Analysis of Motion Feasibility of Multiple Mobile Agents on Graphs

Ellips Masehian,
Hiva Samadian,
Farzaneh Daneshzand
Tarbiat Modares University, Tehran, Iran
{masehian, h.samadian}@modares.ac.ir; f_daneshzand@aut.ac.ir

ABSTRACT

Solvable Graphs (also known as Reachable Graphs) are types of graphs that any arrangement of a specified number of agents located on the graph's vertices can be reached from any initial arrangement through agents' moves along the graph's edges, while avoiding deadlocks (interceptions). In this paper, the properties of Solvable Graphs are investigated, and a new concept in multi agent motion planning, called *Minimal Solvable Graphs* is introduced. Minimal Solvable Graphs are the smallest graphs among Solvable Graphs in terms of the number of vertices. Also, for the first time, the problem of deciding whether a graph is Solvable for *m* agents is answered.

Keywords - Solvable Graphs, Moving Agents, Motion Planning, Deadlocks

1. INTRODUCTION

The necessity of planning the motions of autonomous agents originally arose in early 1970's, when the first industrial robots were to perform automatic tasks of manipulation and navigation. Soon it was realized that the complexity of the robot motion planning problem is PSPACE-hard and NP-complete since the size of the solution space grows exponentially and gets extremely complicated, especially for high degrees of freedom [1].

When multiple moving agents (e.g. robots) share a common workspace, the motion planning task becomes even more difficult and cannot be performed for just one agent without considering others. In this kind of problems, while pursuing their individual (local) goals, agents must coordinate their motions with each other in order to avoid collisions with obstacles and one another, thus contributing to the task of achieving a global goal, which might be minimizing the total time or distance. This problem is called Multi Agent Motion Planning (MAMP) problem. In MAMP, each agent is regarded as a dynamic obstacle for other agents, and so the element of time plays a major role in planning, especially because of its irreversible nature [2].

Space is the most limiting constraint in a typical MAMP problem: often, because of lack of sufficient space around moving agents, they cannot reach their destinations without obstructing each other's way, causing deadlocks. Deadlocks are situations in which two (or more) agents intercept each other's motions and are prevented from reaching their goals. This happens generally in narrow passageways where autonomous moving agents cannot pass by each other. To resolve such a deadlock, one of the agents should leave and evacuate the passageway (by usually backtracking), and let the opposite agent move out of the passage. This kind of problems is prevalent in large warehouses, plants, and transportation systems, where Automatic Guided Vehicles (AGVs) convey material and products (Fig. 1).

By reducing the workspace into a graph with vertices including the starts and goals of all agents, the MAMP problem can turn into a sequencing problem where the agents are planed to move sequentially (or concurrently) toward their destinations, without colliding with each other. The graph structure stipulates them to remain on predefined routs (i.e. graph edges), and so avoid static obstacles existing in the workspace.

The main question in designing a predefined graph, however, is to find out

whether the graph is 'reachable' (solvable) for any initial and final configurations. *Solvable Graphs* allow the transition of any initial configuration of agents (e.g. pebbles (beans), robots, or vehicles) to a final state via their sequential moves along the graph's edges.

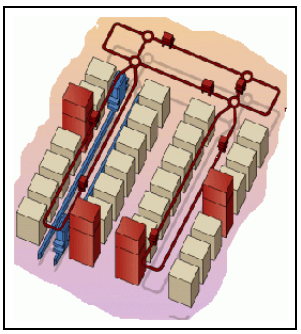


Fig.1. Planning the motions of AGVs to different locations in shop floor is a real-world application of MAMP.

Wilson in [3] worked out a relation between the number of pebbles (k) and the number of vertices (n) of only bi-connected graphs as $k = n - 1$. Kornhauser in [4] improved this result through generalizing the decision problem for all graphs and any number of agents. Auletta et al. in [5] and [6] studied the above problem as *pebble motion problem* by following the generalization of the 15-puzzle and presented a linear algorithm for deciding the reachability of trees. Ryan in [7] studied the possibility of reaching destinations of connected sub-graphs by simplifying the multi robot motion planning between the sub-graphs. He worked on predefined sub-graphs like stack, clique and hall. In [8] it is demonstrated that the environment can be shown through any two-connected graph which has a routing with a practical social law for motion planning.

In this paper, the main problem of finding the maximum number of agents able to navigate on a graph. The topologies and properties of Solvable Graphs are extensively dealt with through a number of lemmas and theorems. Also, considering the fact that the complexity of graph searching operations is directly influenced from the graph size, finding solvable graphs with minimum number of vertices can significantly ease the motion planning task for multiple agents. As a result, the concept of Minimal Solvable Graphs (MSGs) is introduced for the first time in this paper. Accordingly, it can be determined whether a graph is solvable for a certain number of agents or not.

2. Definitions and Assumptions

As mentioned earlier, reducing (or mapping) the configuration space into a graph is very advantageous regarding the significant savings in required time and memory.

In order to lay a proper mathematical foundation for expressing and investigating the properties of graphs, we adopt the standard terminology used in Graph Theory [9]. In addition, some definitions and symbols have been introduced and defined specifically for this work, all presented in Table 1. A number of these concepts are illustrated in Fig. 2.

Table 1. Definitions of used terms and symbols.

Term/Symbol	Description
$\|G\|$	*Order*: The number of vertices of the Graph $G = (V, E)$.
$\|\|G\|\|$	The number of edges on G.
Path	A non-empty graph $P = (V, E)$ of the form $V = \{v_0, v_1, ..., v_k\}$, $E = \{v_0v_1, v_1v_2, ..., v_{k-1}v_k\}$, where all v_i are distinct.
Cycle	A non-empty graph of the form $V = \{v_0, v_1, ..., v_k\}$, $E = \{v_0v_1, v_1v_2, ..., v_{k-1}v_k, v_kv_0\}$, where all v_i are distinct.
Cycle Edge	An edge on a cycle.
Tree	An acyclic subgraph connected to a cycle.
Leaf, L	A vertex with a degree $d = 1$.
Cycle Vertex, C	A vertex located on a cycle.
Internal Vertex, I	A vertex with a degree $d > 1$ which is not a Cycle Vertex.
Stem, S	The longest path in the graph with its one end (or both ends, if located between two cycles) connected to a cycle vertex and including it, and not containing any Cycle Edges. None of the edges of the Stem are cycle edges. If not unique, the Stem is selected arbitrarily. The number of vertices on the Stem (i.e. its order) is shown by $\|I_S\|$.
Configuration	An arrangement of agents on the graph vertices such that no vertex is occupied by more than one agent.

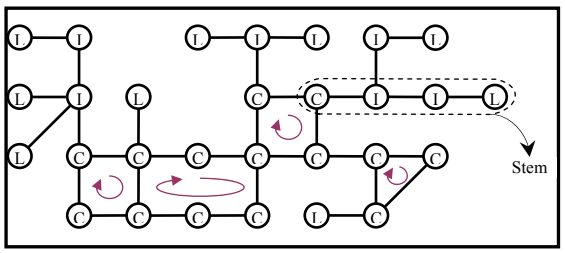


Fig. 2. Some concepts illustrated: L, I, and C denote Leaf, Internal vertex and Cycle vertex, respectively. The symbol ↻ indicates a cycle, and the dashed area designates the Stem.

When designing a graph or networks of routs, it is always important to consider current transportation demands, as well as future developments of the system. In the context of MAMP, this consideration requires that the system designer decides the proper topology of the network, the number of agents (as mobile robots or vehicles) required to move along the routs, and the possibilities of expanding the network for future increased transportation traffic.

Concerned about these issues, we will comprehensively investigate the concept of *Solvable Graphs*. To date, the notion of graph solvability has been essentially depended on the initial and final configurations (situations) of the moving objects. For instance, the question whether a tree-like graph is solvable for a given initial and final configuration of pebbles is solvable or not is addressed in [5] and [6]. However, no work exists in the literature for all types of graphs, and never has the problem of deciding if a graph is *always* solvable for a specific number of agents for *any* initial and final configuration been mentioned or addressed.

In this paper we focus on some related problems, such as:

- What is the maximum number of agents a graph can accommodate such that any final configuration can be reached from any initial configuration?

- What topology must a graph have to be solvable for a specific number of agents?

- What is the 'smallest' graph solvable for a specific number of agents?

Before dealing with the answers to the above questions, three new fundamental and correlated notions are presented below:

Definition 1. A *Solvable Graph* is a graph on which *any* configuration of at most m agents can be reached from any initial configuration through their moves on graph edges, and is shown by SG^m.

Definition 2. A *Partially Solvable Graph* is a graph on which only *some* configurations of m agents can be reached from any initial configuration through their moves on graph edges, and is shown by PSG^m.

Definition 3. A *Minimal Solvable Graph* is the smallest graph on which any configuration of at most m agents can be reached from any initial configuration through their moves on graph edges, and is shown by MSG^m. In this definition, 'smallest' can be expressed and measured in terms of the number of either vertices or edges.

A Compound graph is a combination of Cyclic and Acyclic graphs; that is, it contains at least one loop and at least one leaf. Since Compound graphs constitute a large portion of graphs and have the broadest applications among graphs, we will deal with this kind of graphs in this paper, as a part of our research on graph-based motion planning. Therefore, all graphs mentioned in the next Sections of the paper are Compound graphs.

A) Assumptions

In this paper some simplifying (yet not limiting) assumptions about the graph and agents are made as following:

1. An essential assumption is that the designed graph is finite, connected, planar, undirected, and represents the free space. This means that edges intersect only at vertices.
2. The graph is assumed to be Compound, i.e. has at least one cycle (loop) and at least one Leaf. Actually this assumption is not a restrictive one in most real-world problems since a natural roadmap near a simple disjoint obstacle always forms a loop around it.
3. The initial and final locations of all agents lie on the graph and are known.

4. All agents share the same graph and can (and may) move on the edges of the graph and stay on the vertices of the graph. A *Move* is defined as transferring an agent from a vertex to its neighboring vertex via their connecting edge.
5. Two or more agents may not simultaneously occupy the same vertex in the graph. That is, the vertices are supposed to be spaced sufficiently far apart so that two agents can occupy any two distinct vertices without having collision.
6. Agents have sequential (i.e. one at a time) movements on non-Cycle Edges of the graph. In other words, an agent at vertex v can move to its neighboring Leaf or Internal vertex u only if u is unoccupied. Agents occupying other vertices in the graph do not affect this movement.
7. Agents can have concurrent movements on Cycle Edges of the graph with the following condition: an agent at Cycle Vertex v can move to its neighboring Cycle Vertex u if u is unoccupied, or the agent on u can evacuate the vertex u before the first agent reaches it, or no other agent is approaching vertex u via another edge.

3. SOLVABLE GRAPHS

In Solvable Graphs (SG^m) any configuration of at most m agents can be reached from any initial configuration through agents' moves along the graph's edges. However, a principal question is to determine the maximum number of agents a graph with known topology can accommodate such that any final configuration can be reached from any initial configuration. This question can be rephrased as "what topology a graph must have to be solvable for a specific number of agents?"

It is noted that we are trying to find the *maximum* number of agents a graph is solvable for. Obviously, any graph solvable for m agents is also solvable for $k < m$ agents since there will be more empty vertices and so deadlocks can be resolved more easily.

For finding the maximum number of agents a graph is solvable for, it is essential to investigate the conditions for changing the arrangements of a number of agents through sequential or concurrent moves. Regarding that the graph is assumed to be cyclic, we will first study the solvability conditions of a single cycle, and then expand the results to general Compound graphs through a number of lemmas and theorems.

Imagine a single cycle of k vertices: the total number of distinct configurations of k agents located on the vertices of the cycle is $k!$. However, regarding that the only permitted movements on a cycle is the concurrent clockwise or counterclockwise rotation of agents, only k distinct configurations can be reached from an initial configuration, all of which have the same sequence. It follows that a single cycle is not sufficient for achieving all $k!$ permutations of agents, and so additional vertices are needed for changing the sequence of agents. Lemma 1 formalizes this fact:

Lemma 1. *A specific sequence of agents on a cycle can be reordered iff at least an empty vertex is connected to the cycle.*

Proof. A sequence of agents $a_1, a_2, \ldots, a_k$ can be reordered (rearranged) only when any arbitrary agent, say a_i, could be located between any two other adjacent agents, such as a_j and a_r. This is possible only by removing a_i from the agents' chain (sequence) and reinserting it between a_j and a_r. Apparently, as Fig. 4 illustrates, any outside vertex connected to the cycle (such as vertex u) is a feasible position on which a_i can lay temporarily. If the departure of a_i from the cycle is not possible directly, the whole sequence of agents must rotate until a_i resides on vertex v, after which a_i can move to u. The agents remaining in the cycle further rotate until a_j and a_r locate on both sides of the vertex v. Now a_i can re-enter the cycle, between a_j and a_r.

Conversely, if no vertex exists outside of the cycle, then a_i cannot exit the cycle and therefore cannot locate between a_j and a_r. □

For our future reference, we will define a *Basic Unicycle Graph* (*BUG*) as a single cycle fully occupied by agents connected to one empty leaf (as shown in Fig. 3).

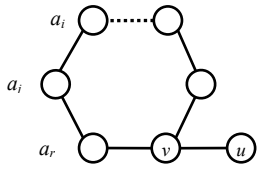


Fig. 3. Illustration for Lemma 1.

As Lemma 1 implies, cycles and their connected empty vertices play a critical role in reordering the agents' sequences, and hence in the solvability of Compound graphs. The connected empty vertices help to make start-to-goal paths of agents as free as possible and facilitate resolving deadlocks.

Lemma 2. *A graph is solvable iff any two agents can interchange their positions.*
Proof. If we show the transformation of a configuration C_1 to another configuration C_2 merely through position interchange of any 2 agents by f_2: $C_1 \rightarrow C_2$, then using the Chain Rule for n agents, the transformation of any initial configuration C_i to any final configuration C_f can be shown by a sequence of 2-agent exchanges, as a compound function f_n: $C_i \rightarrow C_f \equiv f_2^1 \circ f_2^2 \circ \ldots \circ f_2^n$. It follows that if according to the premise of the lemma any 2-agent interchange is possible, then any n-agent interchange is also possible due to the Chain Rule, which means the graph is solvable.

On the other hand, if a graph is solvable, then any configuration is reachable from any initial configuration, a special case of which could be the interchanging of just two agents. This shows that graph solvability and 2-agent interchanging are logically equal. □

Corollary 1. *A Basic Unicycle Graph is solvable iff one vertex in the graph is empty.*
Proof. Regarding the proof of Lemma 1, for m agents on a BUG, any final arrangement is accessible from any initial arrangement, and so the graph is solvable. Conversely, if there is no empty vertex in the graph, then no sequence can be rearranged on the cycle. Therefore, the assumption of no empty vertices is incorrect. □

Corollary 2. *A graph comprised of a chain of i vertices connected to the leaf of a Basic Unicycle Graph is solvable iff $h = i + 1$ vertices in the graph are empty.*
Proof. Let us first consider a graph made of one extra vertex connected to the leaf of a BUG occupied by k agents (as in Fig. 4). If the extra vertex is empty (hence there are k agents and 2 empty vertices in total), then according to Lemma 1 the graph is solvable for k agents. If the extra vertex is occupied by an agent (hence there are $k + 1$ agents and 1 empty vertex in total), then since the new agent cannot enter the cycle, the graph is still solvable only for k agents, and the number of necessary empty vertices will be $h = 2$. By the same logic, the graph is expanded by consecutively appending up to i vertices to the leaf of the BUG, for which the number of necessary empty vertices must increase as much as $i + 1$. If $h < i + 1$, then there would be some agents on the chain that are unable to reach the cycle and hence cannot be reordered. □

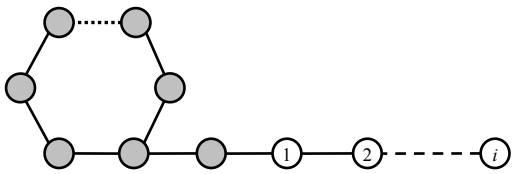


Fig. 4. A Basic Unicycle Graph (gray vertices) is connected to a chain of i vertices.

For a later reference, at this point we define a special type of graph derived from the Basic Unicycle Graph, called an *Extended Unicycle Graph* (*EUG*): An Extended Unicycle Graph is a graph comprised of j chains of vertices each with lengths of $1 \le i \le l_{\max}$ connected to some or all vertices of a Basic Unicycle Graph such that there is no internal vertex with degree $d(I) > 2$. Fig. 5 depicts an example of EUG.

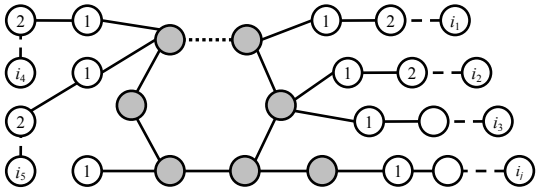


Fig. 5. In a typical Extended Unicycle Graph (EUG) the length of the longest chain is l_{max}.

Lemma 3. *An Extended Unicycle Graph is solvable iff at least $h = l_{max}$ vertices in the graph are empty.*

Proof. We know that the maximum distance from the EUG's cycle to any vertex in the graph is l_{max}. The worst case of interchanging the positions of two agents occurs when an agent a_i located on the vertex with maximum distance from the cycle (i.e. l_{max}) has to move to the depth of another chain of the graph. Since there is only one cycle in the graph, at least the path P connecting the vertex $v(a_i)$ and including the nearest cycle vertex must be either initially empty, or able to be emptied by motions of other agents. By having that many empty vertices, a_i can reach the cycle and concurrently move with other agents of the cycle. Moreover, since the longest chain P is (or can be) emptied, then all the agents on a_i's destination chain can be accommodated on the P, making room for a_i to occupy its final destination vertex. After relocation of a_i, all other agents can return to their original positions via moves in reverse order of their evacuation. This concludes that at least $h = l_{max}$ vertices in the graph should be empty to enable any two agents to interchange and hence make the graph solvable. □

Corollary 3. *A Compound Graph is solvable iff at least $h = l_{max}$ vertices in the graph are empty, where l_{max} is the length (diameter) of the graph's Stem.*

Proof. Any Compound graph with c cycles can be regarded as a set of c distinct Extended Unicycle Graphs. Since the Stem of the whole graph is the longest non-cyclic path with a length of l_{max}, then any path P connecting any agent on the graph to its nearest cycle vertex has at most a length of l_{max}. Because the graph is connected, it is possible to make any non-cyclic path free of agents by having at least $h = l_{max}$ empty vertices in the graph via agents moving. Therefore, concluding from the Lemma 3, all cycles of the graph are solvable, and since each two adjacent cycles share a path between, it is possible to move any agent to any vertex of a 'far' EUG through moving on intermediate cycles located in between.□

Theorem 1. *The maximum number of agents for which a Compound graph is solvable is $m = |G| - |I_S| - 1$, in which $|I_S|$ is the number of Internal Vertices on the Stem.*

Proof. As stated in Corollary 3, the number of necessary empty vertices in a solvable graph must be at least $h = ||S||$, in which $||S||$ is the number of edges on the Stem. Referring to the definition of the Stem in Table 1 and Fig. 3, the length of a Stem can be expressed as $||S|| = |I_S| + 1$, in which $|I_S|$ is the number of its Internal vertices. Regarding that the order of every graph is equal to the sum of vertices occupied by agents and empty vertices, i.e., $|G| = m + h$, it is concluded that $|G| = m + |I_S| + 1$, and therefore the maximum number of agents on a solvable Compound graph is $m = |G| - |I_S| - 1$. □

The result of Theorem 1 serves as a foundation for the next Sections of the paper.

4. CONVERTING SG^M INTO $SG^{M'}$

As discussed in Theorem 1, the maximum number of agents for which a graph can be solvable is determined by the order of the graph, $|G|$, and the number of internal vertices on the Stem, $|I_S|$. On the other hand, sometimes it is desirable to modify a given Solvable Graph SG^m in order to accommodate larger or smaller numbers of moving agents. This happens for instance when the graph represents the routes of Automatic Guided Vehicles (AGVs) on plant floor, or railways connecting urban or rural districts.

Converting an SG^m into $SG^{m'}$ has two aspects:

(1) If $m' > m$, then the SG^m is partially solvable for m' agents (i.e., it is a $PSG^{m'}$). In this case, some vertices

and/or edges must be inserted or relocated to give an $SG^{m'}$.

(2) If $m' \leq m$, then the SG^m is solvable for m' agents. In this case, there might be some redundant vertices and/or edges in the graph which can be truncated or relocated to give a 'lean' $SG^{m'}$.

In this section, mainly the first case, i.e. the problem of converting an SG^m into an $SG^{m'}$ ($m' > m$), is dealt with, where $n = m' - m$ additional agents should navigate on the graph. Regarding that in an SG^m the maximum number of agents is $m = |G| - |I_S| - 1$, accommodating n additional agents requires that the difference $|G| - |I_S|$ be increased by n. The graph expansion/modification is done through four basic operations: Vertex Insertion, Vertex Relocation, Edge Insertion, and Edge Relocation.

It is noteworthy that for the second case above ($m' \leq m$), all of the above basic operations can be performed in reverse order. Precisely, Vertex/Edge Insertion operations change to Vertex/Edge Deletion operations, respectively, and Vertex/Edge Relocation operations remain Vertex/Edge Relocations, but in reverse order.

A) Vertex Insertion

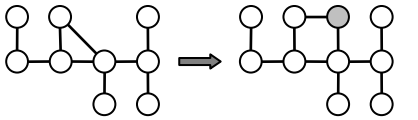
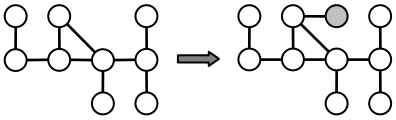
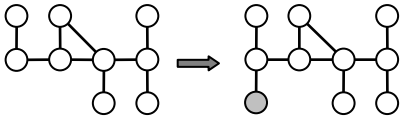
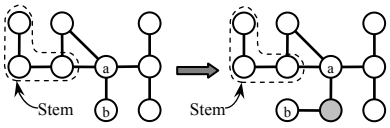
In this operation the difference $|G| - |I_S|$ increases by locating new vertices on the graph in a way that augmenting $|G|$ does not increase $|I_S|$. This is done generally by creating cycle vertices, or inserting new leaves or internal vertices. Table 2 illustrates different variations of converting an SG^m into SG^{m+1} via vertex insertion. For obtaining an SG^{m+n} any combination of these variations should be repeated for n times.

B) Vertex Relocation

In this operation, instead of adding new vertices to the graph, existing vertices plus their connected edges are relocated such that the difference $|G| - |I_S|$ is increased. For converting an SG^m into SG^{m+1}, since $|G|$ remains constant, $|I_S|$ must be reduced by relocating its Leaf to somewhere in the graph other than the Stem (as in Fig. 6(a)), through one of the methods explained in the Table 2. The Vertex Relocation operation must be repeated for obtaining an SG^{m+n}.

It is noted that when the Stem is not unique (as in Fig. 6(b)), relocating its Leaf will not increase the maximum number of navigable agents since an alternative path will be the new Stem with a length equal to the previous Stem. In such cases, Vertex Relocation must be repeated until $|I_S|$ decreases (Fig. 6(c) and 6(d)). Note that this operation must be done in a way that the graph's connectivity is maintained.

Table 2. Possible variations of converting an SG^m into SG^{m+1} through Vertex Insertion.

Description	Graphical Example (for m = 7)	
	SG^7	SG^8
Expanding a cycle		
Inserting a Leaf on a cycle vertex		
Inserting a Leaf on an internal vertex		
Inserting an internal vertex such that $\|I_S\|$ does not increase		

C) Edge Insertion

In this operation no new vertices are added to the graph; instead, new edges connecting existing vertices are inserted such that the difference $|G| - |I_S|$ is increased. For converting an SG^m into SG^{m+n} through the Edge Insertion operation, since $|G|$ remains constant in this operation, $|I_S|$ must reduce by converting the Stem's internal vertices into cycle vertices. As a result, new cycles are created, as shown in Fig. 7.

It is noted that when the Stem is not unique (as {h-e-d} and {h-f-g} in Fig. 7(b)), similar to the Vertex Relocation operation, inserting an edge on the Stem will not increase the maximum number of navigable agents since an alternative path

will be the new Stem with a length equal to the previous Stem. In such cases, edge insertion must be repeated until $|I_S|$ decreases (Figs. 8(c) and 8(d)).

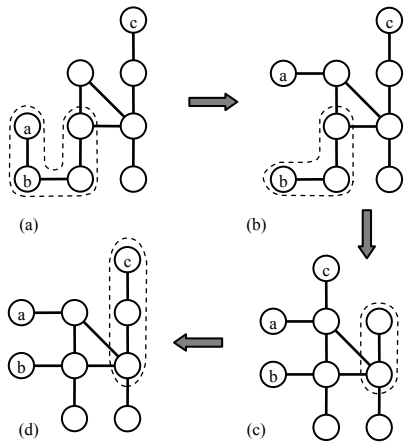


Fig. 6. Graph modification through Vertex Relocation. Dashed areas indicate Stems. The graphs are SG^6, SG^7, SG^7, and SG^8, respectively from (a) to (d).

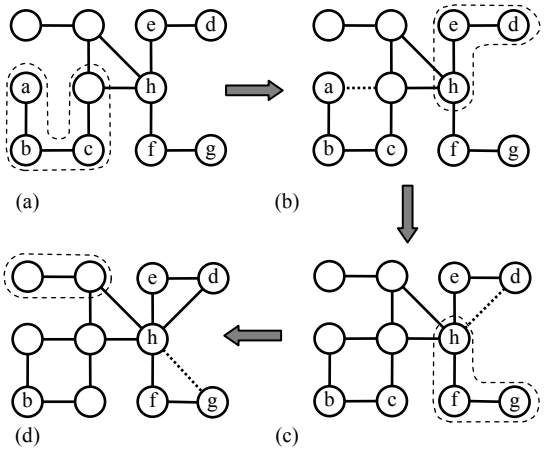


Fig. 7. Graph modification through Edge Insertion. Dashed areas indicate Stems, which change as new edges are inserted in the graph. The graphs are SG^8, SG^9, SG^9, and SG^{10}, respectively from (a) to (d).

D) Edge Relocation

In this operation, instead of adding new edges to the graph, existing cycle edges are relocated to create 'longer' cycles (i.e. cycles with larger number of vertices within) such that the difference $|G| - |I_S|$ is increased. For converting an SG^m into SG^{m+n} through the Edge Relocation operation, since $|G|$ remains constant, $|I_S|$ must reduce by converting the Stem's internal vertices into cycle vertices, as shown in Fig. 8. Note that this operation must be done in a way that the graph's connectivity is maintained.

As mentioned earlier, when the Stem is not unique, relocating a cycle edge will not increase the maximum number of navigable agents since an alternative path will be the new Stem with a length equal to the previous Stem. In such cases, Edge Relocation must be repeated until $|I_S|$ decreases.

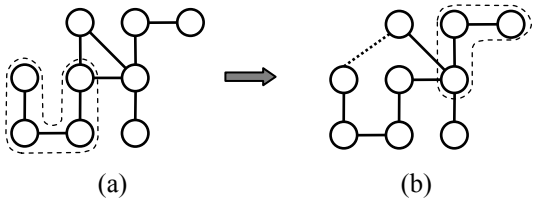


Fig. 8. Graph modification through Edge Relocation. Dashed areas indicate Stems. The graphs are SG^6 and SG^7, respectively.

It should be noted that in real world applications, where existing of multiple cycles or cycles with large number of vertices are not practically feasible, instead of expanding the number or size of cycles, we may expand the trees connected to cycles of graphs such that the difference $|G| - |I_S|$ remains unchanged, and the graph remains solvable for the same number of agents.

5. MINIMAL SOLVABLE GRAPHS

For a specific number of agents (m), a notable subclass of Solvable Graphs SG^m is the set of Minimal Solvable Graphs (MSG^m) which have the minimum number of vertices necessary for accommodating m agents. Considering that the complexity of graph searching operations is directly influenced from the graph size, finding Minimal Solvable Graphs would significantly ease the tasks of graph designing and multi agent motion planning. In this Section the topologies of Minimal Solvable Graphs are introduced through a number of theorems. Also, a special subset of MSGs are identified which have the minimal number of edges.

Theorem 2. *The minimum number of vertices for a graph to be solvable is $m + 1$.*

Proof. As stated in the Theorem 1, the maximum number of agents in a solvable graph is $m = |G| - |I_S| - 1$, and so $|G| = m + |I_S| + 1$. For minimal number of vertices, $|I_S|$ must take the least possible value, which is 0. It follows that $|G| = m + 0 + 1 = m + 1$, and so MSG^{m}s have $m + 1$ vertices. □

Corollary 4. *Minimal Solvable Graphs do not contain any internal vertices.*

Proof. As proved in Theorem 2, the Stem in MSGs does not contain internal vertices (i.e. $|I_S| = 0$). On the other hand, since the Stem is the longest non-cyclic chain and no other path may have more internal vertices than it, then there should be no internal vertices in the entire graph. □

Corollary 5. *An MSG^m is not solvable for more than m agents.*

Proof. Since an MSG^m has $m + 1$ vertices, placing one more agent on the graph will make the graph fully occupied, and hence no sequential moves would be possible. □

Theorem 3. *An MSG^m with c cycles has $m + c$ edges.*

Proof. According to the Euler's Formula, for a connected planar graph with V vertices, E edges, and F faces, the following equation holds: $V - E + F = 2$ (proved in [9]). Each cycle divides the space into two faces: a finite face enclosed in the cycle, and an infinite face outside of the cycle. In the context of our definitions and notations, by excluding the infinite face from both sides of the Euler's Formula, it can be rewritten as

$$|G| - ||G|| + c = 1. \quad (1)$$

As proved in Theorem 3, in an MSG^m, $m = |G| - 1$. Therefore,

$$m + 1 - ||G|| + c = 1, \quad (2)$$

$$\Rightarrow \quad ||G|| = m + c, \quad (3)$$

which proves the theorem. □

Corollary 6. *An MSG has minimal edges if there is only one cycle in the graph.*

Proof. It was proved previously that an MSG^m containing c cycles has $m + 1$ vertices and $m + c$ edges. The number of edges is minimal when $c = 1$, i.e. there is be only one cycle in the graph. □

For constructing MSG^m, $m + 1$ vertices must be connected such that at least one cycle and no internal vertices are formed. As an example, all possible topologies of Minimal Solvable compound Graphs for $m = 6$ agents are shown in Fig. 9. All these graphs have $6 + 1 = 7$ vertices, 0 internal vertices, and $6 + c$ edges, in which c is the number of cycles.

In order to obtain various topologies for MSGs, a number of transformation operations are worked out and illustrated in Table 3.

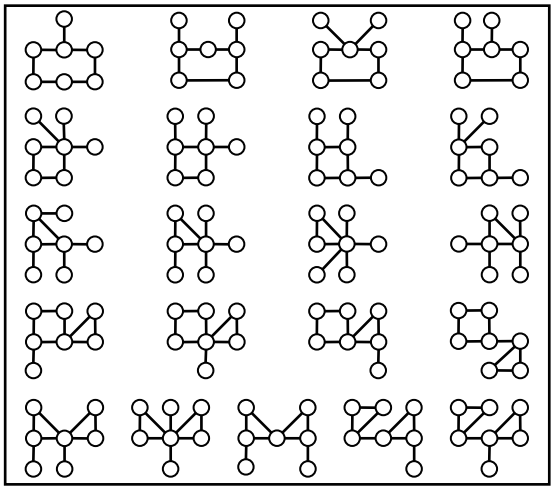
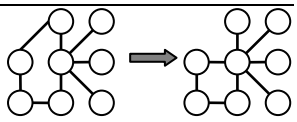
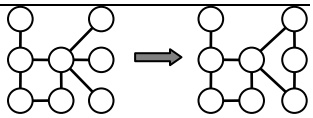
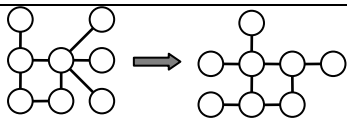
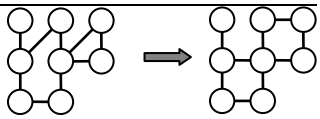


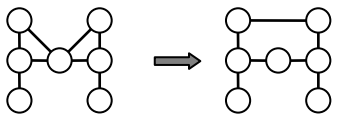
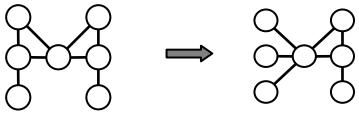
Fig. 9. All possible Minimal Compound Solvable Graphs for $m = 6$ agents (MSG^6).

Table 3. Operations for creating different topologies of MSGs.

Description	Graphical Examples (for $m = 7$)
Converting a cycle vertex into a leaf connected to a cycle	
Converting a leaf connected to a cycle into a cycle vertex	
Relocating a leaf connected to a cycle between cycle-vertices	
Transforming cycles with lengths of c_1 and c_2 into cycles with lengths $c_1 - 1$ and $c_2 + 1$	

Since MSG^{m}s have $|G| = m + 1$ vertices and $||G|| = m + c$ edges, and regarding that compound graphs have at least one cycle, Minimal Solvable Graphs with one cycle have minimal number of edges, as well as vertices. Therefore, for minimizing the edges of an existing MSG^m, multiple cycles must be decomposed into one cycle, via operations described in Table 4.

Table 4. Methods of creating MSGs with minimal number of edges.

Method	Description	Graphical Examples (for $m = 6$)
Vertex Relocation	Creating larger cycles instead of two or more cycles	
	Converting cycles vertices into leaves connected to cycles	
Vertex Deletion	Deleting common edges in cycles	

6. DISCUSSION AND CONCLUSION

The time complexity of the proposed method for verifying the solvability of a graph is in $O(n^2)$ which is spent on detecting cycles and identifying the Stem, where n is the number of graph vertices. Also, calculating the maximum number of agents operable on a given graph takes is performed in the same time order. In contrast, investigating the solvability of a Multi Agent Motion Planning problem of m agents on a graph with n vertices through exhaustive enumeration will require $\frac{n!}{(n-m)!}$ different permutations of agents to be checked, which is far beyond the time order of the presented algorithm.

Moreover, verifying whether a graph is SG^m would require $\sum_{i=1}^{m}\left(\frac{n!}{(n-i)!}\right)^2$ operations to be checked, for any initial and final configurations, which is again exponentially time consuming. These figures demonstrate the effectiveness of our findings in terms of required time and memory.

In designing transportation networks for multiple autonomous agents (such as mobile robots, AGVs, cars, etc.) which can merely move along the network's arcs, it is important to make sure that the graph has a proper topology and sufficient number of vertices (relative to the number of agents) to enable the agents move planning. This paper deals with the topology of Solvable Graphs and introduces the new concept of Minimal Solvable Graphs and investigates their properties, which are the smallest graphs that satisfy the feasibility conditions for multi agent motion planning for any initial and final configurations of agents.

References

[1] J. F. Canny, *The Complexity of Robot Motion Planning*, The MIT press, Cambridge, Mass., 1988.

[2] J.C. Latombe, *Robot Motion Planning*, Kluwer Acad. Pub., London, 1991.

[3] R.M. Wilson, "Graph puzzles, homotopy, and the alternating group", *Journal of Combinatorial Theory*, Series B, 1974, Vol. 16, pp. 86-94.

[4] D. Kornhauser, G. Miller, and P. Spirakis, Coordinating pebble motion on graphs, the diameter of permutations groups and applications", in *Proc. 25th IEEE Symp. On Found. Comp. Sci.,* (1984), pp. 241-250.

[5] V. Auletta, A. Monti, D. Parente and G. Persiano, "A Linear-Time Algorithm for the Feasibility of Pebble Motion on Trees", *Algorithmica*, 23(3) (1999), pp. 223-245.

[6] V. Auletta, D. Parente and G. Persiano, "A New Approach to Optimal Planning of Robot Motion on a Tree with Obstacle", *in Proc. 4th European Symposium on Algorithms (ESA),* Spain, 1996, pp. 25-27.

[7] M.R.K. Ryan, "Multi-robot path planning with subgraphs", in *Proc. 19th Australasian Conf. Rob. Autom.,* New Zealand, (2006).

[8] *S.* Onn, and M. Tennenholtz, "Determination of social laws for agent mobilization", *Artificial Intelligence*, Vol. 95, (1997), pp. 155-167.

[9] R. Diestel, *Graph Theory*, Springer-Verlag, New York, 2000.