Verification of Visibility-Based Properties on Multiple Moving Robots

Ali Narenji Sheshkalani(✉), Ramtin Khosravi, and Mayssam Mohammadi

School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran
{narenji,r.khosravi}@ut.ac.ir, mayssam.moh@gmail.com

Abstract. In a multi-robot system, a number of autonomous robots sense, communicate, and decide to move within a given domain to achieve a common goal. To prove such a system satisfies certain properties, one must either provide an analytical proof, or use an automated verification method. To enable the second approach, we propose a method to automatically generate a discrete state space of a given robot system. This allows using existing model checking tools and algorithms. We construct the state space of a number of robots, each arbitrarily moving along a certain path within a bounded polygonal area. This state space is then used to verify visibility properties (e.g., if the communication graph of the robots is connected) by means of model-checking tools. Using our method, there is no need to analytically prove that the properties are preserved with every change in the motion strategy of the robots. We have provided a theoretical upper bound on the complexity of the state space, and also implemented a tool to automatically generate the state space and verify some properties to demonstrate the applicability of our method in various environments.

Keywords: Formal methods for robotics · Distributed robot systems · Verification

1 Introduction

Mobile robots are able to sense, communicate, and interact with the physical world, and are able to collaboratively solve problems in a wide range of applications. In many applications within the general area of robot motion planning, visibility problems play an important role.

There has been a close relationship between robot motion planning and computational geometry in the applications where the robots are constrained to move within a geometric domain. Traditionally, there has been a research area with the goal of minimizing the number of (stationary) guards or surveillance cameras to guard an area in the shape of a certain geometric domain like extensions of art-gallery problems [21]. Moving to the area of mobile guards, Durocher et al. [7] considered the sliding cameras problem in which the cameras travel back and forth along an axis-aligned line segments inside an orthogonal polygon. In the Minimum Sliding Cameras (MSC) problem, the objective is to guard the polygon

with the minimum number of sliding cameras. In MSC problem, it is assumed that the polygon is covered by the cameras if the union of the visibility polygons of the axis-aligned segments equals the polygon. One of the original works on the subject of mobile guards is studied by Efrat et al. [8] considering the problem of sweeping simple polygons with a chain of guards. They developed an algorithm to compute the minimum number of guards needed to sweep a simple polygon.

Traditionally, the correctness of robot motion planning algorithms within the context of computational geometry such as the ones mentioned above is investigated by manual proofs. It may be hard for certain types of planning algorithms to prove they correctly satisfy the problem's constraints (such as connectivity among the robots or covering of the whole area). On the other hand, when it comes to practical applications of motion planning algorithms, the designer may heuristically tune the algorithm's parameters or even the whole strategy in order to find the best solution that fits both the problem constraints and practical restrictions. In these cases, manually proving the algorithm with every change may be impractical.

An alternative and more reliable approach to examine the correctness of the planning algorithms is formal verification, and more specifically, model-checking [5] which has become more popular in recent years. Here, a mathematical model of all possible behaviors of the system is constructed, often as a state transition system, and is automatically verified against the desired correctness properties over all possible paths. The properties are often expressed in temporal logic formulas.

In a few existing works, model checking has been employed to verify motion planning algorithms. In [10], the authors used a discrete representation of the continuous space of the movement of a single robot, producing a finite state transition system. Later, [9] extended the previous framework to multiple robots. These frameworks generate a motion plan for the robot to meet some regions of interest inside a polygon in order to satisfy a given Linear Temporal Logic (LTL) [22] formula.

Another related area to which model-checking techniques have been applied are robot swarms. In [18] a swarm of foraging robots is presented and in [17] is analyzed using the probabilistic model-checker PRISM [15]. A hierarchical framework for model-checking of planning and controlling robot swarms is suggested in [16] to make some abstraction of the problem including the location of the individual robots. Dixon et al. [6] used model-checking techniques to check whether temporal properties are satisfied in order to analyze emergent behaviors of robotic swarms. Moreover, [4] presented property-driven design, a novel top-down design method for robot swarms based on prescriptive modeling and model checking. In 2014, Guo and Dimarogonas [13] proposed a knowledge transfer scheme for cooperative motion planning of multi-agent systems. They assumed that the workspace is partially known by the agents where the agents have independently-assigned local tasks, specified as LTL formulas.

More recently, Sheshkalani et al. [25] focused on the verification of certain properties on a multi-robot system where each robot was programmed with an

arbitrary navigation algorithm. The robots were assumed to move along the boundaries of a given polygon. They constructed a transition system on which the visibility properties can be investigated.

We believe that the result presented in [25] is restrictive in the sense that the robots are only allowed to move along the boundary of the environment. On the other hand, allowing the robots to freely move inside the polygon causes the state space to grow considerably. To remedy these problems, we define a generalized version of the problem of [25] by assuming that each robot is able to move freely along a simple path inside the environment. In addition to making the problem much more general, we have improved the result of the mentioned paper in terms of state space complexity.

As an application of the problem studied in this paper, the problem of guarding a bounded environment with a number of sliding cameras can be viewed as a special case of our problem. This way, our method is related to the previous study of [7]. Note that the mentioned study address the combinatorial optimization problem of minimizing the number of cameras. On the other hand, we address the problem of verifying correctness of the motion strategies for the given system. Another, more interesting, application of the problem is to consider the connectivity preserving (global connectivity maintenance) of the communication graph. Sabattini et al. [23] proposed a method to preserve the strong connectivity by estimating the algebraic connectivity of the communication graph in a decentralized manner. This way, our method can be used to guarantee the correctness of the desired requirements related to the *Connectivity* property.

The inputs to our method are comprised of (1) the environment, in the form of a simple polygon, (2) the algorithms controlling the motions of the robots, (3) the paths on which the robots are allowed to move, along with their initial positions, and (4) the requirement, expressed as a LTL formula. The output of the method is a True/False answer to the desired requirement as well as a transition system, labeled by two visibility-related atomic proposition: *Connectivity* (the communication graph of the robots is connected) and *Coverage* (the robots can collectively see the entire environment). The generated transition system is used to model check the visibility properties expressed in temporal logic formulas over the mentioned atomic propositions. The problem is defined more elaborately in Sect. 2.

We define a notion of state for such a system to construct a transition system on which the properties can be verified using the conventional model-checking algorithms (Sect. 3). Our method is abstract from specific motion planning algorithms in the sense that each robot may be programmed with a separate algorithm which during execution may cause the robot to sense the surroundings through various sensors or perform communications with other robots. In the end, all the sensing, communication, and internal logic leads to (possibly several) movement steps which is treated as actions by our method, causing transitions between states. Additionally, we provide a theoretical upper bound on the complexity of the state space (Sect. 4). Finally, we have implemented a tool to automatically generate the state space and verify the correctness of some

sample requirements using CADP [11] tool to demonstrate the applicability of our method in various environments (Sect. 5).

2 Preliminaries and Problem Definition

The following definitions are borrowed from [12]. A simple polygon P is defined as a closed region in the plane bounded by a finite set of line segments (called edges of P) such that there exists a path between any two points inside P which intersects no edge of P. Each endpoint of an edge of P is called a vertex of P. A vertex of P is called *convex* if the interior angle at the vertex formed by two edges of that vertex is at most 180°; otherwise it is called *reflex*.

Definition 1 (Visibility [12]). *Two points p and q in P are said to be* visible *if the line segment joining p and q contains no point on the exterior of P. This means that the segment pq lies totally inside P. This definition allows the segment pq to pass through a reflex vertex or graze along a polygonal edge. We also say that p sees q if p and q are visible in P. It is obvious that if p sees q, q also sees p.*

For a simple polygon P, we use the notation V_p for the visibility polygon of a point $p \in P$. Removing V_p from P may result in a number of disconnected regions called *invisible regions*. Any invisible region has exactly one edge in common with V_p, called a *window* of p, which is characterized by a reflex vertex of P visible from p, like p'. The window is defined as the extension of the (directed) segment pp' from p' to the boundary of P say p''. We denote such a window which consists of two endpoints p' and p'' by $w(p, p')$ (Fig. 1).

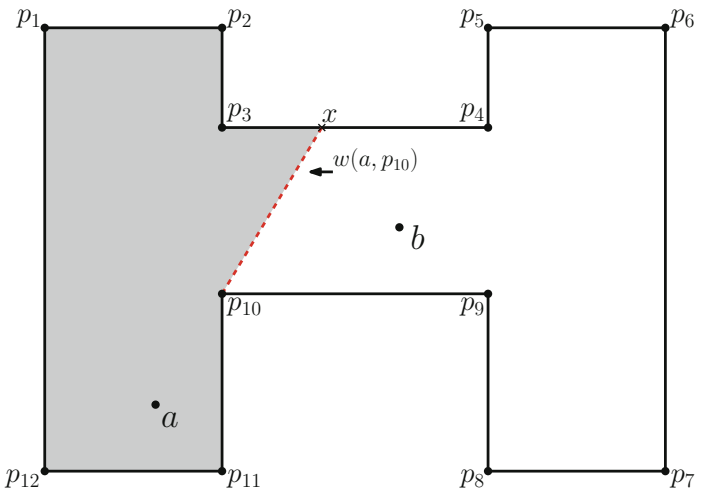

Fig. 1. The shaded area indicates the visibility polygon of point a (V_a). Point b is invisible from a, and its containing invisible region is characterized by the reflex vertex p_{10} which is separated from V_a by segment $p_{10}x$.

Consider a simple polygon P whose boundary is specified by the sequence of n vertices $<p_1, p_2, \ldots, p_n>$ including the set of reflex vertices P_{ref} and convex vertices P_{conv}, a set of robots $R = \{r_1, r_2, \ldots, r_k\}$, the corresponding navigation algorithms $Alg = \{a_1, a_2, \ldots, a_k\}$ (a_i is the navigation algorithm of robot r_i), and the corresponding paths $Paths = \{path_1, path_2, \ldots, path_k\}$ inside P are given with the following properties:

1. Robot r_i can only move along its corresponding path $path_i$,
2. Each step in the movement of each robot is specified by two parameters: direction (one of the two directions) and distance (a real positive number).

To discretize the state space of the system, we assume that the robots have turn taking movements as described in [1,6]. It means that during the movement of a robot, the position of other robots is fixed. Since our method is abstract from specific motion planning algorithms (each algorithm in set Alg is seen as a black-box) and the algorithms may be non-terminating (there might be no end goal), there is no way to determine if the state space has been constructed completely (especially for the cases in which the robots have time-sensitive behavior, e.g., take an action at a certain point in time). Hence, we take a time-bounded approach to state-space generation, and let the modeler determine how long the search for new states must take (using her knowledge of the navigation algorithms). As can be seen from our experimental results stated in Sect. 5, the growth rate of the number of generated states decreases significantly as time goes by, because in our case study the robots have a specific common goal which prevents the robots from making arbitrary actions.

The correctness properties may be built using temporal logics which are formalisms to describe temporal properties of reactive systems [2]. Apart from the logical operators, temporal logic formulas are constructed over a set of atomic propositions which may be true or false in each state of the system. Since our goal is to verify visibility properties, we need to define the two following properties:

Definition 2 (Connectivity). *The set of robots are connected if the graph induced by the visibility relation between pairs of robots is connected.*

Definition 3 (Coverage). *The robots cover P if the union of the visibility polygons of all robots ($\bigcup_{r \in R} V_r$) covers the whole P.*

Since we do not deal with the details of model-checking algorithms directly in this paper, we refer the reader for a detailed description of temporal logics to [2]. However, to bring an example, the LTL formula $\Box((Connectivity \wedge \neg Coverage) \rightarrow \Diamond(Connectivity \wedge Coverage))$ describes the property that whenever (represented by $\Box$) the visibility graph of robots is connected but the environment is not covered, eventually (represented by $\Diamond$) the system reaches a state in which both $Connectivity$ and $Coverage$ properties are satisfied (robots will eventually cover the environment by collaborating with each other).

We define a robot system RS as the tuple $(P, R, Alg, Paths, init)$ in which P indicates the environment of robots to navigate, R is the set of moving robots,

Alg is the set of navigation algorithms of robots, *Paths* is the position of paths in which the corresponding robots can move along them, and *init* specifies the initial position of robots over *Paths*. Our goal is to define the transition system equivalent of *RS*, over which temporal logic formulas may be model-checked.

To simplify our presentation of the method, we assume that the paths used throughout the examples in this paper are line segments.

3 Constructing the Discrete State Space

With the ultimate goal of verifying a temporal logic formula over a robot system $RS = (P, R, Alg, Paths, init)$, we must first construct the equivalent transition system of *RS*. As mentioned before, the states are labeled with the atomic propositions, hence, the transition system is called a Labeled Transition System (LTS) [2].

We define the LTS of *RS* as the tuple $(S, Act, \hookrightarrow, s_0, AP, L)$ where

- S is the set of states (defined below),
- $Act = \{\overleftarrow{move}_{r_i}, \overrightarrow{move}_{r_i} | 1 \leq i \leq k\}$ is the set of actions denoting the movement of robot r_i in its two possible directions,
- $\hookrightarrow \subseteq S \times Act \times S$ is the transition relation, (we use the notation $s \overset{\alpha}{\hookrightarrow} s'$ whenever $(s, \alpha, s') \in \hookrightarrow$),
- $s_0 \in S$ is the initial state (determined based on *init*),
- $AP = \{Connectivity, Coverage\}$ is the set of atomic propositions,
- $L : S \rightarrow 2^{AP}$ is the labeling function.

3.1 System States

The satisfaction of *AP* depends on the distribution of robots' position over *Paths*. We model each state of the system based on the topology of the robots and vertices of *P*. Additionally, we may need to store some extra information in order to identify the next states.

Consider the union of all the windows of the robots $W = \{w(p, p') | p \in R, p' \in P_{ref}, p' \in V_p\}$. The intersection of the line segments in *W* results in a subdivision inside *P* which is denoted by Sub_P (Fig. 2).

Definition 4 (Dual graph). *Let Sub_P be a subdivision of P. The dual graph of Sub_P that is represented by $DG(Sub_P)$ is a graph which has a node corresponding to each cell, and each pair of nodes are connected with an edge, if their related cells have an edge in common [12].*

Each node of $DG(Sub_P)$ is determined by the windows and the polygon edges which define the boundary of the corresponding cell in Sub_P. In Fig. 2, the corresponding node of cell c_i in $DG(Sub_P)$ is associated with the set of edges $\{w(d, p_{10}), w(b, p_4), (p_9, p_{10})\}$. $DG(Sub_P)$ does not change unless some cells are removed from or added to Sub_P. Therefore, we may use the dual graph of *P* to represent Sub_P. Since the satisfaction of *AP* can be determined by considering

Sub_P, we can store $DG(Sub_P)$ as a part of each state. Suppose robot r_i moves in one of its two possible directions (actions $\overleftarrow{move}_{r_i}$ or $\overrightarrow{move}_{r_i}$). The corresponding windows of r_i ($W_{r_i} = \{w(r_i, p') | r_i \in R, p' \in P_{ref}, p' \in V_{r_i}\}$) may move radially around p' during the movement of r_i respectively. Line segments W_{r_i} may construct new cells or destruct existing cells of Sub_P during the movement. Construction or destruction of cells may happen if and only if some line segments in W_{r_i} intersect some vertices of Sub_P. As mentioned before, we may need to store some extra information to correctly determine the next states to be encountered in the future as each robot moves. Let $Seq_{\overleftarrow{move}_{r_i}}$ indicates the sequence of intersection points of W_{r_i} with vertices of Sub_P and the robots during the movement of r_i in $\leftarrow$ direction (the same definition for $Seq_{\overrightarrow{move}_{r_i}}$ can hold as well - e.g., $Seq_{\overrightarrow{move}_d} = <i_3, p_4, i_7>$ in Fig. 2). Storing $Seq_{\overleftarrow{move}_{r_i}}$ and $Seq_{\overrightarrow{move}_{r_i}}$ in states enables efficient computation of the successor states regarding the transition types described in the next section.

We define a state of k robots inside the polygon P as:

1. $DG(Sub_P)$ along with the robots each cell of Sub_P contains,
2. $Seq_{\overleftarrow{move}_{r_i}}$ for all $1 \leq i \leq k$,
3. $Seq_{\overrightarrow{move}_{r_i}}$ for all $1 \leq i \leq k$.

By the definition of LTS, we assume each atomic proposition is either true or false in a state. The following lemma states that moving of the robots does not change the validity of the propositions *Connectivity* and *Coverage*, as long as the state defined above remains the same.

Lemma 1. *Each state s can be uniquely labeled with the atomic propositions $AP = \{Connectivity, Coverage\}$.*

Proof. Assume that the labeling $L(s) \in 2^{AP}$ is satisfied by the current state s. It is sufficient to prove that by moving the robots, $L(s)$ is valid while s does not change. We discuss the two atomic propositions separately.

Connectivity. Two robots r_i and r_j are connected, if one lies in the visible area of the other ($r_i \in V_{r_j}$). Since the boundaries of visible areas for each robot are determined by its corresponding windows (W_{r_j}) which are stored as the line segments in Sub_P, we can decide whether robot r_i locates inside V_{r_j}, by inspecting $DG(Sub_P)$. Assume that robots r_i and r_j are connected, and they are located in cells c_i and c_j respectively (based on Sub_P). If r_i moves in order to get disconnected, it must cross one of the line segments in W_{r_j}. In this case, r_j does not belong to c_j anymore. So, the current state s changes based on the definition of state.

Coverage. Polygon P is covered if and only if all the cells in Sub_P are covered by the robots. Assume that there exists at least one cell say c_i which is visible from none of the k robots (Fig. 2). The polygon remains uncovered as long as c_i is not destructed. More precisely, the polygon may get covered if the uncovered cells destructed. On the other side, assume that all the cells of Sub_P are covered by the robots. In order to make P uncovered, it is needed a new cell which is not

visible from the robots to be constructed in Sub_P. Since any changes in validity of *Coverage* needs to make Sub_P different from its previous structure, *Coverage* is valid while s does not change. □

3.2 Transitions Events

A movement step of robot r_i is specified by the pair $move_{r_i} = (dir, dist)$ where:

- $dir \in \{\leftarrow, \rightarrow\}$ is the direction of the movement along $path_i$,
- $dist$ indicates the length of the movement.

We define $\overleftarrow{move}_{r_i}$ as the tuple $(\leftarrow, \delta)$, where δ is the smallest distance robot r_i can move in that direction along $path_i$ which causes a change in state. Also, we define $\overrightarrow{move}_{r_i}$ for the other direction similarly. We illustrate $\hookrightarrow$ as the smallest relation containing the tuples (s, α, s'), where $s, s' \in S$, $\alpha \in \bigcup_{1 \leq i \leq k} \{\overleftarrow{move}_{r_i}, \overrightarrow{move}_{r_i}\}$, and s' is the state obtained from s by taking action α. While r_i is making its movement, a transition $s \overset{\alpha}{\hookrightarrow} s'$ can occur in the following transition types:

(a) Some cells constructed or destructed in Sub_P which leads to changes in $DG(Sub_P)$,
(b) A robot crosses a window, and moves into another cell of Sub_P,
(c) If none of the two above types have occurred after the movement of the robot, it must be checked whether the order of points in $Seq_{\overleftarrow{move}_{r_j}}$ or $Seq_{\overrightarrow{move}_{r_j}}$ for some $1 \leq j \neq i \leq k$ has changed. If that is the case, we need to have a transition to s' with the same $DG(Sub_P)$ as of s, but having $Seq_{\overleftarrow{move}_{r_i}}$ and $Seq_{\overrightarrow{move}_{r_i}}$ updated for all $1 \leq j \neq i \leq k$.

As an example, consider Fig. 2 (both *Connectivity* and *Coverage* properties are not satisfied). Assume that robot b moves to the right. First, it destructs cells c_i, c_x and c_y, and constructs two new cells c_m and c_n (transition type (a)) before reaching $w(d, p_4)$ (Fig. 3). The *Coverage* property is satisfied in the generated state. Second, it constructs three other new cells c_j, c_k and c_l (Fig. 4). The validity of the properties are preserved in the generated state. Finally, it reaches $w(d, p_4)$, and makes robots b and d visible to each other (transition type (b)) which satisfies the *Connectivity* property as well (Fig. 5).

We may use the plane sweep algorithm [24] in order to find out whenever r_i reaches an intersecting point in Sub_P for type (a). More precisely, radial sweep algorithm [19] may be used to rotate $w(r_i, p')$ about p' in order to discover the intersection points of Sub_P. So, robot r_i may move towards its two possible directions ($\leftarrow$ or $\rightarrow$) until it reaches the end of c_i in order to compute $Seq_{\overleftarrow{move}_{r_i}}$ and $Seq_{\overrightarrow{move}_{r_i}}$ for type (c). Since AP may change only in transition types (a) or (b) based on Lemma 1, the validity of AP remains the same in type (c). Assume that robot r_i moves from its current position pos_i to a new position pos'_i, and a transition from s to s' occurred, in such a way that type (c) happened. Since none of the transition types (a) and (b) has happened, the dual graph of Sub_P and the

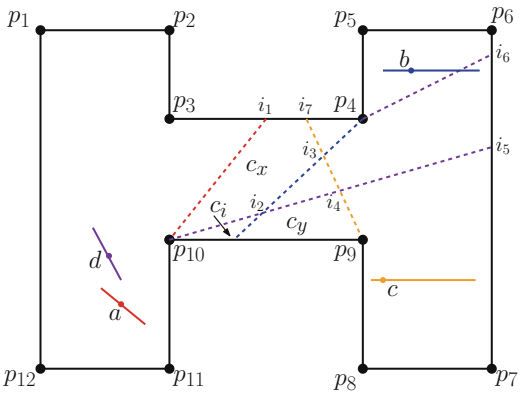

Fig. 2. A subdivision which consists of the intersection of line segments in W inside P.

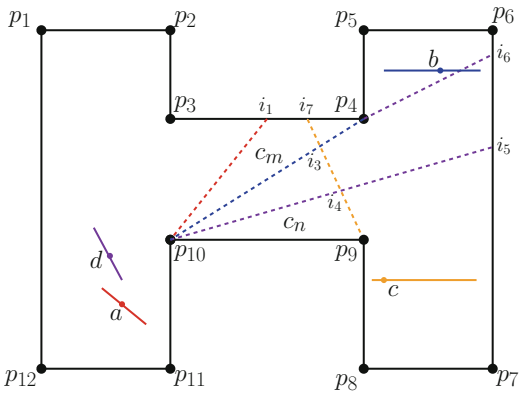

Fig. 3. The subdivision changes after moving b to the right before reaching $w(d, p_4)$. The *Covering* property is satisfied.

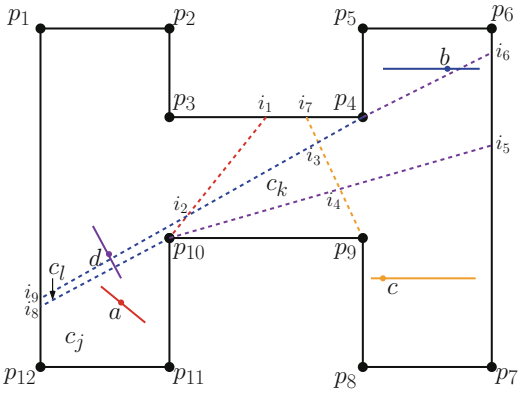

Fig. 4. Construction of two new cells c_k and c_l during the movement of b to the right before reaching $w(d, p_4)$.

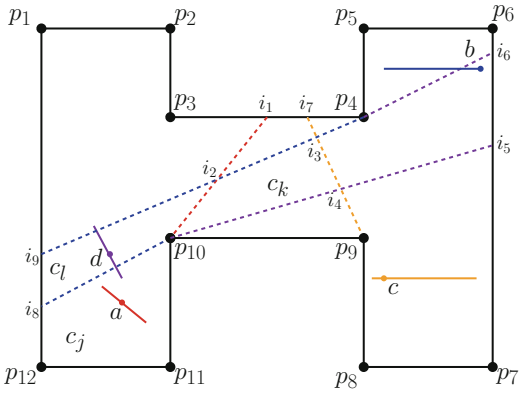

Fig. 5. The *Connectivity* property is satisfied. Transition type (c) happens after taking the action of robot b.

cells which the robots belong to remain the same as in s. It means that $Seq_{\overleftarrow{move}_{r_j}}$ or $Seq_{\overrightarrow{move}_{r_j}}$ for some $1 \leq j \neq i \leq k$ are changed during the movement of r_i. Precisely, the sequences in $Seq_{\overleftarrow{move}_{r_l}}$ or $Seq_{\overrightarrow{move}_{r_l}}$ for some $1 \leq l \leq k$ may change during the movement of r_i before reaching pos'_i, but the corresponding states are not generated. Since AP may change only in transition types (a) or (b), the states which are not generated during the movement have the same labels as in s. In the previous example, the obtained dual graph when robots b and d get visible to each other does not change while robot b reaches the right endpoint of the corresponding path (Fig. 5). On the other hand, $Seq_{\overrightarrow{move}_c}$ changes from $<i_1, p_3>$ to $<i_1, p_3, i_2>$ during the movement of b (after crossing $w(d, p_4)$) till reaching the right endpoint of its path which does not make any transitions of types (a) or (b). After taking the action of robot b, a state is generated (transition type (c)) with the same $DG(Sub_P)$ but different $Seq_{\overrightarrow{move}_c}$. Preventing the construction

of type (c) transitions (during the movement) leads us to achieve a significant reduction in the size of the state space.

4 Analysis

In Sect. 3, the method of constructing an LTS on a given robot system $RS = (P, R, Alg, Paths, init)$ is described. In this section, we prove that the definition of the states and the transition events are consistent. Next, we discuss the state space complexity of our method.

4.1 Proof of Correctness

The following lemma states that for each state, the set of next sates can be uniquely determined independent of the exact position of the robots as long as the current state does not change.

Lemma 2. *Let $A = (S, Act, \hookrightarrow, s_0, AP, L)$ be the LTS of robot system $RS = (P, R, Alg, Paths, init)$. For each state $s \in S$, the set of next sates can be uniquely determined independent of the exact position of the robots as long as the configuration of the system is identical to state s.*

Proof. The transition events may only occur in the three types as explained in Sect. 3.2. Assume that robot r_i wants to move, and the current state of the system is s. The set of next states which belong to transition types (a) or (b) can be uniquely determined:

1. Robot r_i crosses one of the boundary line segments of its cell (type (b)): since the boundary line segments of each cell are unique for each state, the corresponding next states are unique respectively.
2. A window belonging to W_{r_i} crosses an intersection point of Sub_P (type (a)): the sequence of intersection points crossed by a window of W_{r_i} may vary while the corresponding Sub_P does not change. Since the sequence of intersection points ($Seq_{\overleftarrow{move}_{r_i}}$ or $Seq_{\overrightarrow{move}_{r_i}}$) are stored as a part of state s, the first intersection points of the two possible directions ($\leftarrow$ or $\rightarrow$) are unique respectively. It is important to note that the windows which are constructed or destructed during the movement of the robot are taken into account in order to compute $Seq_{\overleftarrow{move}_{r_i}}$ and $Seq_{\overrightarrow{move}_{r_i}}$, as well.

As mentioned in Sect. 3.2, the next state (s') which belongs to type (c) may be constructed at the end of the movement of r_i (none of the transition types (a) or (b) has happened during the movement). It means that there may exist a chain of intermediate states between s and s', but only s' is constructed. All of the intermediate states may be reached if the distance parameter in $move_{r_i} = (dir, dist)$ gets smaller. Since the proximity of all the windows to each other can be determined by $Seq_{\overleftarrow{move}_{r_i}}$ and $Seq_{\overrightarrow{move}_{r_i}}$ for all $1 \leq i \leq k$, the sequence of the intermediate states from s to s' are unique. Hence, the set of next states of s can be uniquely determined as long as the configuration of the system is equal to state s. □

4.2 Complexity

Lemma 3 states an upper bound on the maximum number of states for the robot system RS. The upper bound obtained in the lemma is not tight. In other words, the geometrical properties of the polygon, and therefore the resulting position of windows highly affect the size of the state space.

Lemma 3. *The maximum number of states in order to verify the given robot system* $RS = (P, R, Alg, Paths, init)$ *has the complexity of* $O(n^{k^3})$ *in which* n *and* k *denote the number of vertices of* P *and the number of robots.*

Proof. Consider a simple polygon P with n vertices and k robots inside. In order to compute the complexity of the state space, it is essential to obtain an upper bound on the maximum number of different subdivisions ($DG(Sub_P)$) shown as $C(DG(Sub_P))$ as well as the maximum number of different sequences of the robots ($Seq_{\overleftarrow{move}_{r_i}}$ and $Seq_{\overrightarrow{move}_{r_i}}$) shown as $C(Seq)$. Consider the current state s_i with the sequences $Seq_{\overleftarrow{move}_{r_i}}$ and $Seq_{\overrightarrow{move}_{r_i}}$ for $1 \leq i \leq k$ and $DG(Sub_P)$. As stated in Sect. 3.2 about type (c) transitions, there may exist more than one state with the same $DG(Sub_P)$ as in s_i, but with different sequences. So, each $DG(Sub_P)$ may correspond to more than one group of sequences belonging to the k robots ($C(DG(Sub_P)) \leq C(Seq)$). This way, it suffices to enumerate the number of different sequences for all the k robots as the parts of a state to obtain the maximum number of different states.

Consider a line segment in Sub_P which corresponds to $w(r_i, p')$. Line segment $w(r_i, p')$ may intersect some windows of the set W_{r_j} for some $1 \leq j \neq i \leq k$. In the worst-case scenario, $w(r_i, p')$ may intersect some windows of all $k-1$ robots. Since the polygon is simple (it has no hole inside), at most two windows of W_{r_j} may intersect $w(r_i, p')$ simultaneously [3]. Therefore, the window $w(r_i, p')$ may intersect at most $2(k-1) \in O(k)$ other windows at the same time. Based on the above discussion, the sequence say $Seq_{\overleftarrow{move}_{r_i}}$ may have at most $O(k^2)$ members which specify that robot r_i meets which intersection points of Sub_P during the movement to the left. Since the number of reflex vertices $|P_{ref}| \in O(n)$, each member of $Seq_{\overleftarrow{move}_{r_i}}$ may have $O(k^2n^2)$ options. Additionally, as the sequence has $O(k^2)$ members, we may have $O((k^2)!)$ permutations. Hence, there exist at most $O((k^2)!(k^2n^2)^{k^2})$ different sequences for $Seq_{\overleftarrow{move}_{r_i}}$.

Taking the sequences of all the k robots into account, we obtain complexity $O(((k^2)!(k^2n^2)^{k^2})^k) \in O(n^{k^3})$ as an upper bound on the maximum number of different sequences ($C(Seq)$) which is an upper bound on the maximum number of states as well (assuming $k \leq n$). □

Lemma 3 proves that the complexity of the state space is polynomial in terms of the complexity of the environment. Although the maximum number of states grows exponentially as the number of robots increases, in many applications like the one presented in Sect. 5, there may exist a global goal, so the robots avoid making arbitrary actions. As a comparison with the previous work of [25], they achieved the complexity of $O(n^{2^k})$ which is much greater than the complexity obtained by our method.

5 Experimental Results

We have used Computational Geometry Algorithms Library (CGAL) [14] to implement the proposed method in C++. The program automatically constructs the state space of the robot system $RS = (P, R, Alg, Paths, init)$ during the movement of the robots. Precisely, the states and the transitions are constructed with respect to the decisions made by the robots during their movements (determined by the motion algorithms). The implementation is available online via http://ramtung.ir/visification-1.0.zip which contains the source code as well as a Debian-based package.

As a case study, we consider robot swarms algorithms in which the robots use only local wireless connectivity information to achieve swarm aggregation. Particularly, we use the simplest *Alpha* algorithm which is examined using simulations and real robot experiments in [6,20,26] as the navigation algorithms of the robots in this experiment. It is assumed that the initial position of the robots are on the middle of the corresponding paths. Our experimental environment is an Ubuntu 14.04 machine, Intel Pentium (AMD64) CPU 2.6 GHz with 4 GB RAM.

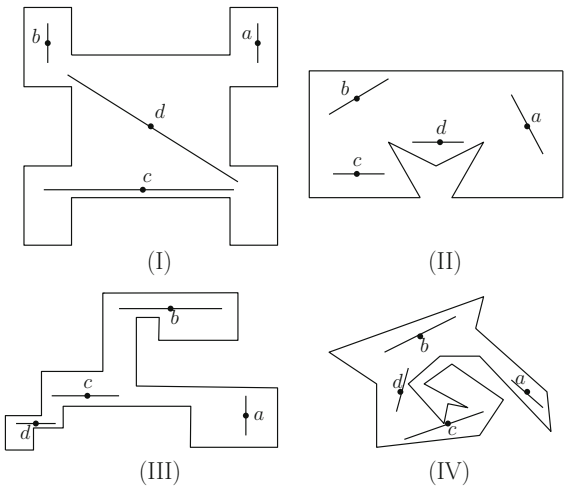

Fig. 6. The polygons used for the experimental results.

Figure 7 shows the growth rate of the size of the state space against the construction time for $k = 3$ (robots a, b and c) and $k = 4$ in different environments shown in Fig. 6 respectively. We executed the state space generation algorithm for 360 min ($timeBound = 360$). It shows that the number of investigated states converge quickly for all the polygons except for (I) when $k = 4$.

We used CADP [11] model-checker to verify the requirements (e.g., expressed in LTL formula) regarding the generated state space. Table 1 shows the results of the verification process after 360 min of running the state space generation algorithm with respect to the mentioned LTL formulas. The first formula $\Box\Diamond Connectivity$ is true, if the robots are infinitely often connected. The second formula $\Diamond(Connectivity \wedge Coverage)$ is true, if the system eventually reaches a

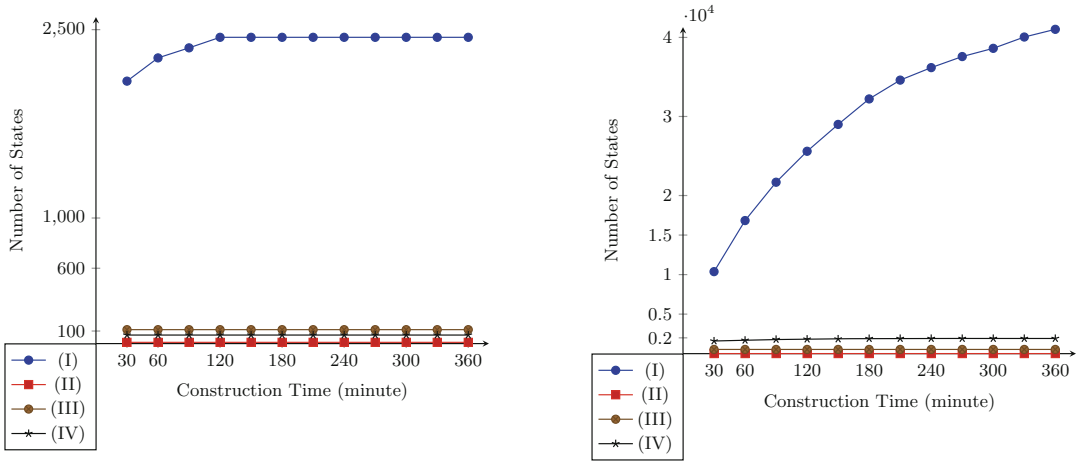

Fig. 7. The number of discovered states for $k = 3$ (left) and $k = 4$ (right).

state in which the communication graph is connected and the environment is fully covered by the robots which may be considered as a goal state. CADP evaluated each formula in less than two seconds for the polygons which shows the applicability of the proposed method. Since the robots are implementing the *Alpha* algorithm (which focuses on maintaining the connectivity) with $\alpha = 1$ (the decision in which the robot continues the previous direction or make a 180° turn depends on the value of α - number of visible robots), the robots in the polygons (III) and (IV) in Fig. 6 cannot reach a state in which the environment is covered. More precisely, consider Polygon (IV) with $k = 4$. Assume that robots c and d are visible to each other. Since robot c wants to keep the connection with d, it cannot cover some portion of the environment. If we increase the value of α by one ($\alpha = 2$), the number of visible robots for c (which is one) falls below the threshold. This way, based on the Alpha algorithm, robot c makes a 180° turn in order to avoid moving out the swarm. So, it may lead to cover the uncovered area.

As a comparison with a previous work, Dixon et al. [6] implemented the *Alpha* algorithm for three robots ($k = 3$) within grid sizes of 6×6 and 7×7, and

Table 1. The results of the verification of two LTL formulas.

LTL formula		$\Box\Diamond Connectivity$	$\Diamond(Connectivity \wedge Coverage)$
Polygon (I)	$k = 3$	True	False
	$k = 4$	True	False
Polygon (II)	$k = 3$	True	True
	$k = 4$	True	True
Polygon (III)	$k = 3$	False	False
	$k = 4$	False	False
Polygon (IV)	$k = 3$	False	True
	$k = 4$	False	False

obtained 168×10^6 and 501×10^6 number of states respectively. Even though they completely abstracted out the geometry of the environment, the number of states achieved are considerably greater than the number of states computed by our method which let the robots move continuously in a geometric domain.

6 Conclusion

We presented a method to construct a discrete state space for a multi-robot system and then verify the correctness properties by means of model-checking techniques. The notion of state has been defined in such a way that each state can be uniquely labeled with the atomic propositions *Connectivity* and *Coverage*. An important aspect of our method is that it treats the navigation algorithms as black-boxes. Iteratively searching for new states, at each step, our algorithm asks the black-box for its next action and creates the states caused by the action based on a precise definition of transitions. Using our provided implementation, the modeler can code the navigation algorithms and generate the state space. The generated state space is used to verify temporal formulas constructed over the mentioned propositions using CADP tool. An important benefit of this approach is to eliminate the need for analytical proof of correctness upon changes to the navigation algorithms.

From a geometric perspective, our method can be easily applied to more complicated cases, e.g., when the robots can move along possibly non-simplified paths (e.g., paths containing points in which a robot has more than two directions to choose). Furthermore, the geometric domain of simple polygons can be extended to polygonal domains, i.e., simple polygons having a number of holes inside. This makes our method applicable to more realistic situations.

References

1. Antuña, L., Araiza-Illan, D., Campos, S., Eder, K.: Symmetry reduction enables model checking of more complex emergent behaviours of swarm navigation algorithms. In: Proceedings of 16th Annual Conference on Towards Autonomous Robotic Systems, pp. 26–37 (2015)
2. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
3. Bose, P., Lubiw, A., Munro, J.I.: Efficient visibility queries in simple polygons. Comput. Geom. Theory Appl. **23**(3), 313–335 (2002)
4. Brambilla, M., Brutschy, A., Dorigo, M., Birattari, M.: Property-driven design for robot swarms: a design method based on prescriptive modeling and model checking. ACM Trans. Auton. Adapt. Syst. **9**(4), 17 (2014)
5. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
6. Dixon, C., Winfield, A.F., Fisher, M., Zeng, C.: Towards temporal verification of swarm robotic systems. Rob. Auton. Syst. **60**(11), 1429–1441 (2012)
7. Durocher, S., Filtser, O., Fraser, R., Mehrabi, A.D., Mehrabi, S.: A (7/2)-approximation algorithm for guarding orthogonal art galleries with sliding cameras. In: Pardo, A., Viola, A. (eds.) LATIN 2014. LNCS, vol. 8392, pp. 294–305. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54423-1_26

8. Efrat, A., Leonidas, J.G., Har-Peled, S., Lin, D.C., Mitchell, J.S.B., Murali, T.M.: Sweeping simple polygons with a chain of guards. In: SODA 2000, pp. 927–936 (2000)
9. Fainekos, G.E., Girard, A., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for dynamic robots. Automatica **45**(2), 343–352 (2009)
10. Fainekos, G.E., Kress-Gazit, H., Pappas, G.: Temporal logic motion planning for mobile robots. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 2020–2025 (2005)
11. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2010: a toolbox for the construction and analysis of distributed processes. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 372–387. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19835-9_33
12. Ghosh, S.K.: Visibility Algorithms in the Plane. Cambridge University Press, Cambridge (2007)
13. Guo, M., Dimarogonas, D.: Distributed plan reconfiguration via knowledge transfer in multi-agent systems under local LTL specifications. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 4304–4309 (2014)
14. Hemmer, M., Huang, K., Bungiu, F., Xu, N.: 2D visibility computation. In: CGAL User and Reference Manual, 4.7 edn. CGAL Editorial Board (2015)
15. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: a tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006). doi:10.1007/11691372_29
16. Kloetzer, M., Belta, C.: Temporal logic planning and control of robotic swarms by hierarchical abstractions. IEEE Trans. Rob. **23**, 320–330 (2007)
17. Konur, S., Dixon, C., Fisher, M.: Analysis robot swarm behaviour via probabilistic model checking. Rob. Auton. Syst. **60**(2), 199–213 (2012)
18. Liu, W., Winfield, A.F.T.: Modeling and optimization of adaptive foraging in swarm robotic systems. Int. J. Rob. Res. **29**(14), 1743–1760 (2010)
19. Mirante, A., Weingarten, N.: The radial sweep algorithm for constructing triangulated irregular networks. IEEE Comput. Graph. Appl. **2**(3), 11–21 (1982)
20. Nembrini, J., Winfield, A., Melhuish, C.: Minimalist coherent swarming of wireless networked autonomous mobile robots. In: Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior on From Animals to Animats, pp. 373–382. ICSAB, MIT Press (2002)
21. O'rourke, J.: Art Gallery Theorems and Algorithms. Oxford University Press, Oxford (1987)
22. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE (1977)
23. Sabattini, L., Secchi, C., Chopra, N.: Decentralized estimation and control for preserving the strong connectivity of directed graphs. IEEE Trans. Cybern. **45**(10), 2273–2286 (2015)
24. Shamos, M.I., Hoey, D.: Geometric intersection problems. In: 17th Annual Symposium on Foundations of Computer Science, pp. 208–215. IEEE (1976)
25. Sheshkalani, A.N., Khosravi, R., Fallah, M.K.: Discretizing the state space of multiple moving robots to verify visibility properties. In: Proceedings of 16th Annual Conference on Towards Autonomous Robotic Systems, pp. 186–191 (2015)
26. Winfield, A.F., Liu, W., Nembrini, J., Martinoli, A.: Modelling a wireless connected swarm of mobile robots. Swarm Intelligence **2**(2–4), 241–266 (2008)