, but again there is no mention of irregular decomposition. We believe our approach of Peano-Cesaro tiling is more flexible in terms of representation of the exploration space. However, it remains to be seen how the novel ideas proposed in this chapter compare with the existing exploratory path-planning approaches in terms of actual implementation on a real system. The concept of Peano-Cesaro tiling used in this paper for path planning application, has been successfully used for pattern based image compression in our earlier work [5].

In this chapter, we use Peano-Cesaro sweep [13] to partition the territory map (including obstacles, if any) into triangular tiles and use the associated Sierpinski tour for exploration. The Peano-Cesaro sweep is a space filling heuristic. Space filling heuristics have been successfully used to solve the Traveling Salesman Problem in a time efficient manner [3], [12]. Even though the solutions obtained are suboptimal, the savings in computation time are immense. The authors in [3] have proved a bounded distance from the global optimum. The Sierpinski space filling curve has been proven to be optimal in terms of the overall tour length when compared to other space filling curves like the Hilbert curve [1].

We represent the mobility constraint of the autonomous vehicle by allowing moves only between tiles that share a side in one time step. We overlay the Peano-Cesaro tiling pattern with a graph, where the tiles are represented by vertices and the edges represent allowed moves. This graph is the same as the edge adjacency dual graph of the Peano-Cesaro triangulation. If the planar region has obstacles, in order to disallow movement to the obstacle tiles, we remove the obstacle vertices and edges incident on them from the dual graph. The resulting graph is a subgraph induced by the non-obstacle vertices of the dual graph. Our objective then is to find a cyclic tour of the induced subgraph such that it includes all the non-obstacle tiles. In practical implementation, this is equivalent to exploring a planar region with obstacles in the most efficient manner, so that it is possible to visit all the non-obstacle regions and also save fuel, time and energy.

A no repetition cyclic tour of the vertices of a graph is known as a Hamiltonian cycle. Thus, an optimal exploration tour, in the presence of obstacles, is a Hamiltonian cycle in the subgraph induced by the non-obstacle tiles in the dual graph. In general, determining whether a graph is Hamiltonian is an NP-complete problem. But, in this paper we use the special properties of the induced dual subgraph to prove existence of Hamiltonian cycles for a certain class of obstacles. Similar work for Triangulated Irregular Networks has been done in [2], which has immense application in computer graphics.

The contribution of this chapter can be summarized under three specific headings: Firstly we have devised an algorithm called ESSENTIAL-CHAINS, which starts with an assumption that a given dual subgraph is Hamiltonian and then proceeds by logical reasoning collecting all components of the graph which provably are essential components of the Hamiltonian cycle, if one exists. For certain obstacle configurations the algorithm comes up with a counter example thus proving that the dual subgraph is non-Hamiltonian. Secondly we prove the existence of Hamiltonian cycles for a special class of obstacles. Lastly we have devised a few heuristics based algorithms to find the minimum repetition tours for a given obstacle configuration. We compare the result of the heuristics based algorithms, based on the attainable optimum established using our existence results.

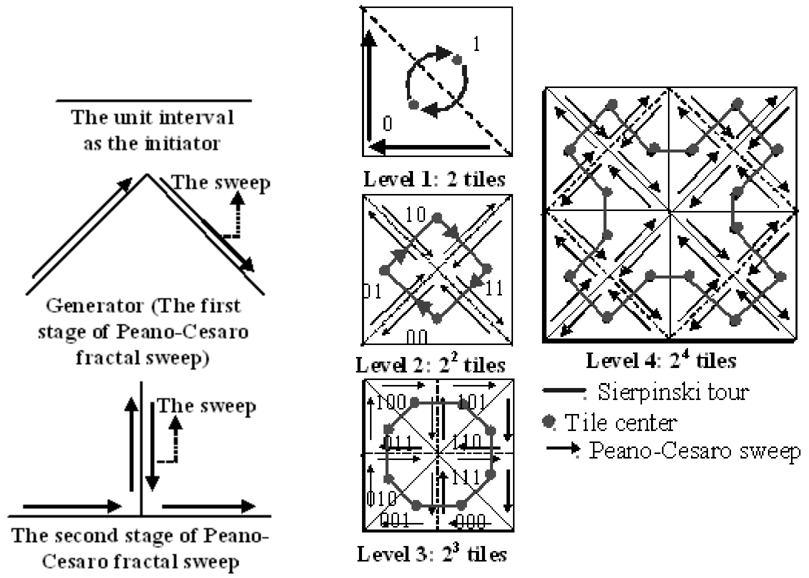
In section 2, we introduce the ideas of Peano-Cesaro sweep and the associated Sierpinski bucketing tour. In section 3 we use the concepts from section 2 and graph theory to pose a combinatorial optimization problem for exploratory path planning in the presence of obstacles. In section 4 we prove existence and non-existence results of Hamiltonian cycles for a certain class of obstacle configurations. In section 5 we present algorithms to find long cycles in our graph of interest. Finally in section 6, we sum up the contributions of this paper.

2 Peano-Cesaro Tiling and the Associated Sierpinski Bucketing Tour

2.1 Peano-Cesaro Fractal Sweep

The central idea behind a fractal sweep is to find a recursive mapping that takes the unit interval into the plane. The key concepts in such constructions are initiator, generator, sweep and rules of arms placement. An example of a fractal is the Peano-Cesaro fractal sweep as illustrated in figure 1(a). If production of the fractal, as in Peano-Cesaro fractal sweep, proceeds indefinitely, a trace that is everywhere continuous but nowhere differentiable would be obtained. The generator of a fractal consists of arms, each treated as scaled down initiators for the next stage of construction. Patterns that replicate the generator as recursion deepens are called self-similar [11] and admit to the concept of dimensionality $D = \log N/\log(1/r)$ where, N is the number of arms of the generator and r is the similarity ratio defined as the length of an arm of the generator to that of the initiator. Thus, the Peano-Cesaro curve, in which the generator has two arms ($N = 2$), each ($r = 1/\sqrt{2}$) factor of the length of the initiator, has dimensionality $D = \log 2/\log\sqrt{2} = 2$. Fractals defined over the plane with D close to 1 are smoother and better behaved than those with D close to 2, which are more plane-filling. Fractals with $D > 2$ exhibit chaotic behavior by multiply crossing regions trapped by the fractal sweep. Brownian motion is a primary example of sweep patterns with $D > 2$, while Koch curve [11] has $1 < D < 2$.

A *sweep* is a walk from the start to the end of the initiator along a defined path and this path defines the fractal. Taking one of the diagonals of the unit square $\mathcal{U}$ on the plane as the initiator, the Peano-Cesaro fractal sweep can be developed using the generator shown in figure 1(a), and the following rule of



(a) Peano-Cesaro fractal sweep. (b) Peano-Cesaro tiling and the associated Sierpinski bucketing tour.

Fig. 1. Peano-Cesaro fractal sweep and tiles

placement: **At every even (odd) stage $L = 1$, walk along $(L-1)$th sweep and place the generator to the right (left) of each and every arm.** Four stages of the Peano-Cesaro fractal sweep in figure 1(b) are indicated by directed paths all beginning and ending at a corner of the unit square $\mathcal{U}$. Note that the Peano-Cesaro sweep as illustrated in figure 1(b) shows no region crossing behavior. To be more accurate, the sweep is linear-wise degenerate, as it visits the vertices of the tiles generated multiple times - for instance the center of $\mathcal{U}$ in stage 2 of the decomposition is visited four times, which in fact can be proven to be the maximum degeneracy for all recursion levels. However, the sweep is planar-wise non-degenerate (non-crossing), precisely because $D = 2$.

The Peano-Cesaro sweep tiles the unit square $\mathcal{U}$ with right-angled isosceles triangles. A tile is composed of the current initiator forming the hypotenuse (denoted by $\mathbf{B}$), and the first and second arms of the corresponding generator as the other two sides (denoted by $\mathbf{F}$ and $\mathbf{S}$ respectively.) The number of tiles is doubled each time the sweep is advanced by one recursion stage (also referred to as recursion level), yielding 2^L triangular tiles at the Lth stage ($L \geq 1$). The domain $\mathcal{U}$ is decomposable to any desired degree and tiling is always regular and isotropic such that at every level of decomposition each tile is visited by the sweep (in the sense that it traverses the two ($\mathbf{F}$ and $\mathbf{S}$) sides of the tile defined by the arms of the generator). These are the properties that make the Peano-Cesaro sweep amenable to analysis of the neighborhood of a triangular tile and, as we shall discuss in this chapter, highly suitable for autonomous

Unmanned Vehicle (UV) path planning, object avoidance and also cooperative strategies for exploration and goal-seeking missions amongst a team of UVs. The decomposition procedure highlighted above can be represented by a binary tree structure wherein nodes on left (right) branches are recursively assigned the values 0(1). This, in a natural way, maps each tile with a binary code sequence indicating a tree descent inheritance from the root (representing the unit square $\mathcal{U}$) to tiles/nodes at various levels of the tree. Denoting a code sequence by C, a tile at the Lth recursion level may be expressed by $C = c_1 \cdots c_L$, where $c_i \in \{0, 1\}$, $i = 1 \cdots L$. Tiles at stages 1, 2 and 3 in figure 1(b) carry their code sequences. Later we use the decimal equivalents of these code sequences to distinguish tiles, we will refer to this decimal indexing as the *Sierpinski ordering.* This concept of distinguishing regions in space using indexing of the subdivisions has also been analyzed in [1], where it was proved that the Sierpinski ordering is the best in terms of preserving spatial adjacency information.

The Peano-Cesaro sweep in figure 1(b) shows four regular stages of tiling and as mentioned the isotropic and self-similarity properties of the tiles ensure that all regions of U at any recursion level have equal probability of being visited, where the probability measure is proportional to the area of the tile at the specific recursion level. These properties, which turn out to be important for single or cooperative autonomous path planning, are not met in most of the other fractal patterns - the reader is invited to consult [11] for a number of examples such as the snowflake, the monkeys tree and the dragon sweep formations. The regular tiling pattern in figure 1(b), though extremely efficient and simple, is not mandatory. Figure 2 shows two (non-homogenous) deviations of the regular decomposition. Figure 2(a) depicts an example of variable tile size Peano-Cesaro decomposition along with its associated Sierpinski sweep. The deviation in figure 2(b) is more drastic, where tile splitting is no longer constrained to the mid-point of B side, though the topological structure of the binary decomposition is left invariant.

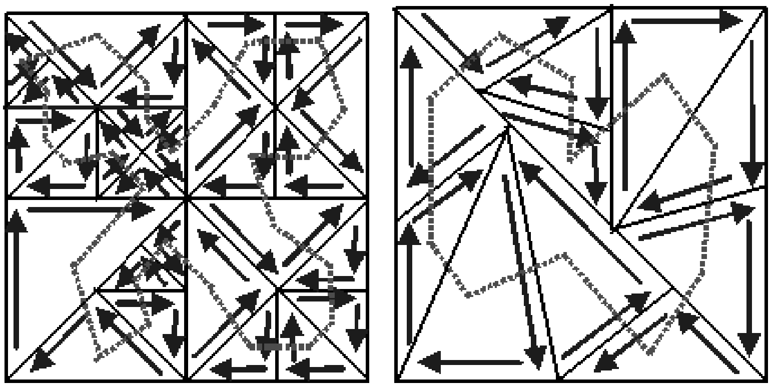
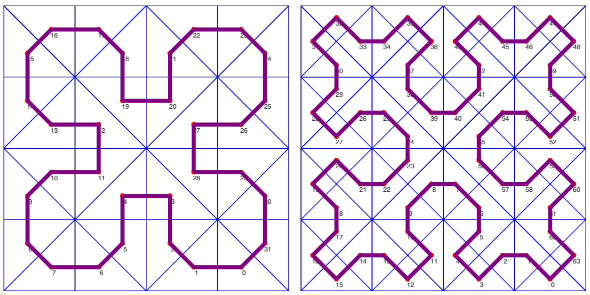

(a) Variable size decomposition. (b) Irregular adaptive decomposition.

Fig. 2. Non-homogenous Peano-Cesaro tiling

2.2 Sierpinski Bucketing Tour

The Peano-Cesaro sweep in figure 1(b) induces what is referred to as a Sierpinski Bucketing tour according to the following rule [8]:

Whenever a tile is swept by the Peano-Cesaro fractal sweep, connect the center of the tile to the center of the preceding tile visited by the Peano-Cesaro sweep.



(a) Sierpinski Tour for level 5. (b) Sierpinski Tour for level 6.

Fig. 3. Sierpinski Tour for decomposition levels 5 and 6

Application of the above rule to regular (and irregular) sweeps yields the regular (and irregular) Sierpinski tours. Figure 3 illustrates the Sierpinski tour for level 5 and 6. Sierpinski tour generation algorithm forms the basis for autonomous agent path planning whether on an exploratory or surveillance missions starting and ending at the same location or a goal oriented mission starting from a source location to a destination. The hierarchical (multi-resolution) nature of the Peano-Cesaro tiling, represented by sparse binary tree structures, entirely carries over to the Sierpinski tour. This hierarchical behavior is highly desirable for path planning missions in situations where the autonomous agent is required to probe more carefully and search finer grain territories while cruising at distance in open spaces (see figure 2(a)) for a scenario. Thus, if the granularity of tiles in the 3^{rd} stage of Sierpinski tour in figure 1(b) is not sufficient, the autonomous agent can easily penetrate one (or more) level(s) deeper into the tree and search the region according to the 4^{th} (or n^{th} in general) stage of the Sierpinski tour where granularity shrinks at a rate of two per level. Due to the fractal dimension of the Peano-Cesaro sweep $D = 2$, the Sierpinski tour drains $\mathcal{U}$ to any level given a sufficiently large n, where n is the recursion level. This property guarantees an exploratory tour from a point A and back to A, or a Hamiltonian chain from a point A to a point B, if such a path exists.

3 Exploring a Planar Region with Minimum Number of Repetitions in the Presence of Obstacles

In section 2 we proposed Peano-Cesaro tiling and the associated Sierpinski bucketing tour as tools to generate way points for a planar autonomous mobile agent. If the agent follows the Sierpinski tour, it visits each tile in the planar region and returns to its origin. The problem of generating exploration way points in an obstacle strewn space is of extreme practical interest. Consider for example, an autonomous patrol boat, to be used for exploration of a coastline which is full of small islands. For the purpose of this paper we assume that the territory map of the planar region is available to the autonomous agent. Application of the proposed tools to path-planning and obstacle avoidance with local sensors and in the presence of mobile obstacles is an avenue of current and future research. It is often the case that the capability of an autonomous agent is limited by it's battery life. For an exploration application it is desirable that the autonomous agent visits every tile in the planar space with obstacles. This mission should be accomplished with minimum repetitions of the tiles to extend battery life. In this section we pose a combinatorial optimization problem to find a cyclic tour that visits all the non-obstacle tiles, with minimum number of repetitions.

Let L be the level of decomposition of the planar region. This will result in the decomposition of the planar region into 2^L tiles. Let the tiles be numbered from 0 to $2^L - 1$ under the regular Sierpinski ordering, as defined in section 2. We will call the set of all tiles as $\mathcal{T}$. We now state our main assumption for this section.

Assumption 1 (Obstacle set). *Any obstacle in the planar region can be represented as a union of tiles. Let $\mathcal{O}$ be the set of tiles marked as obstacles. We call $\mathcal{O}$ the obstacle set.*

We need the above assumption to retain the mathematical structure of the problem. For exploration applications this assumption does not create any limitations as it is always possible (as shown in section 2) to set the decomposition level L sufficiently high such that the obstacle can be closely approximated by a set of contiguous tiles.

Constraint 1 (Allowed moves). *If the autonomous vehicle is at tile i at time instant t, then at time $t + 1$, it can only move to the side neighbors of tile i.*

This assumption conforms to the mobility constraints of unmanned surface vehicles. We now state the problem objective:

Given a set of covering tiles $\mathcal{T}$ and the obstacle set $\mathcal{O}$. Under constraint 1, find a cyclic tour that covers all the tiles in the set $\mathcal{T} - \mathcal{O}$ at least once, with minimum number of revisits of tiles.

In order to analyze the performance of any algorithm that tries to find a cyclic tour with minimum number of tile revisits, it is important that the minimum number of repetitions needed for a given obstacle configuration be known. In the following section we present and prove a few existence results.

4 Results on Existence and Non-existence of Hamiltonian Cycles

We analyze the optimization problem from section 3 using results from graph theory. We first create a edge adjacency dual graph. For each triangular tile we have a vertex in the dual graph and an undirected edge exists between two vertices if and only if it is possible to move between the corresponding tiles under Constraint 1. We refer to the set of all tiles in the plane as $\mathcal{T}$.

The following definition puts these concepts in a mathematically precise formulation.

Definition 1. *The* edge adjacency dual graph *is an undirected graph $\mathcal{S} = (V_{\mathcal{S}}, E_{\mathcal{S}})$, such that for every tile in $\mathcal{T}$, there is a vertex in $V_{\mathcal{S}}$ and for a pair of vertices $x, y \in V_{\mathcal{S}}$, $xy \in E_{\mathcal{S}}$ if and only if x and y are reachable from each other in one time step under Constraint 1.*

Note that the dual graph is a 2-connected graph and the vertices can have maximum degree 3. Since vertices in the dual graph represent tiles, in this section we will use the terms tile and vertex interchangeably. We now define Hamiltonian cycles and graphs.

Definition 2. *For a graph with more than two vertices, a* **Hamiltonian cycle** *is a cycle that contains each vertex of the graph exactly once. If a graph has a Hamiltonian cycle, it is called Hamiltonian.*

A cyclic tour of the planar region that visits all the tiles with no revisits is a Hamiltonian cycle in the associated dual graph.

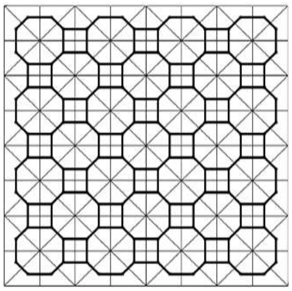
The Sierpinski tour, as defined in section 2, is a Hamiltonian cycle in $\mathcal{S}$. Thus the dual graph, $\mathcal{S}$, is a Hamiltonian graph. Now since we know that $\mathcal{T} - \mathcal{O} \subseteq V_{\mathcal{S}}$, $\mathcal{T} - \mathcal{O}$ induces a subgraph $\mathcal{S}'$ in $\mathcal{S}$. This can be represented as $\mathcal{S}' = \mathcal{S}[\mathcal{T} - \mathcal{O}]$ following the notation of [6].

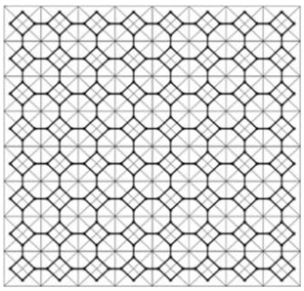
The existence of Hamiltonian cycle in $\mathcal{S}'$, implies that a cyclic exploration tour exists that visits all the tiles in the obstacle strewn planar region with no revisits.

The problem of verifying whether a graph contains a Hamiltonian cycle has been studied for over a hundred years. It is well known that the Hamiltonian-cycle problem is NP complete [4]. We prove existence and non-existence of Hamiltonian cycles for different obstacle configurations using special properties of the dual graph. The optimum of the combinatorial optimization problem described earlier, is zero repetitions in the case a Hamiltonian cycle exists. Once we know that a Hamiltonian cycle exists, we can evaluate the distance from optimum for heuristic based algorithms.

Definition 3. *A* **simple cycle** *is a cycle without any chords. Here, a chord is defined as an edge that joins two vertices of a cycle but is not itself an edge of that cycle.*

As evident in figures 4(a) and 4(b), simple cycles in the dual graph either have 4 or 8 nodes. Such simple cycles look like squares or octagons respectively.


(a) $\mathcal{S}$ for $L = 7$.


(b) $\mathcal{S}$ for $L = 8$.

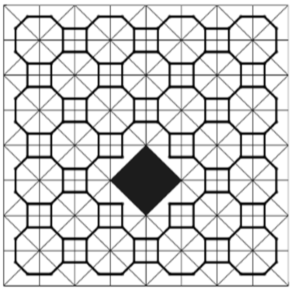

(c) Induced subgraph $\mathcal{S}'$ in the presence of an obstacle.

Fig. 4. $\mathcal{S}$ and $\mathcal{S}'$

Definition 4. *All tiles that share an edge or a vertex with the boundary of the square region will be referred to as the* **periphery tiles**. *We will call all the other tiles as* **interior tiles**.

Our first result establishes non-existence of Hamiltonian cycles when the number of tiles in the obstacle set is odd.

Proposition 1 (Properties)

1. *If $L > 1$, $\mathcal{S}$ is 2-connected.*
2. *$\mathcal{S}$ and $\mathcal{S}'$ are bipartite graphs.*
3. *$|\mathcal{O}|$ is odd then $\mathcal{S}'$ is non-Hamiltonian*

Proof

1. $L > 1$ implies $|\mathcal{T}| = |V_{\mathcal{S}}| > 2$. One needs to remove at least two vertices from $V_{\mathcal{S}}$, such that the subgraph induced by the remaining vertices on $\mathcal{S}$ is disconnected. Hence $\mathcal{S}$ is 2-connected.
2. All cycles in $\mathcal{S}$ have even number of vertices, therefore by proposition 1.6.1 in [6] we know $\mathcal{S}$ is a bipartite graph. $\mathcal{S}'$ being the induced subgraph retains the bipartite property.
3. If $\mathcal{S}'$ is disconnected or 1-connected, then the results hold trivially. If $\mathcal{S}'$ is 2-connected, then because $|\mathcal{O}|$ is odd, $|V_{\mathcal{S}'}| = |\mathcal{T} - \mathcal{O}|$ is also odd. Now if a Hamiltonian cycle exists it will have an odd number of vertices in it, which is a contradiction because $\mathcal{S}'$ is bipartite. Hence $\mathcal{S}'$ is non-Hamiltonian.

The above result proves non-existence of Hamiltonian cycles. We now develop an algorithm which can prove non-existence of Hamiltonian cycles for a larger class of obstacles by logical reasoning. We refer to this algorithm as ESSENTIAL-CHAINS. We need to first define what we mean by a chain.

Definition 5

1. *A* **tree** *is a connected graph that does not have any cycles.*

2. *A* **chain** *is a tree, in which all vertices have degree less than or equal to* 2*. There are exactly* 2 *vertices with degree* 1 *in a chain, which we refer to as the* **terminal vertices** *of a chain.*

The ESSENTIAL-CHAINS algorithm starts with the assumption that $\mathcal{S}'$ is Hamiltonian. It then finds chains in $\mathcal{S}'$ that should form parts of any existing Hamiltonian cycle. For some obstacle scenarios, the algorithm ends up with a contradiction, hence proving $\mathcal{S}'$ is non-Hamiltonian. To develop concepts for this algorithm, we begin with the following definition:

Definition 6

1. **Chain edge***: An edge* $xy \in E_{\mathcal{S}'}$ *is a chain edge, if either* x *or* y *or both have degree* 2*. We refer to the set of all chain edges as* $E^{c}_{\mathcal{S}'}$*.*
2. **Chain vertex***: A vertex* $x \in V_{\mathcal{S}'}$ *is a chain vertex, if one of the edges incident on* x *is a chain edge. We refer to the set of all chain vertices as* $V^{c}_{\mathcal{S}'}$*.*

Consider the following simple observations:

Proposition 2

1. *If* $\mathcal{S}'$ *is Hamiltonian, then any Hamiltonian cycle of* $\mathcal{S}'$ *will contain all the edges in* $E^{c}_{\mathcal{S}'}$*.*
2. *If* $\mathcal{S}'$ *is Hamiltonian and* $|E^{c}_{\mathcal{S}'}| = |V_{\mathcal{S}'}|$*, then there exists a unique Hamiltonian cycle in* $\mathcal{S}'$
3. *All degree* 2*, degree* 1 *or disconnected vertices in* $\mathcal{S}'$ *are either periphery tiles or they share an edge with an obstacle tile.*
4. *If* $\mathcal{S}'$ *is Hamiltonian, a Hamiltonian cycle of* $\mathcal{S}'$ *must contain exactly two of the edges, incident on the every vertex in* $V_{\mathcal{S}'}$*.*

Proof

1. By definition of Hamiltonian cycle must visit all vertices in $\mathcal{S}'$. Every edge in $E^{c}_{\mathcal{S}'}$ is incident to a degree 2 vertex. Therefore, in order to visit the degree 2 vertices, the Hamiltonian cycle must traverse through all edges in $E^{c}_{\mathcal{S}'}$.
2. Any Hamiltonian cycle in $\mathcal{S}'$ has $|V_{\mathcal{S}'}|$ edges, and we already know all members of $E^{c}_{\mathcal{S}'}$ should be part of every Hamiltonian cycle that exists. Hence there is a unique choice for a Hamiltonian cycle.
3. Follows by observation.
4. Follows from the definition of Hamiltonian cycle.

If $\mathcal{S}'$ has any disconnected or degree 1 vertex then it is trivially non-Hamiltonian. From proposition 1, we know if $|\mathcal{O}|$ is odd, then $\mathcal{S}'$ is non-Hamiltonian. Thus the added utility of ESSENTIAL-CHAINS algorithm is evident when the obstacle set has an even number of tiles and $\mathcal{S}'$ is 2-connected. For the rest of the discussion on ESSENTIAL-CHAINS algorithm we will assume $|\mathcal{O}|$ is even and $\mathcal{S}'$ is 2-connected.

4.1 ESSENTIAL-CHAINS Algorithm

The ESSENTIAL-CHAINS algorithm begins by assuming, $\mathcal{S}'$ is Hamiltonian. The input to ESSENTIAL-CHAINS is the graph $\mathcal{S}'$. There are three possible outcomes of ESSENTIAL-CHAINS:

1. Either the algorithm comes up with a contradiction and exits abruptly, thus proving $\mathcal{S}'$ is non-Hamiltonian. Or,
2. it finds a graph K whose components are chains. These chains are essential components of any Hamiltonian cycle of $\mathcal{S}'$. Or,
3. it comes up with a K, such that $V_K = V_{\mathcal{S}'}$, thus finding a unique Hamiltonian cycle K for $\mathcal{S}'$.

We refer to the neighborhood set of x in $\mathcal{S}'$ as $n_{\mathcal{S}'}(x)$ and the neighborhood set in K as $n_K(x)$. Using the same notation, for an $x \in K$, $d_K(x)$ and $d_{\mathcal{S}'}(x)$ refer to the degree of the vertex x in K and $\mathcal{S}'$ respectively.

As a first step of ESSENTIAL-CHAINS, we add all chain vertices of $\mathcal{S}'$ and all chain edges incident on them, into the vertex and edge set of K respectively.

```
K = (V_K, E_K)
First-Step(S')
1   Initialize $V_K = \{\}$ ,  $E_K = \{\}$
2   for all $x$, such that $x \in V_{\mathcal{S}'}$, s.t. $d_{\mathcal{S}'}(x) = 2$
3       do Let $y, z \in n_{\mathcal{S}'}(x)$
4           if $\exists$ a path between $y$ and $z$ in $K$
5               then $\mathcal{S}'$ is non-Hamiltonian  EXIT
6               else $V_K \leftarrow V_K \cup x \cup y \cup z$
7                    $E_K \leftarrow E_K \cup xy \cup xz$
```

Note that after the execution of FIRST-STEP, $V_K = V^c_{\mathcal{S}'}$ and $E_K = E^c_{\mathcal{S}'}$. The next module is central to the ESSENTIAL-CHAINS algorithm. This module loops over all vertices which are in V_K, and are degree 3 in $\mathcal{S}'$. We update a candidate set $\mathcal{X}_c$, after execution of this module.

```
Essential-Chains()
1   Initialize $\mathcal{X}_c = \{x : x \in V_K, d_{\mathcal{S}'}(x) = 3\}$
2   while $\mathcal{X}_c \neq \{\}$
3       do
4           Pick an $x \in \mathcal{X}_c$ Let $n_{\mathcal{S}'}(x) = \{y, z, w\}$
5           switch
6               case $d_K(x) = 1$ :  Let $y \in n_K(x)$ and $z, w \notin n_K(x)$
7                   switch
8                       case $\exists$ a path in $K$ between $z$ and $w$ :
9                           switch
10                              case $d_K(z) = 2$ and $d_K(w) = 2$ :
11                                  $\mathcal{S}'$ is non-Hamiltonian  EXIT
12                              case $d_K(z) = 2$ and $d_K(w) = 1$ :
13                                  $V_K \leftarrow V_K \cup w$
```

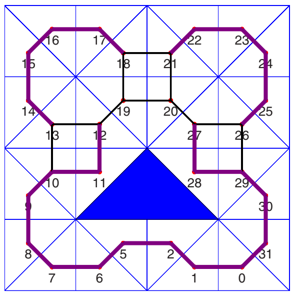
```
14                        $E_K \leftarrow E_K \cup xw$
15                        **if** $K$ has changed
16                          **then** UPDATE $\mathcal{X}_c([w, z], x)$
17            **case** $\exists$ a path in $K$ between $z$ and $x$ :
18                **if** $d_K(w) \neq 2$
19                  **then if** path-length in $K$ between $z, x < |V_{\mathcal{S}'}|$
20                        **then** $V_K \leftarrow V_K \cup w$
21                             $E_K \leftarrow E_K \cup xw$
22                             **if** $K$ has changed
23                               **then** UPDATE $\mathcal{X}_c([w, z], x)$
24                        **else** $\mathcal{S}'$ is Hamiltonian and $K$
25                              is the unique Hamiltonian
26                              cycle EXIT
27                  **else** $\mathcal{S}'$ is non-Hamiltonian EXIT
28      **case** $d_K(x) = 2$ :
29           Let $y, z \in n_K(x)$ and $w \notin n_K(x)$
30          **switch**
31            **case** $d_{\mathcal{S}'}(w) = 2$ : $\mathcal{S}'$ is non-Hamiltonian EXIT
32            **case** $d_{\mathcal{S}'}(w) = 3$ : Let $n_{\mathcal{S}'}(w) = \{a, b, x\}$
33                $V_K \leftarrow V_K \cup w \cup a \cup b$
34                $E_K \leftarrow E_K \cup aw \cup bw$
35                **if** $K$ has changed
36                  **then** UPDATE $\mathcal{X}_c([a, b, w, y, z], x)$
37      **case** $d_K(x) = 3$ :
38          $\mathcal{S}'$ is non Hamiltonian EXIT
39    $\mathcal{X}_c \leftarrow \mathcal{X}_c - x$
```

The function UPDATE $\mathcal{X}_c(Y, x)$, first selects all vertices in Y with degree 3 in $\mathcal{S}'$, lets call this set of vertices $Y_3 \subset Y$. The function deletes all previous occurrences of the elements of Y_3 from $\mathcal{X}_c$ and then adds the elements of Y_3 right after the occurrence of x. The function then deletes x from $\mathcal{X}_c$ before returning.

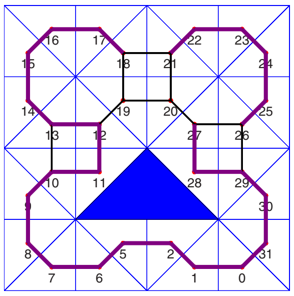
We now illustrate the execution of the ESSENTIAL-CHAINS algorithm using an example

Consider $L = 5$ and $\mathcal{O} = \{3, 4\}$, as shown in figure 5. Figure 5(a), shows K after the execution of the FIRST-STEP. The candidate set $\mathcal{X}_c$ after FIRST-STEP is initialized to $\{10, 12, 13, 18, 21, 26, 27, 29\}$ Now we present a step by step execution of the ESSENTIAL-CHAINS algorithm

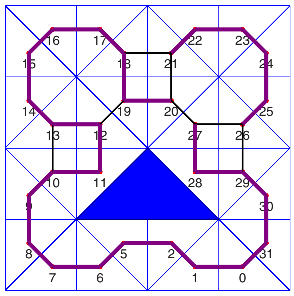
1. Pick $x = 10$ from $\mathcal{X}_c$, $d_K(x = 10) = 2, \{y, z, w\} = \{9, 11, 13\}$. The control goes to the case in line 32, finds $\{a, b\} = 14, 12$ and adds edge $12 - 13$ to K, figure 5(b). $Y = \{13, 9, 11, 12, 14\}$, $Y_3 = \{13, 12\}$ UPDATE $\mathcal{X}_c$ returns $\{12, 13, 18, 21, 26, 27, 29\}$.
2. Pick $x = 12$, $d_K(x = 12) = 2, \{y, z, w\} = \{11, 13, 19\}$. Again in line 32, the algorithm finds $\{a, b\} = 18, 20$, adds edges $18 - 19$, $19 - 20$ and vertices $\{19, 20\}$ to K, figure 5(c). $Y = \{19, 11, 13, 18, 20\}$, $Y_3 = \{19, 13, 18, 20\}$ UPDATE $\mathcal{X}_c$ returns $\{20, 18, 13, 19, 21, 26, 27, 29\}$.



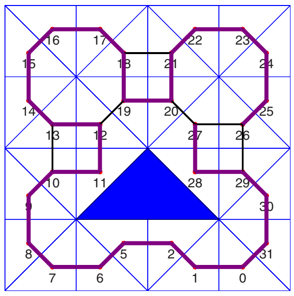
(a) K after FIRST-STEP.



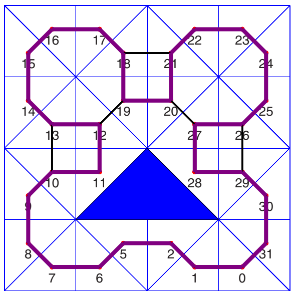
(b) K after $x = 10$ is picked.



(c) K after $x = 12$ is picked.



(d) K after $x = 20$ is picked.



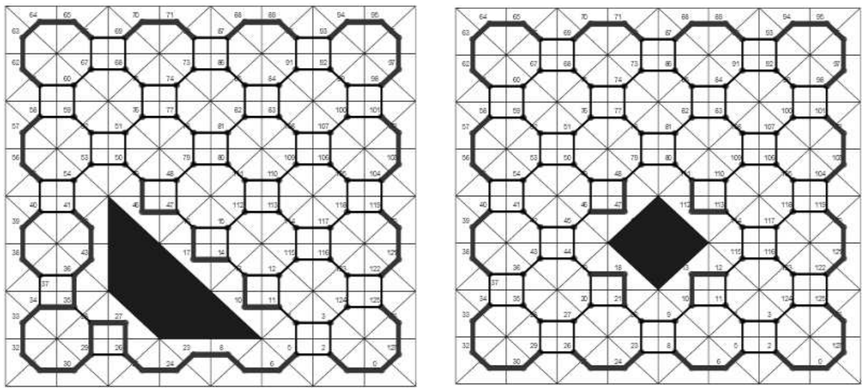
(e) K after $x = 27$ is picked.

Fig. 5. Illustration of ESSENTIAL-CHAINS algorithm. Graph K is shown using thick edges.

3. Pick $x = 20$, $d_K(x = 20) = 1, \{y, z, w\} = \{19, 27, 21\}$. There exists a path in K between $x = 20$ and $z = 27$, and its length is less than $|V_{\mathcal{S}'}|$. Therefore, control goes to the case in line 20, and adds edge $20 - 21$ to K, figure 5(d). $Y = Y_3 = \{21, 27\}$ and UPDATE $\mathcal{X}_c$ returns $\{27, 21, 18, 13, 19, 26, 29\}$.
4. Pick $x = 27$, $d_K(x = 27) = 1, \{y, z, w\} = \{28, 20, 26\}$. There exists a path in K between $z = 20$ and $w = 26$, $d_K(z = 20) = 2$ and $d_K(w = 26) = 1$. Therefore, control goes to the case in line 12, and adds edge $27 - 26$ to K, figure 5(e). $Y = Y_3 = \{20, 26\}$ and UPDATE $\mathcal{X}_c$ returns $\{26, 21, 18, 13, 19, 29\}$.
5. For the remaining elements in $\mathcal{X}_c$, K does not change. Therefore, there are no further additions to $\mathcal{X}_c$. The algorithm eventually terminates when $\mathcal{X}_c$ becomes empty. In this case, for the given obstacle configuration, the algorithm finds the unique Hamiltonian cycle.

In figure 6(a) the ESSENTIAL-CHAINS algorithm finds a contradiction. When vertex 37 is picked from the candidate set $\mathcal{X}_c$, the set $n_{\mathcal{S}'}(37) - n_K(37) = \{36, 34\}$. Both 34 and 36 have degree 2 in K, thus the algorithm concludes that $\mathcal{S}'$ is non-Hamiltonian. This can be seen in the following manner: under the assumption that a Hamiltonian cycle exists, K should contain exactly two edges incident on every node. One edge incident on 37 is already a member of K. For the other edge, one has a choice between $37 - 34$ and $37 - 36$. Now since two edges incident on each of the vertices 36 and 34 are already in K, for both choices $37 - 34$ and $37 - 36$, a degree 3 vertex will result. This is a contradiction.

In figure 6(b), ESSENTIAL-CHAINS is unable to determine whether $\mathcal{S}'$ is Hamiltonian or not. The thick edges are essential in a Hamiltonian cycle if one exists. Our main result of this paper, proves that $\mathcal{S}'$ Hamiltonian for obstacle configuration shown in figure 6(b).



(a) Non-Hamiltonian graph: There is a contradiction for vertex 37 in the lower left corner.

(b) Indeterminate result by ESSENTIAL-CHAINS.

Fig. 6. Outcomes of ESSENTIAL-CHAINS

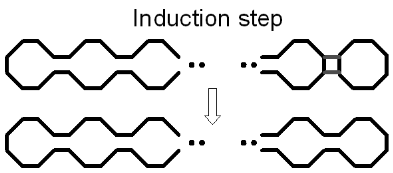
4.2 Main Result

The main result of the chapter is a positive result. But before we present our main result we need the following Lemma to prove it.

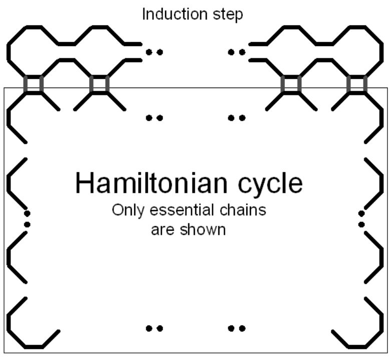
Lemma 1. *Let L be odd and $V \subset V_{\mathcal{S}}$ such that the induced subgraph $\mathcal{S}[V]$ is a rectangular lattice with octagons on the four corners. Then $\mathcal{S}[V]$ is Hamiltonian.*

Proof. We prove this lemma by mathematical induction on the dimension of the lattice. First step verification: the lattice is 1×1. In other words, if V is such that $\mathcal{S}[V]$ is an octagon, then $\mathcal{S}[V]$ is trivially Hamiltonian.

All chains of octagons with interleaving squares can be proved to be Hamiltonian using induction as shown in figure 7(a). Now we know that all chains of m octagons with interleaving squares are Hamiltonian. As our new induction hypothesis, we assume that a rectangular lattice $m \times n$ of octagons with interleaving squares is Hamiltonian. Now as shown in figure 7(b), we apply the induction step again to prove that the rectangular lattice of dimension $m + 1 \times n$ bounded by octagons on the four sides is Hamiltonian.



(a) Hamiltonian cycle in m × 1 lattice of octagons.



(b) Hamiltonian cycle in m × n lattice of octagons.

Fig. 7. Odd level dual graph for any rectangular region is Hamiltonian

Theorem 1. *If the induced graph $\mathcal{S}[\mathcal{O}]$ is a simple cycle, and ESSENTIAL-CHAINS does not come up with a contradiction, then $\mathcal{S}'$ is Hamiltonian.*

Proof. We prove this result by explicitly showing that a Hamiltonian cycle can be constructed in $\mathcal{S}'$. Here we give a sketch of the proof for an odd level of decomposition $L = 2k+1$. Since $\mathcal{S}[\mathcal{O}]$ is a simple cycle, it can only be a square or an octagon. If $\mathcal{S}[\mathcal{O}]$ is an octagon, it partitions the entire graph into rectangular regions, both when it is an interior octagon (figure 8(a)) and a periphery octagon (figure 8(b)). Now we know by Lemma 1 that the rectangular partitions of the graph are Hamiltonian. The Hamiltonian cycles of each of these rectangular partitions can be stitched together to form one Hamiltonian cycle for $\mathcal{S}'$, as shown in figure 8(e). If $\mathcal{S}[\mathcal{O}]$ is a square, then for all the squares that exist on the periphery of the graph, the ESSENTIAL-CHAINS algorithm finds a counter example, thus proving the non-existence of a Hamiltonian cycle. However, for other squares, ESSENTIAL-CHAINS does not come up with a contradiction and for all such cases, a bounding cycle can be found (figures 8(c) and 8(d)). Again, as in the case of an octagon, the rectangular partitions created are Hamiltonian, by Lemma 1. Stitching together Hamiltonian cycles as shown in figure 8(f) gives us a Hamiltonian cycle for $\mathcal{S}'$.

The above result holds true for even values of L and the proof is very similar to the one above.

Theorem 1 presents a lot of interesting and intriguing research questions. The idea of finding a bounding box and connecting Hamiltonian cycles can be used when the planar region to be explored has multiple disconnected obstacles. This idea can also be utilized for collaborative exploration by multiple vehicles, where each vehicle is assigned a rectangular partition and it executes the optimum cyclic tour computed for that particular partition. Collaborating vehicles can also exploit the multiple connecting edges their optimal cycles to those of their partners and switch tours. This may be helpful when the collaborating agents

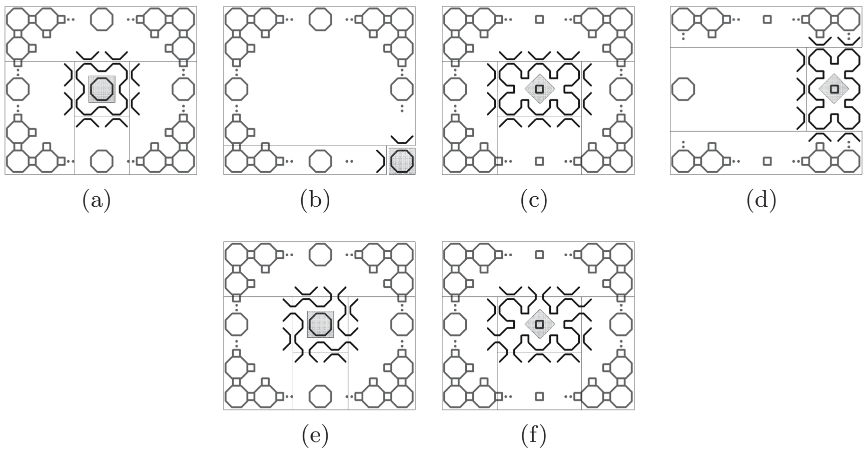


Fig. 8. Illustration of proof of the main result

are heterogenous and a vehicle with a certain kind of capability is needed at a certain location. A stronger and more useful result will be: to find the largest dimension of a bounding box such that if the obstacle configuration results in a non-Hamiltonian $\mathcal{S}'$ for this bounding box, the obstacle configuration will result in non-Hamiltonian $\mathcal{S}'$ for any bounding box, no matter how large it is. We are currently investigating the properties of the dual graph to gain further insight into the problem. The eventual aim of this research is to characterize the minimum number of repetitions of tiles for all obstacle configurations and to find provable algorithms to find the optimal cyclic tours. We now present a few heuristics based algorithms. Using results from this section we evaluate the performance of the algorithms for obstacle configurations where we know the optimum.

5 Algorithm Development and Comparison

In section 4.1, we described the ESSENTIAL-CHAINS algorithm that determines the set of chains K (as in figure 6(a)) that are essential in a Hamiltonian cycle, if one exists. As we observed in figure 6(b), the algorithm can terminate without a conclusive answer to the existence of a Hamiltonian cycle. In this section, we describe a heuristic based cycle maximization algorithm that tries to find the longest cycle in the induced dual subgraph S'. Later in this section, we show that the cycle maximization algorithm does not necessarily find the optimal solution. We also show that by including a few conditions, we can make the cycle maximization step always output a Hamiltonian cycle for the class of obstacles considered in Theorem 1.

5.1 Pre-cycle Computation

In this section, we present the first stage of the cycle maximization algorithm. We refer to this as the pre-cycle computation step. In the presence of obstacles, the Sierpinski tour gets partitioned into one or more chains as shown in figure 9(a). The idea is to use these chains to find a Hamiltonian cycle in $\mathcal{S}'$. We describe the algorithm in more detail below.

Group the obstacle tiles into k subsets $\mathcal{O}_1, \mathcal{O}_2, \cdots \mathcal{O}_k$, where each subset $\mathcal{O}_i$ is the largest set such that all its members have continuous Sierpinski ordering. For example (figure 9(a)), for the Sierpinski decomposition level $L = 7$, let $\mathcal{O}$ be $\{47, 80, 81, 82\}$. It turns out, for this example, $k = 2$, $\mathcal{O}_1 = \{47\}$ and $\mathcal{O}_2 = \{80, 81, 82\}$.

The k subsets, $\mathcal{O}_1, \mathcal{O}_2, \cdots, \mathcal{O}_k$, generate an equal number of chain partitions $\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_k$ as shown in figure 9(a). In our example, the two subsets $\mathcal{O}_1$ and $\mathcal{O}_2$ create two chain partitions $\mathcal{P}_1 = \{48, 49, \cdots 78, 79\}$ and $\mathcal{P}_2 = \{83, 84, \cdots 126, 127, 0, 1, 2, \cdots, 45, 46\}$ respectively.

After finding the chain partitions, we find the candidate bridges. A candidate bridge is essentially a pair of vertices (x, y) in the induced subgraph $\mathcal{S}'$ such that the edge $xy \in E_{\mathcal{S}'}$ (figure 9(a)). In other words, tiles that correspond to x and y in the original Peano-Cesaro triangulation are adjacent. To find the candidate bridges for a subset, say, $\mathcal{O}_i = \{w_{m+1}, w_{m+2}, \cdots, w_{m+r}\}$, we first identify the pair of vertices (u, v) such that u is the largest tile index less than $(m + 1)$ and v is the smallest tile index greater than $(m + r)$. In our example, $(u, v) = (46, 48)$ corresponding to the subset $\mathcal{O}_1$ and $(79, 83)$ corresponding to the subset $\mathcal{O}_2$.

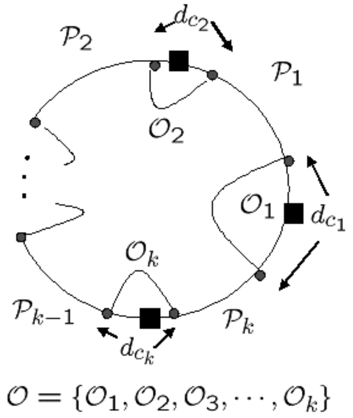
Now, for each subset $\mathcal{O}_i$, starting from the pair (u, v), we search (tile indices lesser than u for x and tile indices greater than v for y) for the first occurrence of a pair of vertices $(x, y) \in \mathcal{S}'$, where, x and y are defined in the previous paragraph. Geometrically interpreting the figure 9(a), we search (outwards and starting from the pair (u, v)) for the pair of vertices $(x, y) \in \mathcal{S}'$ that minimizes the distance d_c between them. Here, the distance d_c between two vertices u and v is expressed as:

$$d_c = min(abs(u - v) - 1, 2^L - abs(u - v) - 1) \tag{1}$$

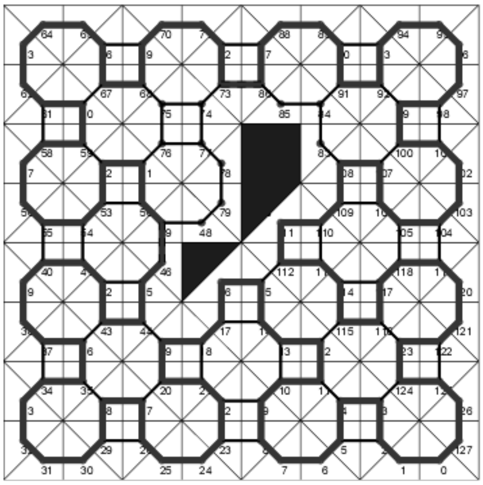
where L is the Peano-Cesaro decomposition level, $abs()$ is the absolute value function and $min()$ returns the minimum value. In the above example, the two candidate bridge pairs (x, y) determined are $(46, 49)$ and $(73, 86)$ for $\mathcal{O}_1$ and $\mathcal{O}_2$ respectively. The pre-cycle computation outputs a cycle (figure 9(b)) which is the input to the cycle maximization step discussed in the next section.

5.2 Cycle Maximization

Denote the cycle found in section 5.1 by $\mathcal{C}$ (figure 9(b)). The cycle $\mathcal{C}$ is the input to the *cycle maximization* step. Denote the set of vertices in the cycle $\mathcal{C}$ by $V_{\mathcal{C}}$. We call the graph $\mathcal{S}[V_{\mathcal{S}'} - V_{\mathcal{C}}]$ as $\mathcal{G}$. Now if $\mathcal{S}'$ is 2-connected, there exists at least 2 vertices in $\mathcal{C}$ whose neighborhood set $n_{\mathcal{S}'}()$ in $\mathcal{S}'$ has vertices in $\mathcal{G}$. Let us denote two such vertices in $\mathcal{C}$ by $x_{\mathcal{C}}$ and $y_{\mathcal{C}}$. Let their neighbors in $\mathcal{G}$ be $x_{\mathcal{G}}$

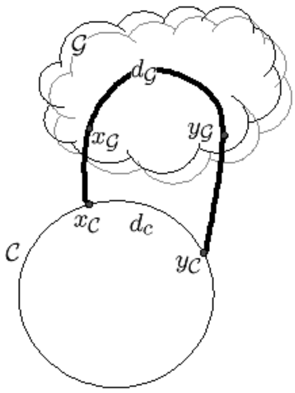


(a) Finding chains and bridges.

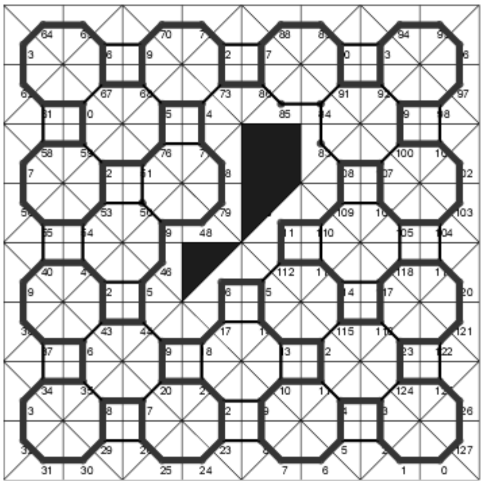


(b) Output of pre-cycle computation.

Fig. 9. Illustration for finding partition chains, finding bridges and computing pre-cycle



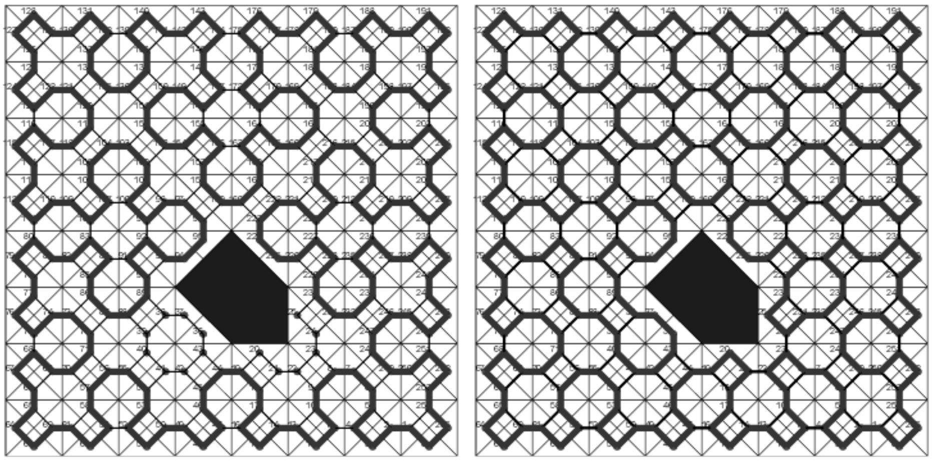
(a) Illustration of cycle maximization.



(b) Final solution cycle computed by the cycle maximization step.

Fig. 10. Cycle maximization and its output

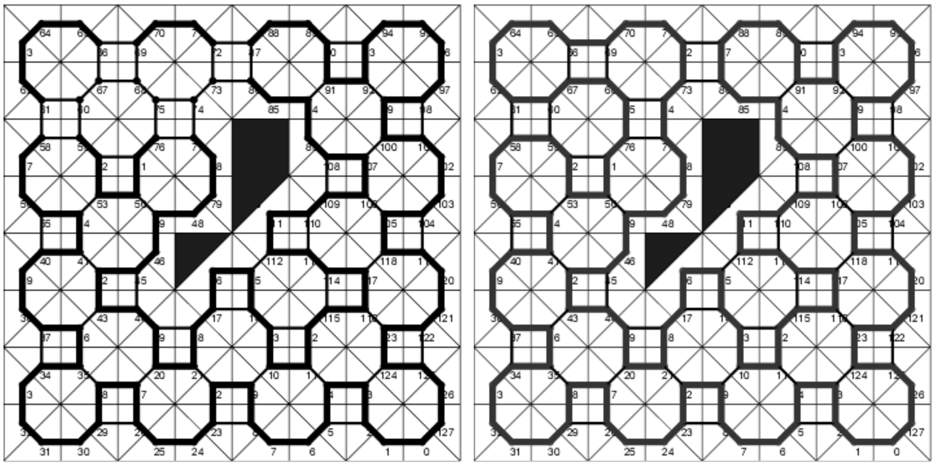
and $y_{\mathcal{G}}$ respectively (figure 10(a)). The algorithm now finds the shortest path in $\mathcal{G}$ between $x_{\mathcal{G}}$ and $y_{\mathcal{G}}$. Let us denote the distance of this shortest path as $d_{\mathcal{G}}$. Therefore the distance between $x_{\mathcal{C}}$ and $y_{\mathcal{C}}$ in $\mathcal{G}$ is $d_{\mathcal{G}} + 2$. Now if $d_{\mathcal{G}} + 2 > d_{\mathcal{C}}$, where $d_{\mathcal{C}}$ is the shortest path distance between $x_{\mathcal{C}}$ and $y_{\mathcal{C}}$ in $\mathcal{C}$, then it is possible to increase the number of nodes in the initial cycle $\mathcal{C}$. This can be done by



(a) Initial cycle $\mathcal{C}$ (*thick lines*) determined by the pre-cycle computation step.

(b) A Hamiltonian cycle *(thick lines)* determined as the solution cycle using cycle maximization step.

Fig. 11. Illustration of cycle maximization. In this example, cycle maximization comes up with a Hamiltonian cycle (an optimal solution in this case).



(a) Essential Chains generated.

(b) Hamiltonian cycle solution.

Fig. 12. Performance of ESSENTIAL-CHAINS algorithm

replacing the path in $\mathcal{C}$ between $x_{\mathcal{C}}$ and $y_{\mathcal{C}}$ by the path between them in $\mathcal{G}$. The algorithm terminates when no such increase in the cycle length is possible. This concept is illustrated in figure 10(a).

After all the possible replacements are done, the resulting cycle $\mathcal{C}$ is the final solution cycle. Figure 10(b) illustrates a solution cycle determined using the cycle maximization step. For certain cases, cycle maximization finds the optimal tour as illustrated in figures 11(a) and 11(b).

In order to assess the performance of our heuristic based cycle maximization algorithm, we need to determine the minimum number of repetitions attainable. For this, we ran our *ESSENTIAL- CHAINS* algorithm and the computed set of essential chains, K, is illustrated in figure 12(a). Now by observation, we join the computed essential chains to find a Hamiltonian cycle for this obstacle configuration as shown in figure 12(b). Thus, for this example, we observe that our heuristic based cycle maximization algorithm is four vertices away from the attainable optimal solution.

We also observe that the vertices missed $(50, 83, 84, 85)$ by the cycle maximization step (figure 10(b)) were a part of the essential chains set, K (figure 12(a)). Therefore, it may be possible that by forcing the cycle maximization not to drop vertices that are essential, the optimal solution can always be obtained. We are currently working on conditions, which, if included in the cycle maximization step, can always output the optimal tour for the class of obstacles described in Theorem 1.

6 Conclusion

In this chapter we use Peano-Cesaro tiling to divide the exploration region into triangular tiles, with some tiles marked as obstacles. The problem of finding the minimum repetition cyclic tour of the non-obstacle tiles, under mobility constraints has been posed using results from graph theory, where vertices and edges represent tiles and allowed moves respectively. The resulting graph is referred to as $\mathcal{S}'$. We have devised an algorithm that collects all the essential components of a Hamiltonian cycle (assuming one exists). The algorithm determines whether $\mathcal{S}'$ is Hamiltonian or not in some cases. In cases where the number of obstacle tiles is sufficiently low, and the obstacles are far away from the boundary of the region, the algorithm does not provide a deterministic answer. However, it does output chains that are essential components of a Hamiltonian cycle, if one exists. The main result is: *$\mathcal{S}'$ is Hamiltonian if the obstacle tiles form a simple cycle.* We also provide a heuristic algorithm and compare its performance for the obstacle scenarios where the minimum number of repetitions is known.

Acknowledgements

The authors would like to acknowledge the help provided by Dr. Junxian Wang, UtopiaCompression Corporation for the preliminary part of this work. This work was supported under NASA phase II SBIR grant number NNM06AA08C.

References

1. J. J. Bartholdi and P. Goldsman. Continuous Indexing of Hierarchical Subdivisions of the Globe. *International Journal of Geographical Information Science*, 15(6):489–522, 2001.
2. J. J. Bartholdi and P. Goldsman. The Vertex-Adjacency Dual of a Triangulated Irregular Network has a Hamiltonian Cycle. *Operations Research Letters*, 32:304–308, 2004.
3. J. J. Bartholdi and L. K. Platzman. Heuristics based on spacefilling curves for combinatorial problems in the plane. *Management Science*, 34(3):291–305, 1988.
4. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, second edition.
5. UtopiaCompression Corporation. Advanced, Disruptive Intelligent Based Image Compression Technologies. *National Institute of Standards and Technology: Advanced Technology Program (Final report)*, Nov 2006.
6. R. Diestel. *Graph Theory*. Springer, second edition.
7. S. Hutchinson G. Kantor W. Burgard L. Kavraki S. Thrun H.Choset, K. Lynch. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, 2005.
8. H. Imai. Worst-case Analysis for Planar Matching and Tour Heuristics with Bucketing Techniques and Spacefilling Curves. *Journal of the Operations Research Society of Japan*, 29(1):43–68, 1986.
9. M. Jun and R. D'Andrea. *Path Planning for Unmanned Aerial Vehicles in Uncertain and Adversarial Environments*, chapter 6, pages 95–111. Cooperative Control: Models, Applications and Algorithms. Kluwer Academic Publisher, 2002.
10. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
11. B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freedman and Co., 1983.
12. L. K. Platzman and J. J. Bartholdi. Spacefilling curves and the planar travelling salesman problem. *Journal of the Association for Computing Machinery*, 36(4):719–737, 1989.
13. H. Sagan. *Space filling curves*. Springe-Verlag, 1994.
14. K. Savla, F. Bullo, and E. Frazolli. On Traveling Salesperson Problems for Dubin's Vehicle: Stochastic and Dynamic Environments. *Proceedings of the IEEE Conference on Decision and Control*, 2005.
15. K. Savla, F. Bullo, and E. Frazolli. Traveling Salesperson Problems for the Dubin's Vehicle. *IEEE Transactions on Automatic Control*, June 2006.