# Designing Solvable Graphs for Multiple Moving Agents

Ellips Masehian[a,*], Farzaneh Daneshzand[a]

[a] *Industrial Engineering Dept., Tarbiat Modares University, Tehran, 14115-317, Iran*

Received 3 Nov., 2008; revised 4 Dec., 2008; accepted 12 Feb. 2009

**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, and a new algorithm is presented for making an existing graph solvable and lean for a given number of agents. Finally, through an industrial example, it is demonstrated that how the findings of this paper can be used in designing and reshaping transportation networks (e.g. railways, traffic roads, AGV routs, robotic workspaces, etc.) for multiple moving agents such as trains, vehicles, and robots.

*Keywords:* Solvable Graphs, Intelligent Moving Agents, Motion Planning, Deadlocks;

## 1. Introduction

Future generation autonomous agents are expected to operate in remote and dangerous places like outer space, undersea, hazardous waste sites, and are therefore anticipated to be far more self-directed than today's existing agents. The ability of an agent to plan its own motion seems pivotal to its autonomy. For over more than three decades, agent motion planning in general, and robot motion planning in particular, have attracted much research in various fields and have become central topics in autonomous agents and artificial intelligence. Although today the term 'motion planning is considered to cover a wide variety of problems, we will use it for the problem of planning collision-free motions for an autonomous agent moving among obstacles.

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 [3].

Many techniques have been developed for solving the Robot motion planning problem, including the 'Skeleton' Or 'Roadmap' approach [4]. In this approach, the continuous workspace is mapped into a one-dimensional graph with vertices including the start and goal of the robot, and edges as paths between vertices. This graph is then searched to find a collision-free start-to-goal path. *Visibility Graph* is a type of roadmap which is the collection of lines in the free space connecting vertices of an object to those of another (Fig. 1(a)). There are $O(n^2)$ edges in the Visibility Graph, and it can be constructed in $O(n^2)$ time and space in 2D, where $n$ is the number of vertices. *Voronoi Diagram* is another roadmap defined as the set of points equidistant from two or more objects (Fig. 1(b)). The Voronoi Diagram partitions the space into regions, where each region contains one object. For each point in a region, the object within the region is the closest to that point than any other object. There are only $O(n)$ edges in the Voronoi Diagram, and it can be efficiently constructed in $\Omega(n\log n)$ time, where $n$ is the number of objects [7].

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

[*] Corresponding author. Tel.: +98-21-82883987; e-mail: masehian@modares.ac.ir.

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 therefore the element of time plays a major role in planning, especially because of its irreversible nature [9].

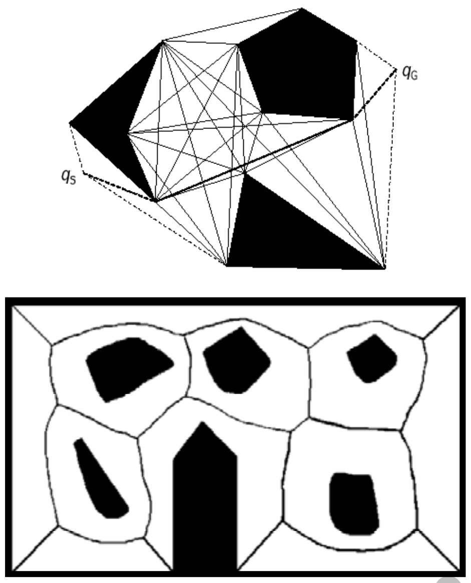


Fig.1. (up) Visibility Graph, and (down) Voronoi Diagram.

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 therefore the element of time plays a major role in planning, especially because of its irreversible nature [9].

Although classic roadmaps like Visibility Graph and Voronoi diagram have proved to be effective for single-agent problems, they do not provide straightforward solutions to MAMP problems, including the corridor-like environment shown in Fig. 2(a). This kind of environments is prevalent in large warehouses, plants, and transportation systems, where Automatic Guided Vehicles (AGVs) convey material and products (Fig. 2(b)).

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.

A well-known cooperative behavior of agents is following independent start-to-goal paths while resolving deadlocks by reshuffling, circumnavigating, detouring, and speed regulating (also known as Velocity Tuning). Another approach developed for resolving deadlocks is the Prioritized Planning, in which the agents are sorted by their moving priorities. Higher-priority agents are planned first, whereas lower-priority agents plan their motion subsequently by considering higher-priority agents as dynamic obstacles.

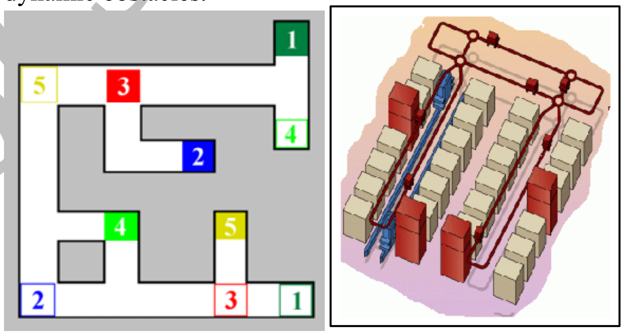


Fig.2. (left) Each agent must move toward its final destination while allowing other agents to reach their goals. (right) Planning the motions of AGVs to different locations in a shop floor or warehouse is a real-world application of MAMP.

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.

Wilson in [14] 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 [8] improved this result through generalizing the decision problem for all graphs and any number of agents. Auletta et al. in [1] and [2] 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 [13] 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 [12] 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.

During our research on optimal multi agent motion planning on graphs we encountered 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. We then develop a new algorithm for making a Solvable Graph 'leaner' and smaller while it remains solvable for the same number of agents before and after performing the algorithm. Lastly, the presented theories are applied in a practical industrial example.

## 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 [5]. 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.

Generally, an MAMP in a continuous space is composed of the following phases:

(a) Constructing a network (graph) with nodes (vertices) including the agents' initial and final stopping locations, together with all the locations the agents should be able to temporarily stop at and offer a service. The arcs (edges) of the graph must pass through free spaces (e.g. unoccupied rooms or corridors) such that no obstacle may be collided with during the agents' motions along the edges.

(b) Planning the agents' motions along the arcs of the network from their starting to final locations, such that a cost criterion (e.g. time, distance, or expense) is minimized.

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. |

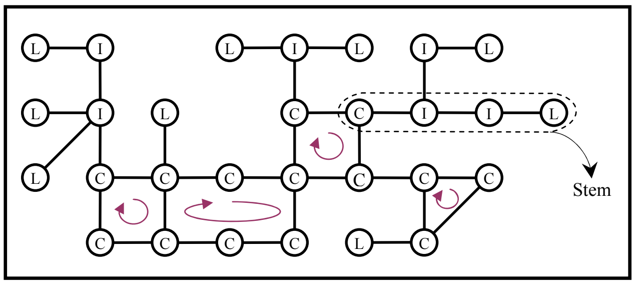


Fig. 3. Some concepts illustrated: L, I, and C denote Leaf, Internal vertex and Cycle vertex, respectively. The symbol ↻ indicates a cycle, and the dashed area designates the Stem.

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 [1] and [2]. 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 the first phase of MAMP, that is, the graph construction and optimization, and propound some related problems of broad scope, 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.

In the Graph Theory literature, graphs are categorized into two general classes: Cyclic graphs (having loops), and Acyclic graphs (without loops, i.e. trees). However, based on our findings in this and other relevant papers, we propose a more precise categorization of graphs, as: (1) Acyclic graphs, (2) Purely Cyclic graphs, and (3) Compound graphs. A Purely Cyclic graph is defined as graph with merely cyclic edges, and therefore has no leaves or internal vertices. A Compound graph is a combination of Acyclic and Purely Cyclic 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. The properties and topologies of solvable acyclic graphs are presented in our previous research, in [10] and [11]. Therefore, all graphs mentioned in the next Sections of the paper are Compound graphs.

### *2.1. Assumptions*

In the phase of graph construction, the topological, and not geometric, features of the world are important; features like the number and degrees of vertices, existing of edges between certain pairs of vertices, existing of loops and their sizes, etc. Nevertheless, the geometrical features of the world, such as the exact coordinates of the vertices, the lengths of edges, etc. become decisive in the phase of motion planning.

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, and represents the free space. This means that edges intersect only at vertices.
2. The graph is assumed to be a Compound graph, 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. An important issue, however, is to identify the cycles of the graph and verify that the graph is cyclic. A straightforward method for cycle detection is by performing a Depth-First Search in the graph: starting from a desired vertex, all children of that vertex is explored in depth, and so on. If a previously visited vertex is encountered, then all the vertices explored between the two visits of that vertex are identified as cycle vertices. The running time of this method is the same as that of the Depth-First Search. A number of cycle-detecting strategies are reviewed in [6].
3. The graph is undirected, and a path exists from any vertex *v* to *u* and vice versa.
4. The initial and final locations of all agents lie on the graph and are known.
5. 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.
6. 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.
7. 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.

8. 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 $a_1$, $a_2$, …, $a_k$ 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 in Fig. 4).

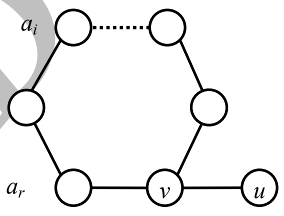


Fig. 4. Illustration for Lemma 1.

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 $f_n$: $C_i \rightarrow C_f \equiv f_2^1 \circ f_2^2 \circ \ ... \circ \ f_2^n$. 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. 5). 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. □

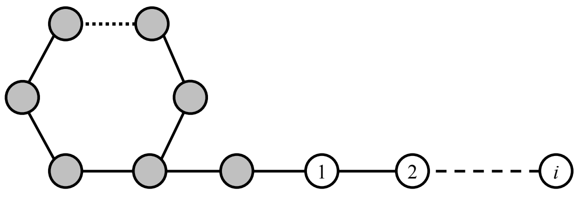


Fig. 5. A Basic Unicycle Graph (gray vertices) is connected to a chain of $i$ vertices.

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 $1 \leq i \leq l_{\max}$ connected to some or all vertices of a Basic Unicycle Graph such that there is no internal vertex with degree $d(I) > 2$. Fig. 6 depicts an example of EUG.

**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

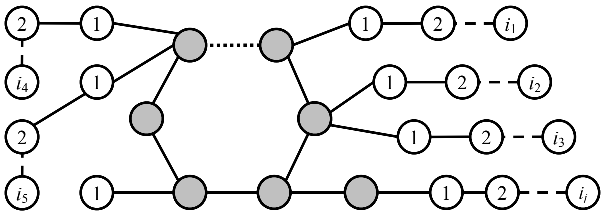


Fig. 6. In a typical Extended Unicycle Graph (EUG) the length of the longest chain is $l_{\max}$.

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.

### *4.1. 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.

### *4.2. 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. 7(a)), through one of the methods explained in the Table 2. The Vertex Relocation operation must be repeated for obtaining an $SG^{m+n}$.

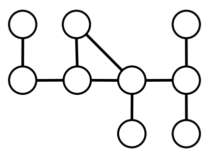
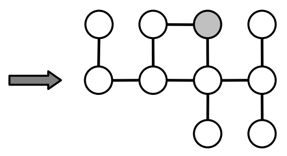
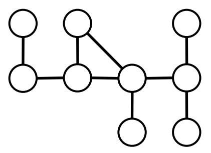
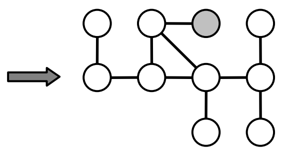
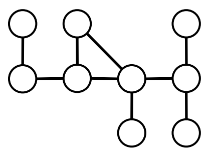
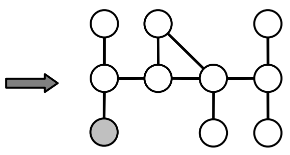
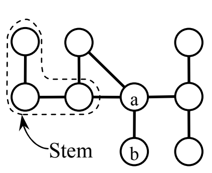
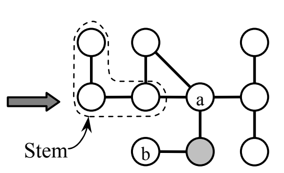
It is noted that when the Stem is not unique (as in Fig. 7(b)), relocating its Leaf will not increase the maximum number of navigable agents since an alternative path will be the new Stem with a length equal to the previous Stem. In such cases, Vertex Relocation must be repeated until $|I_S|$ decreases (Fig. 7(c) and 7(d)). Note that this operation must be done in a way that the graph's connectivity is maintained.

### *4.3. 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. 8.

It is noted that when the Stem is not unique (as {h-e-d} and {h-f-g} in Fig. 8(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)).

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 |  |  |

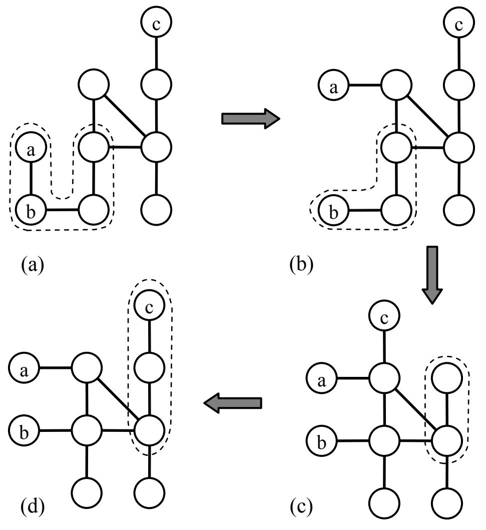


Fig. 7. Graph modification through Vertex Relocation. Dashed areas indicate Stems. The graphs are $SG^6$, $SG^7$, $SG^7$, and $SG^8$, respectively.

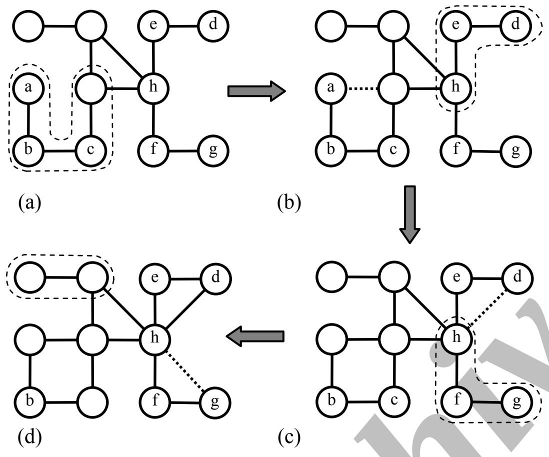


Fig. 8. Graph modification through Edge Insertion. Dashed areas indicate Stems, which change as new edges are inserted in the graph. The graphs are $SG^8$, $SG^9$, $SG^9$, and $SG^{10}$, respectively.

### 4.4. 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. 9. 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.

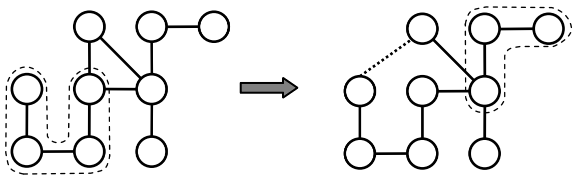


Fig. 9. Graph modification through Edge Relocation. Dashed areas indicate Stems. The graphs are $SG^6$ and $SG^7$, respectively.

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 [5]). 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

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

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

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

$$\Rightarrow \quad m - ||G|| + c = 0, \qquad (3)$$

$$\Rightarrow \quad ||G|| = m + c, \qquad (4)$$

which proves the theorem. □

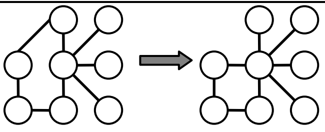
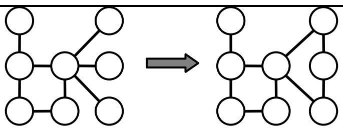
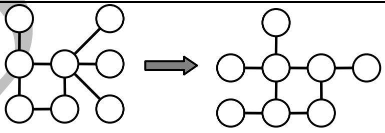
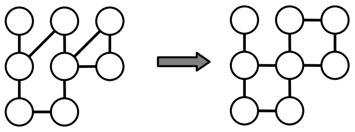
**Corollary 6.** *An MSG has minimal edges if there is only one cycle in the graph.*

*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 MSGm, 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. 10. 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.

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.

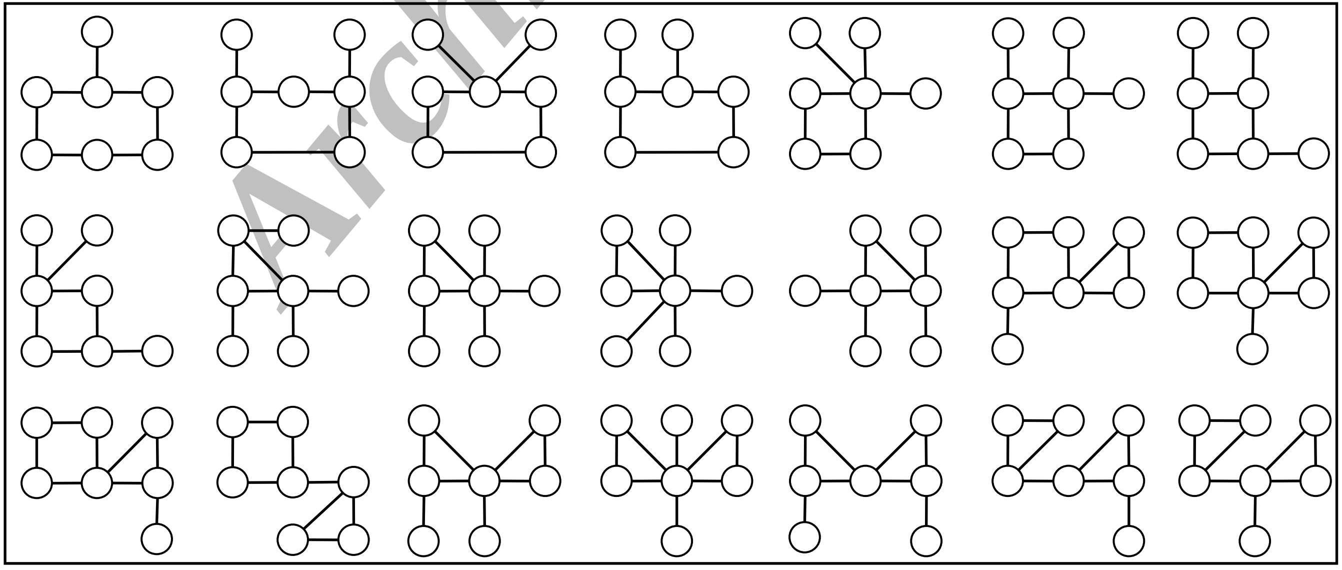
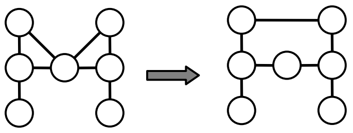
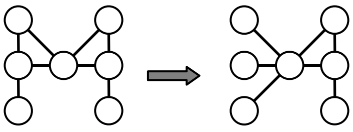
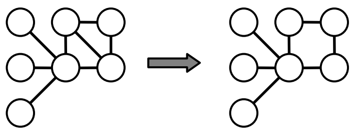


Fig. 10. All possible Minimal Compound Solvable Graphs for $m$ = 6 agents (MSG$^6$).

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 |  |

Although the results of the previous sections are of theoretical nature, they can be used straightforwardly in real world applications, such as designing and reshaping transportation networks, railways, traffic roads, AGV routs, and robotic workspaces for multiple moving agents such as trains, vehicles, and robots.

On the other hand, in order for a graph to be utilized as the structure of a real network, its topology must be suited to the application it is designed for. Thus, efficiency is an important issue in designing and tailoring applied graphs.

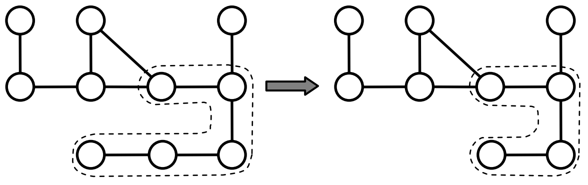
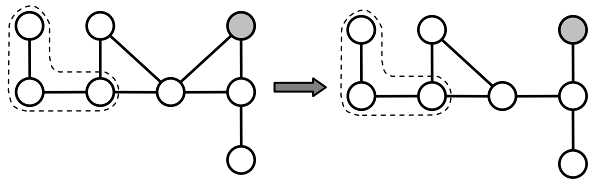
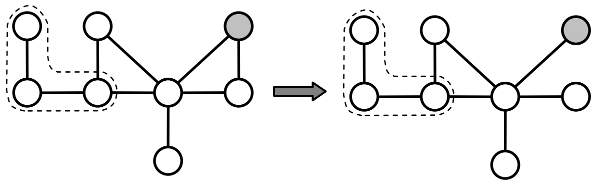
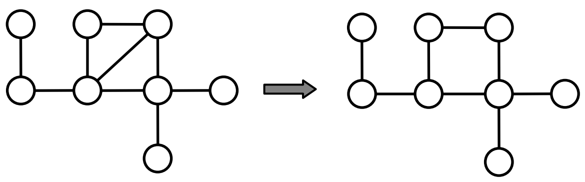
A graph is considered to be efficient if, in addition to being solvable, has no or very few redundant edges and vertices. Lacking redundant elements (i.e. edges and vertices) in a solvable graph becomes substantially significant particularly when the cost of constructing real world counterparts of graph elements (such as roads, railways, canals, etc.) is remarkably high.

In order to implement the findings of the previous sections in making an efficient graph, a new algorithm is developed in this section, which operates in two phases: (a) Compatibility phase, and (b) Rightsizing phase.

## 6. Designing Lean Graphs

Table 5

Possible operations in the Rightsizing Phase.

| Method | Description | Results | Graphical Examples (for *m* = 6) |
|---|---|---|---|
| Vertex Deletion | Deleting the Stem's leaf if the Stem is unique | The Stem becomes shorter |  |
| Edge Deletion | Deleting the cycle edge if it does not change the Stem | A cycle vertex is converted into a leaf connected to an internal vertex |  |
| | | A cycle vertex is converted into a leaf connected to a cycle vertex |  |
| | | No change in the type of the vertex |  |

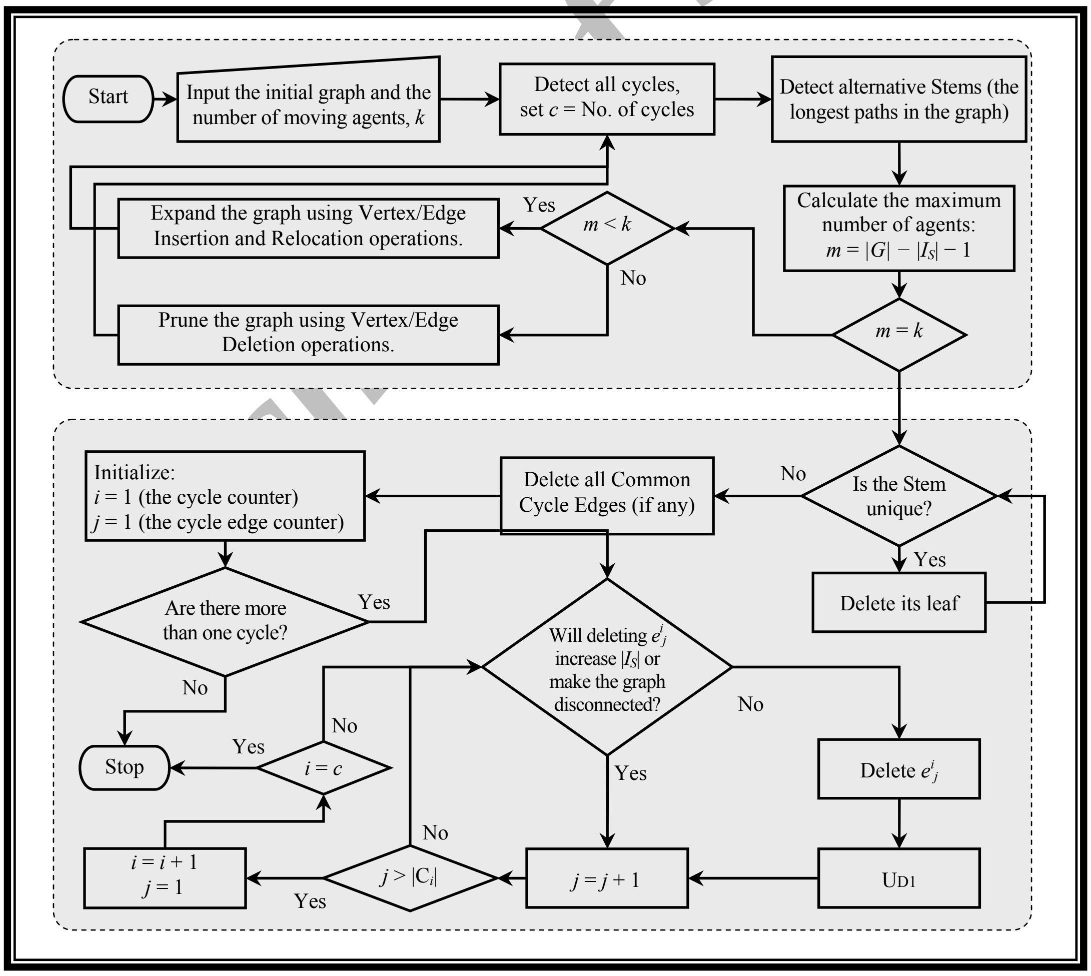
In the Compatibility phase, the given graph is made compatible with the needs and constraints of the application. This may include making the graph solvable for the specified number of agents by expanding it if it's not solvable, or reducing the size of the graph if it is solvable for an excessive, unnecessary number of agents. The output of this phase is a Solvable Graph for exactly the predefined number of agents, e.g., $k$.

In the Rightsizing phase, the $SG^k$ is further examined to find any redundant edges or vertices for pruning. We will call the resultant graph a 'Lean' graph since it is fully operational and optimized for the application at hand.

In expanding the size of the graph in the Compatibility phase special care must be taken to comply with the limitations and conditions of the application. For example, there might be no sufficient space for inserting a vertex or edge, or that inserting an edge is infeasible economically or technically. In case that the graph is not capable of accommodating the specified number of agents, a possible engineering solution could be either reducing the number of agents or inserting the required graph elements such that the sustained expense is minimized.

On the other hand, in reducing/pruning the graph, some vertices might be indelible as they are essential to the system, as the positions for loading, feeding, or parking machines or vehicles. Therefore, possible operations in the Rightsizing phase are Vertex Deletion and Edge Deletion operations, and Vertex/Edge Relocation and Insertion are not considered. The results of Vertex/Edge Deletion operations are illustrated in Table 5.

It must be noted that although deletion of redundant vertices or edges from the graph may reduce the overall cost of network design and construction, it may lead to higher costs of agents' movements as well, since by possessing a more limited space for maneuvering and reshuffling, agents may be forced to travel / stop more frequently for resolving deadlocks. The details of the algorithm is explained in the flowchart illustrated in Fig. 11.



Start
Input the initial graph and the number of moving agents, $k$
Detect all cycles, set $c$ = No. of cycles
Detect alternative Stems (the longest paths in the graph)
Calculate the maximum number of agents: $m = |G| - |I_S| - 1$
$m = k$
$m < k$
Yes
No
Expand the graph using Vertex/Edge Insertion and Relocation operations.
Prune the graph using Vertex/Edge Deletion operations.
Is the Stem unique?
No
Yes
Delete its leaf
Delete all Common Cycle Edges (if any)
Initialize: $i = 1$ (the cycle counter) $j = 1$ (the cycle edge counter)
Are there more than one cycle?
Yes
No
Will deleting $e^i_j$ increase $|I_S|$ or make the graph disconnected?
No
Yes
Delete $e^i_j$
UD1
$j = j + 1$
$j > |C_i|$
No
Yes
$i = i + 1$ $j = 1$
$i = c$
No
Yes
Stop

Fig. 11. Flowchart of the algorithm for designing a lean graph. $|C_i|$ is the order of the $i^{th}$ cycle. The dashed gray areas include the two main phases, the Compatibility and Rightsizing phases.

## 7. Practical Application

In this section a practical illustrative example is provided for demonstrating a typical application of the results of the previous sections. Specifically, it is shown how designing a Solvable Graph and then optimizing it might help solving an industrial shop floor problem.

Suppose a factory is consisted of a workshop and two warehouses, one for raw material and one for finished products. Te production area accommodates two pressing, three drilling, and three grinding machines, as well as other equipment for milling, forging, turning and assembly, arranged as departments. Each of these machines or departments requires certain amount of raw material (or unfinished parts) to be loaded, processed, unloaded and then transported to other divisions of the factory. The layout of the factory and the loading/unloading positions for each machine and department are shown in Fig. 12.

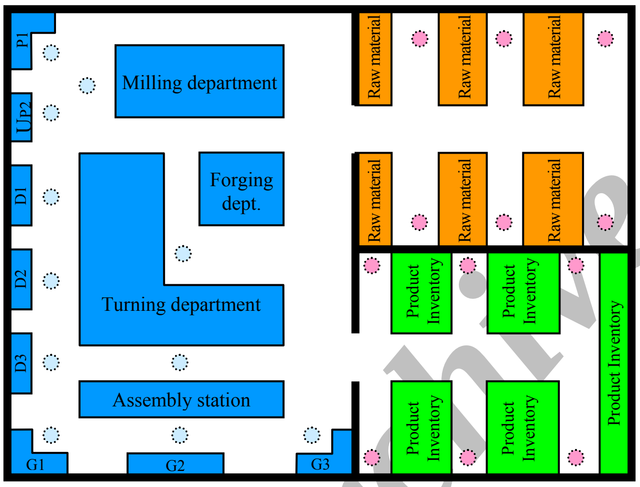


Fig. 12. Layout of the factory. The specific locations for loading / unloading parts and material are shown with small circles.

Currently the material and products are carried manually by carts and pallets, but the management decides to upgrade the factory's transportation system and utilize Automatic Guided Vehicles (AGVs). However, there are a number of constraints and conditions in this regard.

**Constraints**:

1. The AGVs can only move along predefined paths realized by embedding special type of wires in the floor and covering them with epoxy resins for safety and durability.
2. Due to space limitation, no two AGVs can move along a single passageway or corridor side by side, and should therefore wait until the passageway (i.e. the wire track) is evacuated and emptied from other AGVs.
3. AGVs can only stop at certain locations for loading/unloading or changing their tracks. They cannot stay at any point on the track.
4. Since the factory's production system is Job-Shop, the volume of interdepartmental transportation is high. Therefore, all AGVs need to have access to all points of the transportation network. In other words, all AGVs must be able to reach any destination from any initial position.
5. Each AGV is equipped with some batteries that actuate it and provide power for its sensors and control unit, and therefore needs to be recharged daily. The recharging is done during the factory's idle times, i.e. at nights. However, for safety reasons, recharging hubs are located in warehouses so that AGVs can be recharged and kept secured at nights.
6. The cost of laying out wire tracks is proportionate to their lengths. Also, situating each stop point requires additional expense due to the special sensors and circuits needed. These initial costs are much higher than the costs of increased movements the AGVs need to do for avoiding collisions or deadlocks.
7. The management has decided to carry out the upgrade in two phases: in the first phase, 12 AGVs are planned to operate in the factory, and in the second phase, in parallel with increasing the production volume and variety, the number of AGVs will be increased to 18.

**Problem:**

The problem is to design a network of wire tracks and stopping stations such that the total cost of upgrading is minimized while the above constraints are fulfilled. In other words, it is desired to plan a Solvable Graph for 12 agents (for the first phase) which is efficient in terms of the number of vertices and edges.

**Solution:**

As a starting point, the Voronoi Diagram of the factory is calculated, which is the collection of points equidistant from two or more facilities or walls (refer to Section 1 for more details). Implementing the Voronoi Diagram is quite proper since it yields the safest paths passing through narrow corridors and production facilities. Fig. 13 depicts the Voronoi Diagram of the factory.

As can be seen in the Fig. 13, the Voronoi Diagrams has some small redundant edges connected to concave corners of the workspace. These edges, however, can be easily pruned (removed) from the diagram. Also, there are some vertices very close to each other (as shown in dashed circles in the Fig. 13) which can be merged for simplicity and practicality.

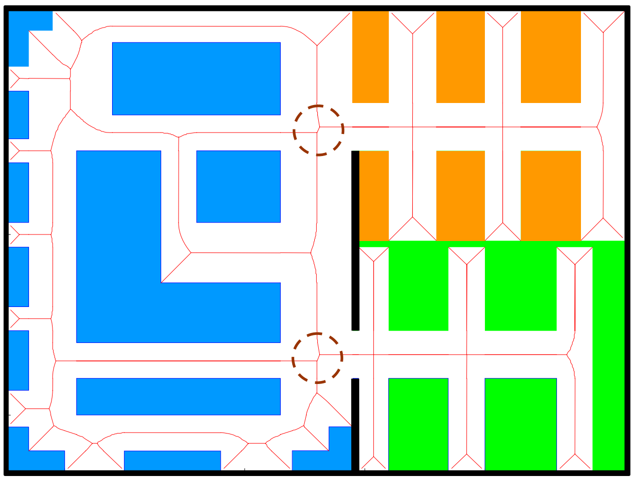


Fig. 13. Voronoi Diagram of the factory, which is the collection of safest paths amid facilities.

The next step is to modify the pruned Voronoi Diagram in order to comply with the requirements of loading/unloading stations imposed by the constraint 3. This is done by superimposing the Voronoi Diagram and the stations' locations, as illustrated in Fig. 14.

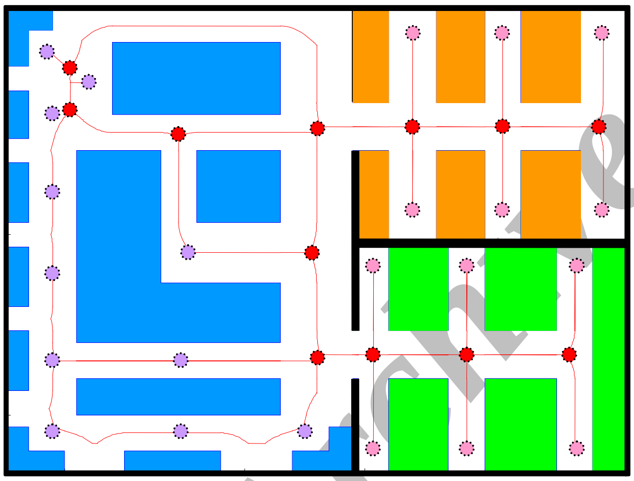


Fig. 14. Superimposition of the Voronoi Diagram and stopping stations.

Nearly all stopping stations situate at proper positions on the Voronoi Diagram, except for the upper-left corner, for which we proposed a simplifying solution depicted in Fig. 15.

As stated in the constraint 7 above, the management has decided to operate the network for 12 AGVs in the first phase. By taking into consideration all possible vertices of the network, the maximum number of AGVs that can travel in the whole factory is $m = 33 - 3 - 1 = 29$. Although in this case a large number of AGVs can operate, they have to make use of all vertices (including those in warehouses) for accessing all stations, which may make their travels long and unnecessary, or expose the warehouses to disorder and insecurity.

The engineering team responsible for designing the network recommends limiting the scope of AGVs' motions to the production workshop, such that no AGV can enter the warehouses except for material loading or unloading, and therefore is only allowed to travel in the production area for resolving deadlocks with other AGVs. In this case, the maximum number of agents able to move will be $m = 15 - 3 - 1 = 11$. Since this number is less that the required number of AGVs the management has planned, either of the vertices $v_{17}$ or $v_{26}$, which are the nearest vertices to the production area, is selected to be included in the operational network (refer to Fig. 15 for vertex numbers). This resolution will not affect the size of the network's Stem, and so the number of operational AGVs increases to $m = 16 - 3 - 1 = 12$, which will make the network compatible with the requirements of the management.

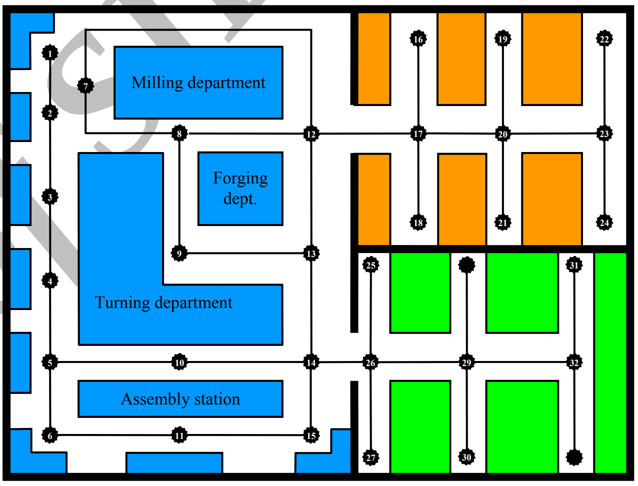


Fig. 15. A simplified AGV transportation network for the factory.

For Rightsizing the network, deleting edges $v_{12}$-$v_8$ and $v_{12}$-$v_7$ is proposed, regarding their lengths. Also, if there would be the possibility of creating an edge between $v_2$ and $v_7$, then there would be no need to utilize vertices $v_{17}$ or $v_{26}$ as recommended earlier.

For increasing the production volume and variety, the number of AGVs is planned to increase to 18. For this phase two possible scenarios are:

(1) Utilizing 7 additional vertices located in the warehouse areas, such as $v_{16}$, $v_{17}$, $v_{18}$, $v_{19}$, $v_{20}$, $v_{21}$ and $v_{23}$. In this case the maximum number of AGVs will be $m = 22 - 3 - 1 = 18$.

(2) Creating an edge between vertices $v_2$ and $v_7$ and utilizing 4 additional vertices located in the warehouse areas (i.e. $v_{17}$, $v_{20}$, $v_{26}$, $v_{29}$). In this case the maximum number of AGVs will be $m = 19 - 0 - 1 = 18$, which will enable us to delete the edges $v_{12}$-$v_8$ and $v_{12}$-$v_7$ in the Rightsizing phase.

Each of the above solutions should undergo financial and technical feasibility studies to be selected and finalized.

## 8. Discussion and Conclusion

The time complexity of the proposed method for verifying the solvability of a graph is in O(n2) 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 $p$ different permutations of agents to be checked as calculated in (5), which is far beyond the time order of the presented algorithm.

$$p = \frac{n!}{(n-m)!} \tag{5}$$

Moreover, verifying whether a graph is SG$^m$ would require $q$ operations to be checked as calculated in (6), 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.

$$q = \sum_{i=1}^{m} \left( \frac{n!}{(n-i)!} \right)^2 \tag{6}$$

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. Also a new algorithm is proposed for making an existing graph solvable and lean for a given number of agents.

In this paper we supposed that the agents may move concurrently on cycles and sequentially on internal vertices and leaves of the graph. For further research it is possible to assume all moves to be concurrent and find the minimum sequence of moves to reach the final configuration with different edge lengths and agents velocities. Also, further research is underway for working out solvability conditions for Purely Cyclic Graphs and Partially Solvable Graphs.

## References

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

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

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

[4] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, S. Thrun, Principles of Robot Motion: Theory, Algorithms, and Implementations, MIT Press, Boston, 2005.

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

[6] B. C herkassky, A.V. Goldberg, Negative-cycle detection algorithms, Mathematical Programming, Vol. 85(2), 277-311, 1999.

[7] Y.K. Hwang, N. Ahuja, Gross motion planning - a survey, ACM Comp. Surveys, Vol. 24(3), 219-291, 1992.

[8] D. Kornhauser, G. Miller, P. Spirakis, Coordinating pebble motion on graphs, the diameter of permutations groups and applications,proceding of 25th IEEE Symp. On Foundations of Comp. Science, 241-250, 1984.

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

[10] E. Masehian, A.H. Nejad, Multi robot motion planning on trees; Part I: feasibility of motions, submitted to IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), St. Louis, USA, 2009.

[11] E. Masehian and A.H. Nejad, Multi robot motion planning on trees; Part II: minimal agent motions, submitted to IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), St. Louis, USA, 2009.

[12] S. Onn, M. Tennenholtz, Determination of social laws for agent mobilization, Artificial Intelligence, Vol. 95, 155-167, 1997.

[13] M.R.K. Ryan, Multi-robot path planning with subgraphs, proceding of 19th Australasian Conf. Rob. and Autom, Auckland, New Zealand, 2006.

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