Dynamic graph-based search in unknown environments

HAYNES, Paul, ALBOUL, Lyuba and PENDERS, Jacques

Published version

HAYNES, Paul, ALBOUL, Lyuba and PENDERS, Jacques (2012). Dynamic graph-based search in unknown environments. Journal of Discrete Algorithms, 12, 2-13.

Dynamic graph-based search in unknown environments

Paul S. Haynes, Lyuba Alboul, Jacques Penders

Centre for Automation and Robotics Research, Sheffield Hallam University, City Campus, Howard Street, S1 1WB

Abstract

A novel graph-based approach to search in unknown environments is presented. A virtual geometric structure is imposed on the environment represented in computer memory by a graph. Algorithms use this representation to coordinate a team of robots (or entities). Local discovery of environmental features cause dynamic expansion of the graph resulting in global exploration of the unknown environment. The algorithm is shown to have $O(k \cdot n_H)$ time complexity, where n_H is the number of vertices of the discovered environment and $1 \leq k \leq n_H$. A maximum bound on the length of the resulting walk Ω is given.

Keywords: Team Robotics, Dynamic Search, Graph Theory, Multi-robot Localization

1. Introduction

The method presented in this paper stems from the research in multi-robot systems within the remits of the recently completed GUARDIANS project[1]. Autonomous mobile robotics, in particular collective and cooperative robotics, has gained a lot of attention recently.

Multi-robot systems pose new challenging problems such as cooperative perception and localization, cooperative task planning and execution, team navigation behaviors, robot interactions among themselves and with humans, cooperative learning, and communication.

There have been some significant advances in tackling the aforementioned problems, often based, however, on empirical approaches. They are either

[1] GUARDIANS, Group of Unmanned Assistant Robots Deployed in Aggregative Navigation supported by Scent Detection, EU FP6 ICT 045269

driven by informal expert knowledge, or by resource-intensive trial-and-error processes [6].

There is a demanding need for formalization of methodologies and theoretical frameworks capable of providing solutions to general classes of problems specific to multi-robot systems.

In this paper such a framework is proposed for the problem of global self-localization of multi-robot teams, without *a priori* information about the environment.

The problem of self-localization is central in robotics, and is particularly difficult in unknown indoor environments where such tools as GPS are unavailable.

It is directly related to the famous SLAM problem of a robot simultaneously localizing and building a map of the environment. This problem has been studied extensively in the robotics literature, focusing mostly on a single robot. Conceptually, the SLAM problem for a single robot in 2D is considered to be solved, but in practice it may still encounter difficulties, even outdoors, in urban areas or forests. SLAM approaches are mainly probabilistic in their nature due to the uncertainty of acquired information. Data association methods used in SLAM require significant computation in real-life implementations, and contribute to increased complexity [7].

The problem of multi-robot localization and encountered difficulties has not yet been fully researched [8]. A multi-robot team, by definition, represents a sensor network. An important aspect of a multiple robotic system, as opposed to a single robot, is the richness of available information. In a cooperative multi-robot team, robots obtain information from their own sensors as well as other robots. This information can be of various types: perceptual (data from lasers, various distributed cameras) as well as non-perceptual (symbolic information, directions, and commands, obtained from other robots or a database). Therefore, such plethora of information should be taken into account.

In the last decade, several works appeared that tackle the problem of cooperative multi-robot localization. Whereas some approaches still consider this problem within the SLAM framework, by treating the problem of multi-robot localization as a Multi-SLAM problem [9], others, while still using probabilistic methods, attempt to take into consideration robots as landmarks themselves [10]. Another trend is based on robot distribution on site, which can work well if the group of robots is large and communication between them is robust [11].

A promising mathematical tool to characterize a multi-robot system is a graph. Indeed, the problem of coordination in multi-robot systems can be characterized naturally by a finite representation of the configuration space using Graph Theory. Vertices represent robots with resources limited by sensors, control design, and computational power. Edges are virtual entities describing local interactions and can support information flow between vertices/robots. If other sensor devices are present in the environment they can be added to the sensor robot networks. Graph theory facilitates analysis of the interplay between the communications network and robot dynamics, and to choose strategies for information exchange which mitigate these effects.

Graph-theoretical approaches have been increasingly used for building and analyzing communication and sensor networks [12].

In this paper we describe a graph-theoretical framework for cooperative multi-robot localization. The (unknown) site is initially covered by an infinite virtual triangular grid (triangular tiling) T^{∞}, depicted in Fig. 1.

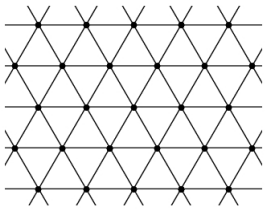

Figure 1: Virtual triangular grid.

The grid spans all directions, and as robots explore the site local parts of the grid become actualized. The environment, therefore, represents a subgraph L of T^{∞}. Each robot is equipped with a Laser Range Finder (LRF) which is used as the main sensor for position detection with radio signal as a backup.

The length of the edges is limited by the range of the LRF, or can be smaller depending on the initial position of the robots. Our robot team consists minimally of three robots, and robots act as dynamic and static graph vertices; they switch between these two modes in a prescribed manner. Coordination of robots whilst correcting for odometry errors then becomes more manageable and from this framework we develop a cooperative exploration algorithm.

The choice of three robots is due to several reasons. One is that this allows accurate calculation of robot positions and poses without assuming

that robots are equipped with a proprioceptive motion detector as suggested in [10], as two robots act as static beacons whilst the third robot is moving. It also allows to develop a robust movement strategy that minimizes the number of robot steps. Indeed, our goal is not only to achieve robust self-localization of robots, but also explore the unknown environment in the most optimal manner, reducing the number of visits to previously visited vertices in L.

From a theoretical point of view, our method, to a certain extent, represents a fusion and further development of strategies proposed in [14] and [13]. One crucial difference is that movements of the robots in our approach are not random, but are determined in a structured and adaptive manner. The robots build the representation of the environment simultaneously whilst moving. For this reason, we consider the dual graph H^∞ to T^∞; the vertices of this dual graph are possible positions of the 3-robot team considered as a whole.

Surprisingly, our result bears some similarity to that of [15] in which a Kohonen Self-Organizing Network (SOM) is used to obtain a topological graph representation of the environment. The SOM vertex positions change during network convergence, but the graph itself does not, i.e edges are not deleted. Our approach represents the environment better in the sense that unnecessary edges and vertices are removed and obstacles are represented as cycles in the graph. Whilst the SOM approach can build a map of the environment, there is no planning capability and the *single* robot is not guaranteed to cover all of the unknown environment. A further advantage is a lower computational cost; neural network approaches can take a long time to converge.

In the next section our approach is described in detail.

2. Framework

In this section we provide a discrete mathematical framework in which to achieve the following goals.

- Enable a team of 3 robots to autonomously explore an unknown environment.
- To make no assumptions about the environment beyond the graph embedding.

- To cover the whole of the accessible environment (*Completeness*)
- To intelligently recognize and avert the visiting of "redundant" regions (via *Intelligent rules*)
- To make deductions concerning the final walk length.

2.1. Localization and Movement Graphs

Our approach fixes a virtual geometric structure on the unknown environment, thus providing a coordinate system in which to position robots and develop algorithms by means of graph theory. The structure is the infinite triangular grid graph T^∞, chosen for reasons discussed previously. The infinite hexagonal grid graph H^∞ dual to T^∞ is also necessary.

A *localization graph* is an induced subgraph $L \subset T^\infty$ used to represent possible robot locations. The unknown localization graph to be discovered is denoted $\mathcal{L} \subset T^\infty$, with the known graph denoted $L \subset \mathcal{L}$.

The 3-clique of robots progressively learn the unknown localization graph $\mathcal{L}$ as exploration proceeds until $L = \mathcal{L} - \mathcal{L}'$, where $\mathcal{L}'$ is the undiscoverable graph, at which point the algorithm terminates. At any one time L is the learned localization graph. The undiscoverable $\mathcal{L}'$ pertains to enclosed inaccessible regions of the environment.

Likewise, an hexagonal *movement graph* is an induced subgraph of H^∞, with the unknown (at any one time) movement graph denoted $\mathcal{M} \subset H^\infty$, and the known movement graph denoted $M \subset \mathcal{M}$. The movement graph M is dual to L, and represents possible 3-clique movements governed by Rule (1) below.

Rule 1. *Let $C = \{R_i\} \in L$ be a 3-clique of vertices as in Figure (2), with corresponding dual movement graph vertex $m \in M$. A single robot is permitted to move between two stationary robots. This move corresponds to an edge connecting m to some other vertex $m' \in M$ (cf. Figures 2 and 3)*

Movement rule 1 is demonstrated in Figures 2 and 3. In figure 2 the 3-clique of robots, denoted by the square icons with black centres on the localisation vertices, obey rule 1 by moving between two stationary robots, arriving at their new configuration in Figure 2.

The justification of Rule (1) stems from the problem of odometry error correction in real robots described earlier. This well known problem demands careful consideration of the approach to robot movement to minimize the

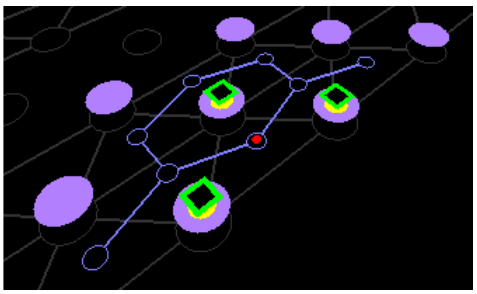

Figure 2: Robots ready for search. Neighbouring unvisited localization vertices are identified (large vertices and edges). The dual movement graph is constructed accordingly (small vertices and edges)

accumulation of odometry error. Small errors in odometry result in large errors over long distances.

2.2. Movement

Vertices of the current localization graph L represent robots (here on referred to as entities) within the environment. However, it is the movement graph M, dual to L, which facilitates actual movement.

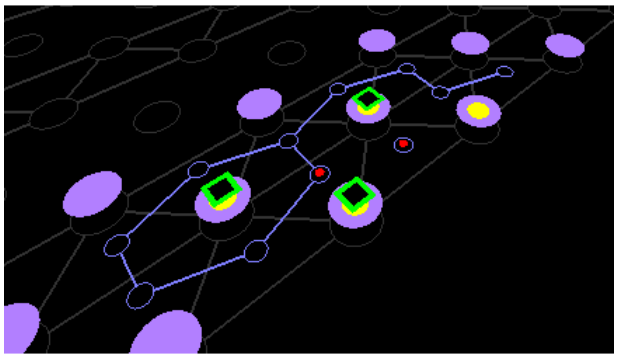

Figure 3: A single time step demonstrating robot movement and dynamic extension.

Our approach uses the principle of dynamic exploration (search) within M by moving from the current vertex to the next vertex on the outer face (also called a level-1 face [3]) of M. On moving to a new location, L and M are updated and the process repeats.

Figure 3 demonstrates updating after a move has occurred. The 3-clique of entities (square icons with black centres), are situated within the known localization graph L (denoted by large solid circles). Localization vertices with highlighted centres are visited, whilst those without, represent known (sensed) vertices. The known (hexagonal) movement graph M shows the moves available to the 3-clique (not necessarily from its current location). Highlighted hexagonal vertices indicate those vertices of M that are visited. The unknown localization graph $\mathcal{L}$ can be seen in the periphery.

As the 3-clique of entities move from vertex m to vertex m' of the movement graph the source vertex m is removed from the graph if removal does not disconnect the graph, i.e. removal is permitted if and only if $\omega(G \backslash m) = \omega(G)$, where $\omega(X)$ is the number of connected components of graph X. This simple principle of

- traversing the current outer face of M;
- dynamically extending L (and subsequently the dual graph M); and
- removing the source vertices where possible,

is a mechanism for automating the search of an unknown environment in an ordered manner. On its own, however, the geometric embeddings imposed on L and M coupled with this simple principle of search means the path taken may not be optimal. Optimization requires the addition of intelligent rules, discussed next.

2.3. Intelligent Rules

In addition to the constraints imposed by the unknown environment (such as forcing the the movement graph to be 1-connected, for example), there are other factors in which the discussed simple principle of search may be non-optimal.

There may emerge, for example, a simple path of a level-1 face whose vertices are enclosed entirely by visited vertices. Clearly it would be inefficient for the entities to revisit such vertices since we may infer from previous investigation that they are empty regions of no interest. Indeed, since sensed vertices were actualized (i.e. there were no obstacles found), and they are surrounded by solely visited vertices, then they may be inferred to be visited as well (since they are empty and of no interest). A depth first search

can quickly identify such regions and disconnect the located (possibly biconnected) region on back-tracking.

This is the purpose of the ValidatePath($\cdot$) function. Following computation of the level-1 face (which is unique at any one time), each vertex of the path proceeding from the current vertex is checked to see if it is enclosed by solely visited vertices. If it is then the graph is disconnected at this vertex since traversing the path is unnecessary and would be inefficient. Otherwise, validation is complete and the entities must be allowed to traverse the path in order to visit the unexplored region.

3. Algorithms and Complexity

3.1. Nomenclature

The logical denotations True ($\top$), False ($\bot$), and the logical AND operation over a set of discrete values ($\bigwedge$) are used. The algorithms are presented from an object oriented perspective, thus $a \rightarrow \mathrm{F}()$ denotes that $\mathrm{F}()$ is a member function of object (vertex) a to be called, for example. This should not be confused with the long arrow notation $u \longrightarrow v$, denoting vertices u and v of a graph to be connected by an edge.

The ComputeOuterFace($\cdot$) function computes the level-1 face (outer face) walk of M [3], details of which are given in the next section. The resulting outer face walk is denoted Ω, with the current member denoted $\omega \in \Omega$. The next element of the walk is denoted $\omega' = \omega + 1$. The list Ω is understood to be cyclic in that $\omega_n + 1 = \omega_1$ and $\omega_1 - 1 = \omega_n$, where ω_1 and ω_n are the first and last elements of Ω respectively, and is implemented in C++ using the list container.

The current 3-clique of entities in the localization graph L are denoted R_i, where $i = 1, 2, 3$. Position vectors associated with a vertex are denoted $a \rightarrow \mathbf{c}$, where a is a given vertex.

All pseudo code is written for the readers convenience, and more efficient logic is possible.

*3.2. The level-*1 *face*

Although simple, the level-1 face algorithm (see Algorithm 1) is given here for completeness. A vertex v is a level-k vertex if it is on the k^{th} nested face, e.g. a level-1 vertex sits on the outer face. We call a cycle of level-k vertices a level-k face [3].

Algorithm 1 ComputeOuterFace(G)
Computes the level-1 face of graph G.

1: Find left most vertex $v \in G$.
2: Let $\mathbf{u} = (0, 1)$
3: Find $\underset{w}{\text{argmin}}\{\angle(\mathbf{u}, \overrightarrow{vw}) | v \longrightarrow w\}$
4: Let $\mathbf{s} = \overrightarrow{vw}$
5: $f = v$
6: **while** $\mathbf{s} \neq \mathbf{u}$ **do**
7: $\quad f + w$
8: $\quad$ Let $\mathbf{u} = \overrightarrow{wv}, v = w$
9: $\quad$ Find $\underset{w}{\text{argmin}}\{\angle(\mathbf{u}, \overrightarrow{vw}) | v \longrightarrow w\}$
10: **end while**
11: **return** f

Computing the level-1 face is equivalent to determining the outer face, for which there is a linear time algorithm.

Figure 4 shows a connected triangular grid graph $G \subset T^{\infty}$. Finding the level-1 face begins by determining the left most vertex $v \in G$, vertex d in this case (if multiple vertices share this position then the last found is chosen by definition of the algorithm)

Now consider a direction vector $\mathbf{u}$ parallel to the vertical axis. Vertex v is called the pivot and is the first vertex of the face. Determining the next vertex requires finding a vertex $w \longrightarrow v$ such that the anti-clockwise angle from $\mathbf{u}$ to $\overrightarrow{vw}$ is minimal, (f in this case).

Direction vector $\mathbf{u}$ is then replaced by $\mathbf{u} = \overrightarrow{wv}$, and the pivot by w. Repeating the process sweeps out the face from vertex to vertex as shown until $\mathbf{u}$ is equal to the initial edge.

The notation $\angle(\mathbf{u}, \mathbf{v})$ in Algorithm 1 denotes the anti-clockwise angle from vector $\mathbf{u}$ to $\mathbf{v}$. The resulting level-1 face is an anti-clockwise cycle of level-1 vertices. This process may be considered the discrete analogue of the continuous curve fitting problem of an arbitrary set of points described in [2], but applied to embedded graphs in the plane.

3.3. Main Algorithms

The main search algorithm is presented in listing Algorithm 2. The approach is partially inspired by the algorithms for Hamiltonian walks in known

Algorithm 2 DynamicSearch()

Search unknown environment

if *graph_altered* **then** {If true, compute new outer face walk}
  $\omega' \leftarrow \emptyset$
  if $\Omega \neq \emptyset$ **then** {If a previous walk exists}
    $\omega' \leftarrow \omega + 1$ {ω points to the next element in the walk}
  end if
  $\Omega \leftarrow (\omega \rightarrow \text{ComputeOuterFace}(\cdot))$ {Compute new walk}
  if $\omega' \neq \emptyset$ **then**
    if there exists $v \in \Omega$ such that $(v = \omega) \wedge ((v + 1) = \omega')$ **then** {Find exact position in Ω if possible (should the local walk remain unchanged)}
      $\omega \leftarrow v$ {Set current position}
      Exit at step 15
    end if
  end if
  Find $\omega' \in \Omega$ such that $\omega' = \omega$ {Since the local walk has changed, find any matching vertex}
  $\omega \leftarrow \omega'$
end if
if $\omega \rightarrow \text{ValidatePath}(\omega + 1)$ **then** {Check necessity of path}
  graph_altered $\leftarrow \top$ {Redundant paths have been removed}
  Restart from step 1
end if
$\omega \leftarrow \omega + 1$ {Move to next vertex in walk}
Find $i \in \{1, 2, 3\}$ such that $R_i \notin (\omega \rightarrow S)$ {Determine entity to move}
$R_i \leftarrow (\omega \rightarrow S) \backslash ((\omega - 1) \rightarrow S)$ {Move the entity}
$r \leftarrow R_i$ {Remember which entity moved}
$r \rightarrow visited \leftarrow \top$ {Set it as visited}
if h is not a cut-vertex **then** {Remove previously visited vertex?}
  Disconnect h from all neighbors.
end if
$(\omega \rightarrow visited) \leftarrow \bigwedge_{i=1}^{3} (R_i \rightarrow visited)$
graph_altered $\leftarrow (\text{RealiseSurroundingArea}(r) > 0)$ {Update L and M}
for all 3-cliques $C_i \in L$ such that $r \in C_i$ and $\bigwedge_{c \in C_i} (c \rightarrow visited)$ **do** {Remove visited movement graph vertices v dual to C_i}
  Let $v \in M$ be the hexagonal vertex dual to C_i.
  if $v \neq \omega$ **then** {Do not consider current clique}
    if v connects to any other vertices **then**
      Disconnect those vertices connecting to v which are not cut-vertices.
      graph_altered $= \top$
    end if
  end if
end for
for all connected neighbors $s \in N(r)$ such that $\neg(s \rightarrow visited)$ **do**
  $s \rightarrow visited \leftarrow \bigwedge_{s' \in N(s)} (s' \rightarrow visited)$ {s becomes visited if its surrounding vertices are visited}
end for
return

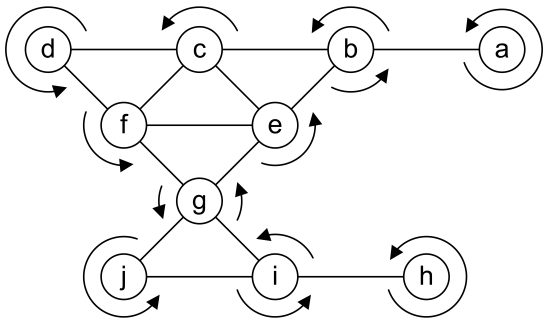

Figure 4: Simplified example of computing a level-1 (outer) face $f_1 = abcdfgjihigeb$.

environments, but adapted to unknown environments (see Takamizawa *et al* [16], for example). Optimal Hamiltonian walks for known graphs that are at least 4-connected are well established (see Tutte [4, 5], for example). The presented work provides a solution where no assumption as to k-connectedness is made.

Algorithm 2 has the following mechanisms:

(i) Computation of ComputeOuterFace$(\cdot)$ and the identification and taking of the next move in the walk, *or*, if the graph local to the 3-clique remains unchanged, taking the next move in the current walk.

(ii) Checking whether the next move is actually necessary and removing (deleting) unnecessary simple paths via ValidatePath$(\cdot)$.

(iii) Disconnecting the previous vertex $\omega - 1$ following a move to ω if $\omega - 1$ is *not* a cut-vertex.

(iv) Dynamic expansion of the 3-clique frontier via RealiseSurroundingArea$(\cdot)$, or similar.

(v) Maintaining the flagging of graph vertices as visited, either explicitly or implicitly.

For this last mechanism, note that explicit flagging occurs when a movement graph vertex is physically surrounded by the 3-clique, whereas implicit flagging occurs, for example, when a recently visited movement vertex has neighbors that are themselves surrounded by entirely visited vertices.

The complexity of algorithm 2 is given by the following proposition.

Proposition 1. *Algorithm 2 has complexity* $O(n_H)$*, where* n_H *is the number of vertices in the final movement graph* $M^* \subset \mathcal{M}$*.*

Proof. The first subroutine of algorithm 2 is ComputeOuterFace($\cdot$) which computes the level-1 face of the current movement graph M. This is a simple $O(n)$ time algorithm as discussed in section 3.2.

Following computation of the level-1 face requires locating where in the new level face corresponds to the previous location in the previous level face so that we can take the next move. This takes $O(|\Omega|)$, where $n_H \leq |\Omega| \leq 2n_H$.

Path validation and removing of unnecessary paths via ValidatePath($\cdot$) takes $O(n_H)$ time (see proposition 2).

The remaining subroutines remove remaining implicitly visited regions local to the 3-clique. Finally, by Proposition 3 (see below), the RealiseSurroundingArea() subroutine has complexity $O(1)$. Summing gives an overall complexity of $O(n_H)$. ■

Algorithm 2 makes use of the ValidatePath($\cdot$) function (see algorithm 3 and Recur(), its helper function), introduced in the previous section, which has complexity given by the following proposition.

Proposition 2. *Algorithm 3 has an upper bound complexity of $O(n_H)$.*

Proof. A level-1 face $P \subset M$ has a maximum of $n_P < n_H$ vertices. Since algorithm 2 is effectively a depth first search of P, its complexity is $O(n_P)$, or a weaker condition states that for any path P algorithm 3 has complexity $O(n_H)$. ■

Algorithm 3 ValidatePath(p)
Searches for and removes unnecessary paths

```
avoid ← this
if p → Recur() then
  Disconnect p from avoid.
  return ⊤
end if
return ⊥
```

The RealiseSurrouondingArea() function (see algorithm 4), used by algorithm 2, depends on the application at hand. A robotics setting would require this function to physically scan the surrounding area to determine which vertices to add to the localization graph L, and to connect vertices appropriately.

However, for simulation purposes an algorithm based on a known connected graph $\mathcal{L}$ is presented. Only those vertices on the periphery of the

ValidatePath: Recur()

1: $rtn \leftarrow \top$
2: $visited \leftarrow \bigwedge_{s' \in S} (s' \rightarrow visited)$
3: $this \rightarrow visited \leftarrow \top$
4: **if** $\neg$ visited **then**
5: **return** $\bot$
6: **end if**
7: **for all** $p \in N(this), p \in \Omega$ such that $p \neq avoid$ of *this* vertex **do**
8: **if** p has not yet been traversed by DFS **then**
9: **if** $(p \rightarrow$ Recur$())$ **then**
10: Disconnect p from all its neighbors.
11: **else**
12: $rtn \leftarrow \bot$
13: **end if**
14: **end if**
15: **end for**
16: **return** rtn

3-clique within $\mathcal{L}$ are made available to the algorithms. Thus, RealiseSurroundingArea() examines the unknown localization graph $\mathcal{L}$, with the entities only being aware of the vertices of the induced subgraph $L \in \mathcal{L}$ which they have previously visited, and the traversal boundary (i.e. unvisited yet sensed, or "known", vertices). A real implementation with robots would see the entities (robots) making use of a sensory device (such as a laser) to realize the surrounding area in real-time.

The complexity of RealiseSurroundingArea() is given by the following proposition, which, given the fixed graph embedding, ought to be the case in practically all applications.

Proposition 3. *Algorithm 4 has complexity* $O(1)$.

Proof. Algorithm 4 operates on induced subgraphs of the infinite triangular grid graph T^{∞}, and the number of 3-cliques about vertex r is constant (cf. Figure 5). Thus, there are a maximum of five such 3-cliques since there are six 3-cliques containing a single given vertex of the induced sub-graph and we disregard the current 3-clique since it is occupied. The set P then has a maximum of 5 elements.

Algorithm 4 RealiseSurroundingArea

Dynamically extend the graph

$P \leftarrow \emptyset + \{(\omega \rightarrow \mathbf{c}, \omega)\}$
$r \rightarrow known \leftarrow \top$
for all 3-cliques $C_i = \{r, a, b\} \in \mathcal{L}$ where $(\neg(a \rightarrow known)) \wedge (b \rightarrow known)$ **do**
    $v \rightarrow \mathbf{c} \leftarrow \frac{1}{3} \sum_{c \in C_i} c \rightarrow \mathbf{c}$ {Make $v \in M$ the dual vertex to $C_i \in L$}
    $v \rightarrow visited \leftarrow \bigwedge_{c \in C_i} (c \rightarrow visited)$
    $v \rightarrow S \leftarrow C_i$
    $P \leftarrow P + \{(v \rightarrow \mathbf{c}, v)\}$
end for
$counter \leftarrow 0$
for all elements $s \in P$ **do**
    for all elements $t \in P$ such that all t proceed s **do**
        if $\|(s \rightarrow \mathbf{c}) - (t \rightarrow \mathbf{c})\|^2 < 3/2$ **then** {Is this a neighboring hexagonal vertex}
            if $s \not\longrightarrow t$ and s has not been previously disconnected from t **then**
                Connect s to t. {Establish new connections (edges)}
                $counter \leftarrow counter + 1$
            end if
        end if
    end for
end for
return *counter*

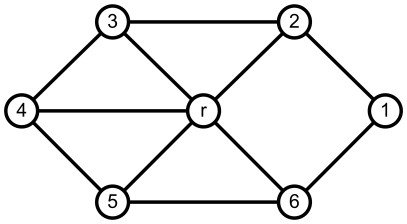

Figure 5: Example of 3-clique formation centered on r. $C = \{\{r23\}, \{r34\}, \{r45\}, \{r56\}\}$. Here two possible cliques are missing.

Finally, each element s of P considers all elements $t \in P$ proceeding s. Since there are a maximum of 5 elements in P this requires a maximum and constant number of $4 + 3 + 2 = 4(4 + 1)/2 - 1 = 10$ operations. Therefore, the total complexity is $O(1)$. ■

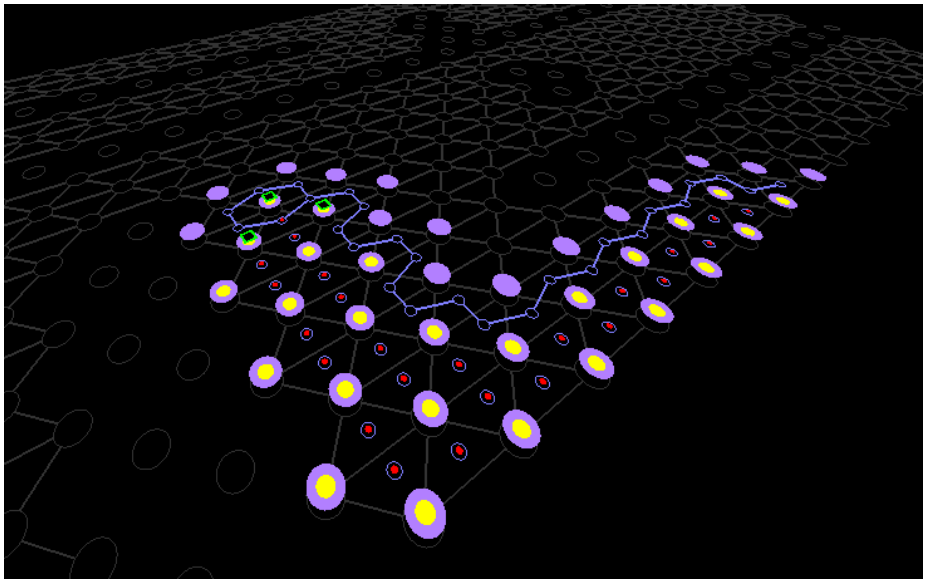

Figure 6: Dynamic graph construction.

4. Analysis and Discussion

Figures 6 and 7 show example outputs of the system (algorithm 2) given different unknown environment graphs $\mathcal{L}$. The system achieves the goals

set out at the beginning of this section, taking into account the restrictions imposed by the unknown environment (such as a lack of information as to the k-connectedness of the representative movement graph).

Empirical results aside, a number of theorems concerning completeness and walk length may be proved.

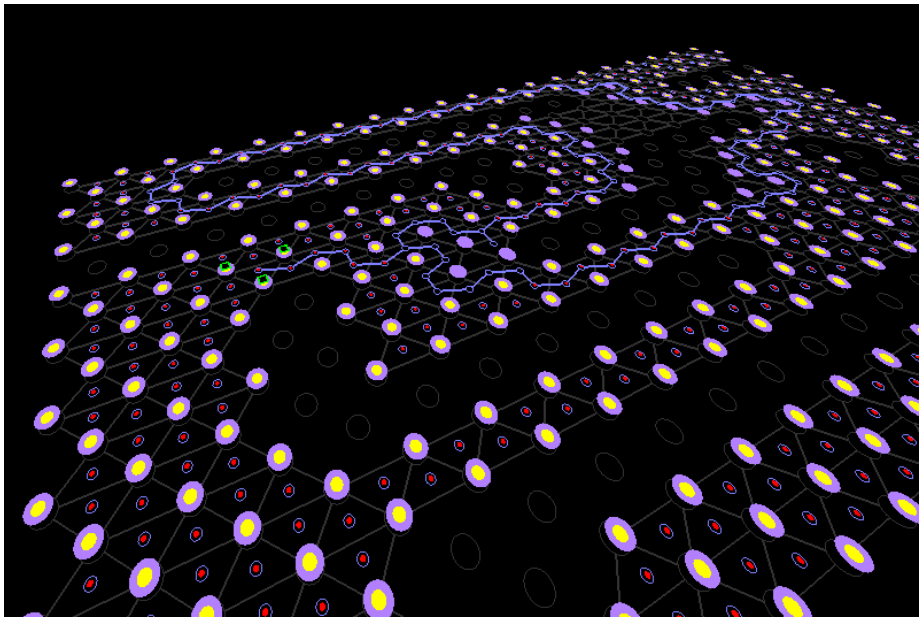

Figure 7: Dynamic graph construction.

4.1. Completeness

Completeness (briefly mentioned in section 2) ensures the algorithm completely covers the accessible induced subgraph of an environment graph $\mathcal{L}$.

Theorem 1 (Completeness). *Let $\mathcal{L}$ be the localization graph of the environment, initially unknown to the 3-clique of entities $C = \{R_i\}$ whose dual vertex is $m \in M$. Then the final walk $\Omega^* \in \mathcal{M}$ produced by Algorithm 2 spans the entire graph $\mathcal{L} - \mathcal{L}'$, where $\mathcal{L}'$ is the graph of undiscoverable vertices of the environment.*

Proof. Consider the unknown localisation graph $\mathcal{L}$ and initial movement graph M (cf. Figures 2 and 3, for example). Wherever $\mathcal{L}$ permits, each

3-clique of L instantiates a connected vertex $m' \in M$ of the movement graph. Moreover, on moving to a new movement node, m' say, where $m \longrightarrow m'$, L is updated according to $\mathcal{L}$. Moving from m to m' will disconnect the two nodes if $\omega(M \setminus m) = \omega(M)$, i.e. m is not a cut vertex. Thus, the mechanism of extension exists to instantiate and connect those vertices having potential to exist, but which have not previously been disconnected. The proof is completed by induction.

By this mechanism of extension, there always exists a simple path $P \in M$ of length $l + 1$, where $P = mp_1p_2 \cdots p_l$, such that there exists $q \in N(p_l)$ unvisited, where $N(p_l)$ is the set of neighboring vertices of p_l. The case for which $l = 0$ is simply the case for which one or more neighbors m' of m are unvisited. If no such simple path exists then the algorithm is complete since, by definition, a path is only ever disconnected when m is a cut vertex rooting one or more biconnected components which are wholly visited or enclosed by wholly visited vertices. Thus, a simple path connecting to an unvisited biconnected component of the graph is never disconnected.

In the case where the next move of the movement graph M relative to the 3-clique is unaltered from the previous level-1 face walk, then the next vertex within the previously calculated level-1 face ($\omega' = \omega + 1$) of $\Omega \in M$ is traversed. Traversal continues until an unvisited vertex is reached, in which case the graph is dynamically extended, and the outer face walk is recalculated, thus completing the induction. ■

4.2. Walk Length

The system deals with unknown environment exploration with no *a priori* knowledge of the search domain. Thus, determining an exact upper bound length for the final walk Ω^* is difficult since clearly this depends on the unknown.

However, in this section we present a logical argument which makes headway in understanding the walk length resulting from algorithm 2. An upper bound is given on the length of the final walk Ω^*.

To do this consideration of the key subroutines (mechanisms (i)-(v) listed in section 3.3) of the algorithm is required.

Let L^* be the final localization graph discovered by algorithm 2, where $L^* = \mathcal{L} - \mathcal{L}'$ and $\mathcal{L}'$ is the graph of undiscoverable vertices due to enclosed inaccessible regions of the environment. Then the final walk length, $h(\Omega^*)$, depends on the features contained within L^* which, of course, directly effects the final movement graph M^*.

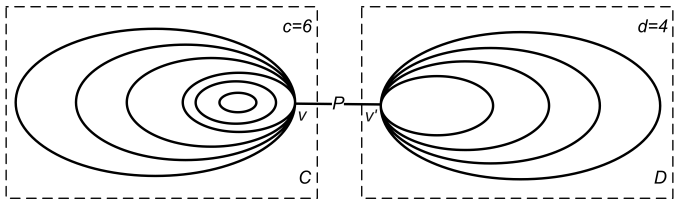

Figure 8: Concentric level-k faces of two regions C and D of graph G connected by a simple path $P = vw_1w_2 \cdots w_mv'$.

Assuming we work with L^* and M^* for the moment, then by mechanisms (i), (iv), and (v) the algorithm, by definition of the level-1 face algorithm, follows the boundary vertices of L^*. In addition mechanism (iii) deletes the graph vertex of all previous moves $\omega - 1$ where possible, thus reducing (before dynamic expansion) the graph of available future moves.

This mechanism causes previously visited vertices to act as "walls" of the environment, thus the algorithm will not tread these vertices on its next return unless doing so would allow access to one or more unvisited regions (such as biconnected components)

We can deduce that this leads to a "spiders-web", or spiraling, approach to graph discovery until all available vertices become visited.

Additionally, however, the remaining mechanism (ii) implements an element of intelligence which makes spiraling more efficient. During the course of the algorithm it may emerge that certain simple paths of the graph are surrounded entirely by visited vertices. Clearly it would be inefficient to traverse such simple paths, and the mechanism identifies and removes them using depth first search.

An inefficient property of the mechanisms presented so far concerns the existence of biconnected components connected by a path, however short, one or more of which may contain a *number* of concentric level-k faces (see Figure 8). This inefficiency is highlighted by the following lemma.

Lemma 1. *Let biconnected components C and D be two regions of M^*, connected by a simple path $P = vw_1w_2 \cdots w_mv'$, containing quantities c and d of level-k faces respectively such that $c \geq d$. Then P must be traversed $2d$ times to discover D fully.*

Proof. The previous discussion demonstrated that cut-vertices are not deleted (by mechanism (iii)) if returning to them would allow access to *one*

or more unvisited regions. This is demonstrated in Figure 8. Traversing the outer boundary (level-1 face) of region C to the indicated cut-vertex v, the level-1 face, by definition, would then traverse path P to join cut-vertex v' in region D before traversing its level-1 face. Traversal would proceed until v' is rejoined and P is traversed in the reverse direction to join v. Any remainder of the level-1 face in C would be traversed until a join side-stepped the outer face walk into the level-2 face. Note that by mechanism (iii) the level-1 face in region D would be fully deleted (assuming no further biconnected components are connected to the level-1 face of region D), as would that of region C. Thus, the simple path P is traversed exactly 2 times, with a remaining $d-1$ outer boundaries in region D.

Clearly, repeating this procedure results in a total traversal of $2d$ traversals of the simple path P to fully discover region D. ■

Now suppose mechanism (ii) is omitted from algorithm 2 for the moment. Then by the previous discussion a spiraling approach to discovery occurs, with recourse to the *outermost* cut-vertices of the boundary of the movement graph as the boundary is traversed (bearing in mind the boundary is continuously reduced where possible by mechanism (iii)).

Therefore, $h(\Omega^*)$ depends on the outer-boundary cut-vertices within M. Now let D_i be the i^{th} biconnected component connected to any other region C by a simple path $P = vw_1w_2\cdots w_mv'$, such that $\sigma(C) \geq \sigma(D_i)$, where $\sigma(X)$ is the number of concentric level-k faces contained by region X such that each level-k face contains the vertex v'.

We may use Lemma 1 to compute the traversal cost of the simple path joining the two regions. However, before doing so, a further consideration is required:

Every time a simple path P connecting regions C to D_i is traversed, the length of P increases since on reaching D_i the walk traverses the outer boundary therein and returns to v'. If this was not the last level-k face of this region then the region will be revisited once more, but to reach an unvisited vertex of that region it must travel 1 vertex further than before. Therefore, each time the path is traversed, then due to mechanism (v) the path length must be noted to increase by a value of exactly one. Therefore, a given isolated region D_i would require the following number of steps.

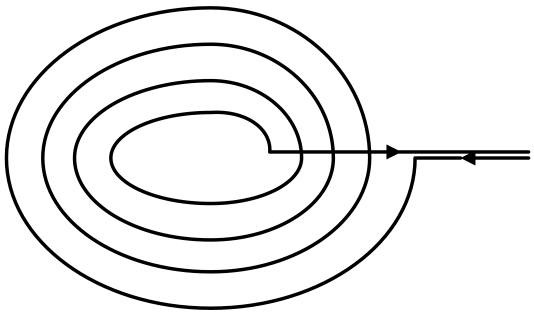

Figure 9: The walk completes isolated regions, unlike in Figure 8

$$
\begin{aligned}
& 2|P_i| + 2(|P_i| + 2) + 2(|P_i| + 3) + \cdots + \\
& \qquad\qquad\qquad\qquad\qquad\qquad 2(|P_i| + \sigma(D_i)) \\
= \;& 2|P_i|\sigma(D_i) + 2(2 + 3 + \cdots + \sigma(D_i)) \\
= \;& 2(|P_i|\sigma(D_i) + \frac{\sigma(D_i)(\sigma(D_i) + 1)}{2} - 1) \\
= \;& \sigma(D_i)(2|P_i| + \sigma(D_i) + 1) - 2,
\end{aligned}
$$

This gives the undesirable result of exiting a biconnected component multiple times, stripping the biconnected component of its level-1 face on every exit (except where additional biconnected components are attached to it, in which case they may persist longer)

It would be much more efficient and desirable if the system completed a biconnected component before exiting (as in Figure 9). To remedy this, the level-1 face algorithm gives priority to unvisited yet known (i.e. sensed) vertices. Visited nodes, in the context of current discussion pertaining to paths connecting biconnected components, are given lower priority. This new mechanism (mechanism (vi)) is in addition to those stated in section 3.3.

Thus, in Figure 10 the biconnected component shown would cause algorithm 1 to consider the cut-vertex dual to the 3-clique as inaccessible. This has the effect of the next level-1 face computed to be that of the interior of the biconnected component. This process continues until the biconnected component is fully explored at which point the region is exited.

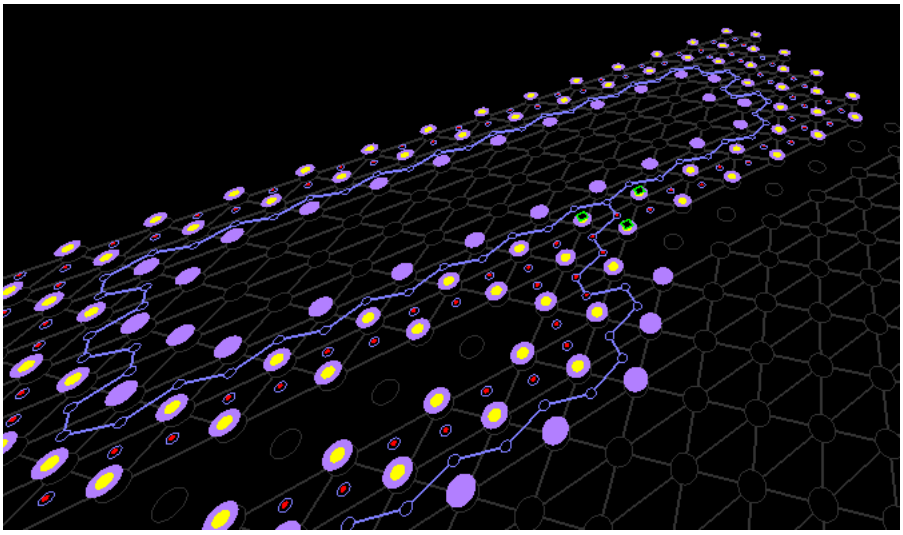

Figure 10: Demonstration of Lemma 1.

This simple addition gives a much improved performance, and will in fact seek out the furthest possible biconnected components, completing biconnected components from the furthest reaches backwards to the starting point. This includes nested biconnected components, meaning that very complex environments are efficiently discovered.

Given the previous discussion and the introduction of mechanism (vi), we can deduce an estimate for a maximum bound of $h(\Omega^*)$,

$$h(\Omega^*) \leq |M^*| + 2\sum_{k=1}^{p-1} |P_k|,$$

where p is the number of biconnected components emerging as M develops and P_k are paths connecting their centres.

4.3. Overall Complexity

Consider the final movement graph M^*. We know there exists an induced subgraph $\Omega \in M^*$, where Ω is the final path taken, such that the 3-clique of robots traverse M^* as optimal as the rules governing algorithm 2 allow. The upper bound of exactly how optimal was given above. Thus, since $|\Omega| \leq |M^*| = n_H$, we are justified in basing all deductions concerning complexity of the algorithm to search an unknown environment on the input size n_H.

Thus, we know that algorithm 2 is called n_H times. Looking at algorithm 2, the very nature of when (if at all) and for what constant of complexity some of the internal functions of algorithm 2 are called depends on the environment. Thus, we may deduce that the algorithm to search an entire unknown environment takes $O(k \cdot n_H)$, where $1 \leq k \leq n_H$ is to be determined and depends entirely on the environment. The trivial example of a square environment, with no internal features, for example, would correspond to $k = 1$.

5. Closing Remarks

This paper gives a solution to the difficult problem of unknown environment search using graph structures and elements of graph theory.

On imposing a virtual structure on the environment, a principle of search, basically amounting to wall following, was developed into a number of algorithms and additional mechanisms were reasoned and applied to achieve a desired result each of which improved efficiency of the search in some way.

The result is a simple, discrete, and robust ready made system of linear time complexity which is both useful in its current form yet allowing room for further development.

The authors believe this to be a novel approach in that the system assigns virtual structure to the environment thus availing pragmatic deployment of entities within the environment and eventual metric map construction. Previous approaches traditionally overlay the topological structure once the environment has been searched and a metric map built.

Future work is to include improvement (possibly by way of convolution) of algorithms, and theoretical improvements of the walk length upper bound. This may itself improve on the already good time complexity. Practical applications on a real world problem (such as robots) is a major goal, and recent work has shown that the outer face walk (algorithm 1) can be optimized to consider only local graph vertices (much like algorithm 4 does), thus reducing to constant time complexity.

Finally, development of algorithms to coordinate n entities for efficient search is desirable, for large team exploration, for example.

References

[1] Alboul, L. S., Abdul-Rahman, H., Haynes, P. S., Penders, J., *An approach to multi-robot site exploration based on principles of self-*

organization, Intelligent Robotics and Applications - Third International Conference, ICIRA 2010, Shanghai, China, November 10-12, 2010. Proceedings, Part II, LNCS 6425, pp.717–729, Springer, 2010.

[2] Alboul, L. and Echeveria, G. and Rodrigues, M., 2004. *Curvature criteria to fit curves to discrete data*. EWCG 19th European Workshop on Computational Geometry.

[3] Baker, B. S, 1994. *Approximation algorithms for NP-complete problems on planar graphs*. Journal of the Association for Computing Machinery, 41:1, pp. 153-180.

[4] Tutte, W. T., 1956. *A theorem on planar graphs*. Transactions of American Mathematical Society, 82, pp. 99-116.

[5] Tutte, W. T., 1977. *Bridges and Hamiltonian circuits in planar graphs*. Aequationes Mathematica, 15, pp. 1-33.

[6] Gerkey, B., Mataric, M., *Multi-robot task allocation: Analyzing the complexity and optimality of key architectures*. In: Proc. of the IEEE International Conference on Robotics and Automation (ICRA) (2003).

[7] Bailey, T.; Durrant-Whyte, H., *Simultaneous localization and mapping (SLAM): part II,*. IEEE Robotics and Automation Magazine, V.13(3), pp. 108-117.

[8] Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun, *A probabilistic approach to collaborative multi-robot localization*. Autonomous Robots, 8(3):325–344, 2000.

[9] Fox, D., Ko, J., Konolige, K., Limketkai, B., Schulz, and D., Stewart, B.: Distributed Multirobot Exploration and Mapping. *Proceedings of the IEEE*, Vol. **94**, No. 7, pp. 1325-1339, (2006).

[10] Howard, A., Mataric, M. J., Sukhatme, G. S.: Localization for mobile robot teams: A distributed MLE approach. In *Experimental Robotics VIII, ser. Advanced Robotics Series*, 146–166, (2002).

[11] Ludwig, L., Gini, M.: Robotic Swarm Dispersion Using Wireless Intensity Signals. In: *Distributed Autonomous Robotic Systems 7*, pages 135–144. Springer Japan, 2007.

[12] Mesbahi, M., Egerstedt, M., 2010, *Graph Theoretical Methods in Multi-agent Networks*. Princeton University Press.

[13] Rekleitis, I., Dudek, G., and Milios, E. : Multi-robot collaboration for robust exploration, Annals of Mathematics and Artificial Intelligence, Vol. 31, pp. 7-40, (2001).

[14] Kurazume, K. R., Hirose, S.: An experimental study of a cooperative positioning system. Autonomous Robots, 8(1):4352, 2000.

[15] Vlassis, N., Papakonstantinou, G., and Tsanakas, P.: Robot Map Building by Kohonen's Self-Organizing Neural Networks. In *Proc. 1st Mobinet Symposium on Robotics for Health*, (1997).

[16] Takamizawa, K., Nishizeki, T., Saito, N., 1980. An Algorithm for Finding a Short Closed Spanning Walk in a Graph. *Networks*, 10, pp. 249-263.