Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 50-1 (2017) 11156–11162

Decomposing the model-checking of mobile robotics actions on a grid

Rim SADDEM [*,**] **Olivier NAUD** [*]
Karen GODARY DEJEAN [**] **Didier CRESTANI** [**]

[*] *Irstea, UMR ITAP, 361 rue Jean-Franois Breton, BP 5095, F-34196 Montpellier , France, (e-mail: {rim.saddem@irstea.fr},{olivier.naud@irstea.fr})*
[**] *Laboratoire d'Informatique Robotique et Microlectronique de Montpellier (LIRMM), UMR 5506, Université de Montpellier, 161 rue Ada, 34 095 Montpellier Cedex 5, France, (e-mail: {godary,crestani}@lirmm.fr)*

Abstract: Mobile automated systems, such as robots or machinery for precision agriculture, may be designed to perform actions that vary in space according to information from sensors or to a mission map. To be reliable, the design process of such systems should involve the combined verification of spatial and dynamic properties. We consider here CTL model-checking of a mobile robot's behavior, using the UppAal Timed Automata verifier. We consider reachability properties including path finding. Space is modeled as a 2D grid and the mobile robot path is unknown a priori. In this case, the exhaustive state space exploration of model-checking leads to the generation of many possible movements. This exposes such model-checking to combinatorial issues depending on the grid size and the complexity of system dynamics. In this paper, we propose a decomposition methodology reducing the memory requirements for the verification task. The decomposition is twofold. The grid is decomposed in sub-grids and the model-checking query on the whole grid is decomposed in a set of queries on the sub-grids. A set of test cases and check the validity of the decomposition concept. The decomposition methodology is compared to a simpler method that verifies the reachability property without proceeding to decomposition.

Keywords: Formal verification, reachability property, models, decomposition methods, automata, robotics, unique action, precision agriculture, spatial process, spatial grid

1. INTRODUCTION & RELATED WORK

The development of mobile automated systems has seen a lot of progress in recent years. Such systems perform actions, make decisions and act in real time with limited human intervention or autonomously. For example self-driving cars are able to interact with their environment and navigate toward their destination without human intervention. Al-A'Abed et al. (2015) reports about the development of a prototype intelligent parking system for self-driving cars. A vehicle parking position is assigned and a path to it is computed with power consumption as optimization criterion. The ability to generate an efficient and sure path from a given initial point to a final destination is still a challenging task (Souissi et al. (2013)).

Research on autonomous equipment in agriculture started back in the early 1960, focusing at the time on the development of automatic steering systems (Bechar and Vigneault (2016)). In the last decade, an increasing number of research projects on agricultural robots were undertaken (Herrera et al. (2016), Kayacan et al. (2013)). AdAP2E (Lenain (2014)) is a research project which aims at developing an interactive and mechanically reconfigurable robotic demonstrator. The reconfigurable robot should be able to perform precise movements in agricultural and natural environments. One task in this project, in which authors of this paper are involved, is to supervise and verify the robot's missions. It can be illustrated in the following way. Let us suppose the robot has its usual parking facility at a specific location in a farm. It would have then to reach a field to perform an operation and move from a field to another. Quite often, there are many points from which the operation on the field can be started, and the space between two fields can be poorly structured so that there is no pre-determined route that the robot should follow. The robot moves in a partially known and dynamic environment. All actions (having an effect on environment or not) that the robot may do during its mission depend on its position, its internal state and its environment. Two objectives for this robotic mission design can then be formulated:

(1) verify *a priori* that the mission is compatible with spatial constraints and robot dynamics,
(2) generate a controller for the robot mission management.

We consider that formal methods such as model checking are appropriate for these design tasks because they provide guarantied results on a model of a system. Model checking differs from simulation in the sense that all the possibilities that are relevant to a model and the property to be verified

10.1016/j.ifacol.2017.08.1236

are explored in an exhaustive way. This exhaustiveness has its drawbacks: model checking often leads to combinatorial problems. In this paper, we deal with the *a priori* verification of a robotic mission (spatial and dynamic properties) using model checking techniques.

Some studies on multi-agent systems include spatial behavior of multiple robots (Claes et al. (2015), Leahy et al. (2015)). But, until now, the modelling and simulation tools of multi-agent systems do not provide functionalities for formal verification of the whole system. Cellular automata include spatial behavior and are widely used for simulation in agriculture (Vanwalleghem et al. (2010)). The classical paradigm is the synchronous evolution of a 2D grid, with a global state that is formed by logical values in each cell. There are also asynchronous interpretation of cellular automata such as in (Wainer and Giambiasi (2001)). Unfortunately, to our knowledge, very few studies were conducted on model-checking for the cellular automaton paradigm and no specific model-checking tool is available.

Timed Automata (Alur and Dill (1994)) have already been used in the domain of agriculture (Hélias et al. (2008); Largouët et al. (2012)). UppAal is a tool designed to validate systems that can be modeled as networks of Timed Automata. The language used in UppAal extends the formalism by adding integer variables, structured data types, user defined functions, and channel synchronisation (Bengtsson and Yi (2004)). UppAal includes the on-the-fly model checking feature, which can help to reduce memory requirements (Larsen et al. (1997, 2003)). However, the language used does not offer specific means for modelling spatial behavior. Modeling the spatial behavior as a movement on a 2D grid induces combinatorial explosion problems for grids of significant size during verification of reachability properties. With the verification of safety properties, which requires to explore all the state space, the combinatorial explosion problem should occur as well.

In the litterature, abstractions and modular verification techniques(Alur et al. (1999)), reduction methods (Clarke et al. (2004)) and decomposition methods (Koo and Mishra (2006)) have been developed to overcome the combinatorial explosion problem. These methods are not specifically based on the spatial distribution issue.

In order to model-check a robotic mission, an alternative approach could use path planning (survey of such methods in Souissi et al. (2013)) or route planning. Including constraints or properties to verify in an exact route planning approach, based on mixed-integer linear programming approach does require the design of custom exact algorithms each time constraints change. A survey of rich vehicle routing problems is given in (Caceres-Cruz et al. (2015)). Route planning might also be combined with model-checking: the planned route would be an input to the verifier. In the case that the desired property would not be satisfied on the planned route, alternative routes would have to be found. Such an iterative method might be error prone.

With the ultimate objective of checking both reachability and liveness or safety properties, and from the results of our survey, we concluded that the possibilities of model-checking tools, such as UppAal, to verify reachability problems linked to robots spatial behavior should be investigated. In order to make model-checking on such problems tractable, we propose in this paper a new decomposition method that is specifically based on the spatial distribution of the problem.

The outline of the paper is as follows. Section 2 describes the example problem, its model made with UppAal, and the reachability property to check. Section 3 introduces the decomposition methodology. An execution example is given. In Section 4, we study the behavior of decomposition algorithm for different scenarios and we compare it to a basic method which does not proceed to decomposition.

2. ILLUSTRATIVE EXAMPLE AND MODEL

In this section, we will explain the problem we intend to study using an example that was designed as an abstraction of some situations that autonomous robots would encounter in agriculture. We provide a model of this example problem and formulate the properties that are to be checked.

2.1 Problem under study

We consider a robot that moves on a regular grid of size width × length depicted in figure 1. Except on border lines, the robot may move on the adjacent cell to the east, the north, the south. Diagonal and west moves are not permitted.

The robot must perform one specific action for each cell it visits. A cell must be visited only once. Let A, B, and C be the actions to be performed by the robot in the field. The figure 1 describes an example of the distribution of actions to be done when a cell is visited.

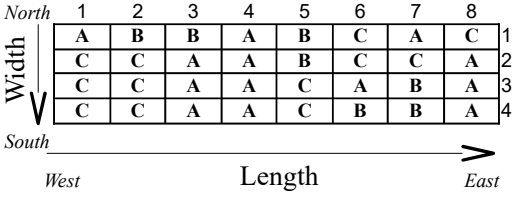

Fig. 1. Distribution of actions accross a grid

Robot internal dynamics are represented in our example model only by precedence constraints on actions. These constraints can be represented by an automaton (Figure 2). In this example, doing action A before action B is authorized but doing action A before action C is not.

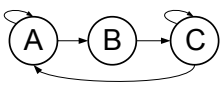

Fig. 2. Automaton of precedence constraints

Sample motion of the robot: Let us suppose that the robot is initially in the north-west corner cell in grid depicted Figure 1. It needs to perform action A. Actions B and C are assigned respectively to next cell to the east and next cell to the south. Action B can be done after A and

action C cannot (see Figure 2). It follows that the robot can only move to the east.

Next subsection provides a model for robot operations, which we call *Movement Based (MB) Model*.

2.2 MB Model

In the following, each cell of a grid is represented by a pair (x,y) where x is the line index, starting north, and y is the column index, starting west. The current position of the robot in the model is denoted (i, j), which is initialised at (i_{init}, j_{init}). The MB model is composed of two automata: a control automaton and a movement automaton.

Control automaton (Figure 3): At init time, the starting cell for the robot is selected. There are three similar parts in the automaton that correspond to each potential movement. Let us examine movements to the east. In the state *"Position"*, the controller checks if, from the current position, and according to movement constraints, the robot may move to the east. If so, the control state may change to *"Seek Neighbor"*. If not, the path leads to state *"Right Not Permitted"*. In this case, the controller updates the variable *depAut* to prevent the controller to cycle infinitely on this test.

From state *"Seek Neighbor"*, the controller may then check if the target neighbor has already been visited. If it has, the path leads to state *"Right Not Permitted"*. Otherwise, the variable `authorization` is updated and next state is *"Check Authorization"*. The value of `authorization` is calculated from precedence constraints (Figure 2).

From state *"Check Authorization"*, the output transition bears a guard on the value of *authorization*. If the value is true, then a command event may be sent to the movement automaton and next state is *"Position"*. Otherwise, next state is *"Right Not Permitted"*.

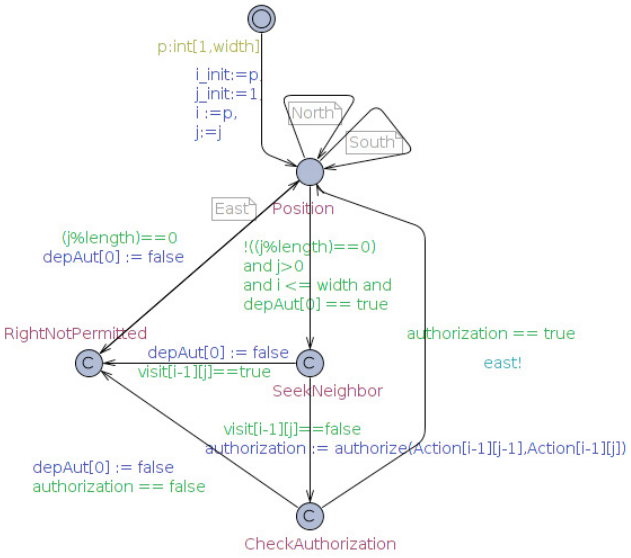

Fig. 3. The control automaton

Movement automaton (Figure 4): This automaton represents the actual robot displacement and memory of the movements. It receives synchronization signals from the control automaton (*north, south, east*). After a movement to the east, for example, the movement automaton updates its variables. The new position of the robot becomes the adjacent cell to the east of previous cell. The new current cell is marked as visited (the MB model updates a boolean array of the size of the grid) and the action attached to the cell is performed by the robot.

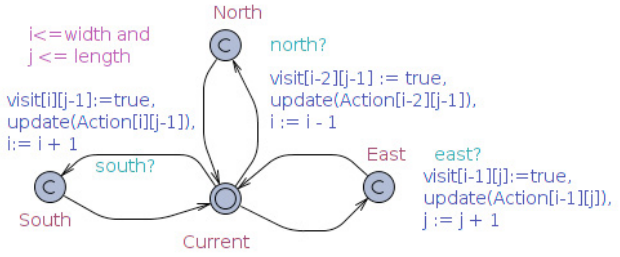

Fig. 4. The movement automaton

It may be noted that MB model is a network of untimed automata. A CTL model-checker, such as e.g. LTSmin [1], could then be used in this simple case. The reason for UppAal choice is that for more realistic applications, the precedence constraints automaton would be replaced by a timed model of mobile equipment dynamics. So, timed models and TCTL model-checking will be required in the end. There exist several TCTL model-checkers such as RED (hybrid automata), TAPAAL and ROMEO (Timed Petri Nets) and UPPAAL (timed automata). UppAal fits our needs.

2.3 Reachability property verification

As said in introduction, the objective of the paper is to propose a method for verifying *a priori* a robotic mission. We consider here that the mission on the grid needs to be fulfilled for any possible start point on west border and any possible end point on east border.

Let us define two functions on a rectangular grid G. $\mathcal{B}_{\mathcal{W}}$ stands for West Border, and $\mathcal{B}_{\mathcal{W}}(G)$ returns the subset of cells that belong to the west border of grid G. Similarly, $\mathcal{B}_{\mathcal{E}}(G)$ returns the subset of cells that belong to the East Border of G. Let us denote by $w \longrightarrow e$ the path (succession of cells) from cell w to cell e, verifying the actions precedence constraints.

The verification property studied in this paper is

$$\forall\, w \in \mathcal{B}_{\mathcal{W}}(G), \forall\, e \in \mathcal{B}_{\mathcal{E}}(G), \exists\, w \longrightarrow e \quad (\Pi)$$

We call the property Π *total reachability side to side property.*

From preliminary studies it came that, in order to verify Π, memory requirements could exceed the possibilities of a usual PC (8Gb) for rectangular grids as small as 4x32 cells. The subject of this paper is to study if a decomposition

[1] *https://fmt.cs.utwente.nl/tools/ltsmin/*

method may help to manage this problem. To this end, two approaches were compared:

(1) The first one, called here *Reference*, does not proceed to decomposition and checks (Π) with a set of reachability queries that correspond to each start point on $\mathcal{B}_{\mathcal{W}}(G)$ and each point on $\mathcal{B}_{\mathcal{E}}(G)$ (end of path).
(2) The second one is specific to the spatial nature of the problem and is called here *Decomposition*. It consists in dividing the grid into subgrids that overlap on a *recovery column* and finding points on that column that have specific properties.

3. DECOMPOSITION OF THE PROBLEM

In this section, the Decomposition methodology that we propose is presented, as well as an analysis on how decomposition modifies problem solving. The algorithm is provided and its behavior is described using an example.

3.1 Description of the decomposition methodology

Our approach involves modeling decomposition and verification decomposition. The modeling decomposition (illustrated in Figure 5) consists in decomposing the global grid G into two sub-grids (S_1 and S_2) with a recovery column $\mathcal{RC}$. This column belongs to both sub-grids and verifies $\mathcal{RC} = \mathcal{B}_{\mathcal{E}}(S_1) = \mathcal{B}_{\mathcal{W}}(S_2)$.

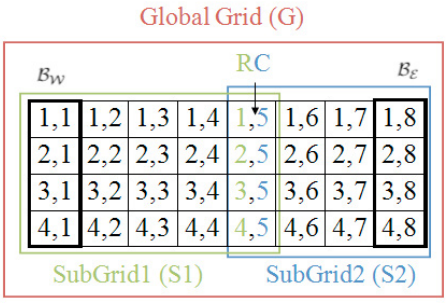

Fig. 5. Example of modeling decomposition

Let us denote P_1 and P_2 the following sets:

$$P_1 = \{u \in \mathcal{B}_{\mathcal{E}}(S1) \;/\; \forall\, w \in \mathcal{B}_{\mathcal{W}}(S1), w \longrightarrow u\}$$
$$P_2 = \{v \in \mathcal{B}_{\mathcal{W}}(S2) \;/\; \forall\, e \in \mathcal{B}_{\mathcal{E}}(S2), v \longrightarrow e\}$$

The verification decomposition consists in decomposing the *total reachability side to side property* Π into two properties Π_1 and Π_2:

$$P_1 \neq \emptyset\ (\Pi_1) \qquad P_2 \neq \emptyset\ (\Pi_2)$$

When Π_1 and Π_2 are verified and $P_1 \cap P_2 \neq \emptyset$, it may be so that the property Π is verified. Conditions are analysed in sub-section 3.2. When $P_1 \cap P_2 = \emptyset$, it is not possible to directly conclude on Π and a further step is needed (see subsection 3.4).

3.2 Analysis of decomposition

Let us suppose that for a given grid, Π_1 and Π_2 are verified and that $P_1 \cap P_2 \neq \emptyset$. Let us consider one element $u \in P_1 \cap P_2$.

Let C_1 be a minimal set of points of $\mathcal{B}_{\mathcal{E}}(S_1)$ that need to be visited in order to reach u, u included. Let C_2 be a minimal set of points of $\mathcal{B}_{\mathcal{E}}(S_1) = \mathcal{B}_{\mathcal{W}}(S_2)$ that need to be visited in order to reach all points of $\mathcal{B}_{\mathcal{E}}(S_2)$ from u, u included.

The necessary condition for Π to be verified, by means of paths that would all include u, is that there exist C_1 and C_2 such that $C_1 \cap C_2 = \{u\}$. This condition is obviously sufficient for proving Π from Π_1 and Π_2. On the contrary, if $\exists v \in C_1 \cap C_2$ with $v \neq u$, then v would be visited twice: once verifying Π_1 (to reach u from $\mathcal{B}_{\mathcal{W}}(S_1)$) and once verifying Π_2 (to reach $\mathcal{B}_{\mathcal{E}}(S_2)$ from v). This would violate the visit constraint.

Considering the MB model of our example problem, the $C_1 \cap C_2 = \{u\}$ condition is sufficient for proving Π from Π_1 and Π_2 using the UppAal verifier.

It can be noted that in case that either Π_1 or Π_2 is not verified, it may be so that Π may still hold. This is because the decomposition heuristic presented so far is based on a single passing point in the recovery column $\mathcal{RC}$ that should include paths of the robot from all points of $\mathcal{B}_{\mathcal{W}}(S_1)$ to all points $\mathcal{B}_{\mathcal{E}}(S_2)$. Π may still hold if several passing points on the recovery column can be used. The procedures described in 3.4 handle this case.

3.3 Decomposition algorithm

Let us introduce some further terminology. A *potential point* is a point u from $P_1 \subset \mathcal{RC}$. It can be reached from all points of $\mathcal{B}_{\mathcal{W}}(S_1)$. Let $u \in C_1$ with C_1 defined as in 3.2. We call *order of* u the cardinal of C_1. A *critical point* u is a potential point from which all points of $\mathcal{B}_{\mathcal{E}}(S_2)$ can be reached. $u \in P_1 \cap P_2$ and it verifies $C_1 \cap C_2 = u$.

The outline of the Decomposition algorithm is as follows:

(1) The models of systems over sub-grids S_1 and S_2 are generated from the model of system over G.
(2) A search of all potential points of order 1 in $\mathcal{RC}$ is performed, and the result is PP_{o1}, subset of $\mathcal{RC}$.
(3) If $PP_{o1} \neq \emptyset$, a critical point is searched in this set.
(4) If at least one critical point exists, Π is verified and the problem is solved.
(5) If no critical point has been found at this step, proceed to East, and then West, Repartition (see 3.4).
(6) Otherwise, the process is repeated for the following orders, until problem is solved or order is greater than the grid width.

The verifier of UppAal requires a specific language based on a BNF-grammar to define properties. Reachability properties are defined as $E <> p$ where p is an expression. In the following, we provide examples of p expressions for several query types. == is the equality sign in UppAal language.

search potential point order 1 :

$$\begin{aligned} &\text{Let } w(x_1, y_1) \in \mathcal{B}_{\mathcal{W}}(S_1) \text{ and } u(x_2, y_2) \in \mathcal{RC} \\ &\qquad p : i_{init} == x_1 \ \textit{and}\ j_{init} == y_1 \\ &\qquad\quad \textit{and}\ i == x_2 \ \textit{and}\ j == y_2 \\ &\qquad\quad \textit{and}\ p_{visit} \\ &p_{visit} : \forall (x, y) \in \mathcal{RC}\ (x, y) \neq (i, j), \\ &\qquad\quad visit[x][y] == false \end{aligned} \tag{1}$$

search potential point order 2 : Let V be a subset of $\mathcal{RC}$ of cardinal 2 such that its elements are adjacent on the grid. Let $w(x_1, y_1) \in \mathcal{B}_{\mathcal{W}}(S_1)$ and $u(x_2, y_2) \in V$

$$\begin{aligned} p : i_{init} == x_1 \ and \ j_{init} == y_1 \\ and \ i == x_2 \ and \ j == y_2 \\ and \ p_{visit2} \\ p_{visit2} : \forall (x, y) \in V, visit[x][y] == true \\ and \ \forall (x, y) \in RC/V, visit[x][y] == false \end{aligned} \tag{2}$$

search critical point (from order 1 potential point): Let $u(x_1, y_1) \in PP_{o1}$, u is starting point, and $e(x_2, y_2) \in \mathcal{B}_{\mathcal{E}}(S_2)$

$$\begin{aligned} p : i_{init} == x_1 \ and \ j_{init} == y_1 \\ and \ i == x_2 \ and \ j == y_2 \end{aligned} \tag{3}$$

The queries for higher orders can easily be deduced from the examples given above.

3.4 Repartition

As was pointed out in 3.2, Π may be true even in the absence of critical points. The Decomposition algorithm includes means to further augment the search of solutions without more queries with the model-checking tool. It includes a marking method that memorizes, from the queries, the existence of a path from each element of $\mathcal{B}_{\mathcal{W}}(S_1)$ to each element of $\mathcal{RC}$ and from each element of $\mathcal{RC}$ to each element of $\mathcal{B}_{\mathcal{E}}(S_2)$. The Decomposition algorithm proceeds to what we call **East repartition**. When there is no critical point, East repartition searches if, using together all potential points, paths from each element of $\mathcal{B}_{\mathcal{W}}(S_1)$ to each element of $\mathcal{B}_{\mathcal{E}}(S_2)$ can be created. The East repartition procedure is inserted in the algorithm as follows:

East Repartition: Formally Π is verified by an East repartition iff:

$$\begin{aligned} &\exists R \subset \mathcal{RC} / \ \forall u \in R, \forall w \in \mathcal{B}_{\mathcal{W}}(S_1), w \longrightarrow u \\ &\wedge \forall u \in R \ \ \exists C_u \subset \mathcal{B}_{\mathcal{E}}(S_2) / \ \forall e \in C_u, u \longrightarrow e \\ &\wedge \bigcup_{u \in R} C_u = \mathcal{B}_{\mathcal{E}}(S_2) \end{aligned} \tag{4}$$

The East Repartition technique provides paths in S_2 in order to look for a solution to Π when no critical point is found in $\mathcal{RC}$. A similar technique can be applied to S_1 when no potential point is found or when East Repartition has been applied to each order step and no solution was found. We call this repartition **West Repartition**.

3.5 Illustrative example

We provide here an illustration of the execution of the Decomposition algorithm. We number the cells of the rectangular grid G in the following manner: index u of a point of coordinates (i, j) is calculated as $u = (i-1) \times l + j$, where l is grid length. The index in the sub-grids remains the same. The Figure 6 gives an example of a distribution of actions that we will use to illustrate the behavior of the algorithm.

(1) Once sub-grids S_1 and S_2 have been created, the search in $\mathcal{RC}$ of all potential points of order 1 is performed. Here, point 13 is reachable at order 1 only from 1 and 9 (see red lines in the Figure 7). Because of precedence constraints, there is no path at order 1 from 17 and 25 to 13. Point 21 is reachable at order 1 only from 17 and 25. Because there is no potential point of order 1, it must be proceeded to order 2.

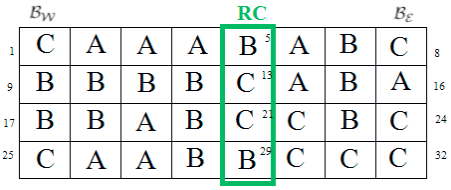

Fig. 6. Example of distribution of actions for G

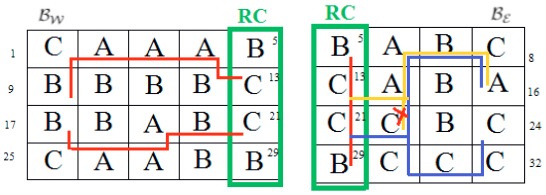

Fig. 7. Execution in S_1 Fig. 8. Execution in S_2

(2) Search in $\mathcal{RC}$ of all potential points of order 2 is performed. 13 and 21 are potential points of order 2 because there exists a path in both directions between them. 13 is reachable at order 1 from 1 and 9, and is reachable from 17 and 25 through 21. Similar reasoning can be made for point 21.

(3) It must now be verified if potential points 13 and 21 are critical (i.e. in S2). 21 is critical and 13 is not (see Figure 8). Indeed, 8 and 16 can be reached from 13 through 14, 6, 7, 8, 16. There are paths from 13 to 24 and 32 through 21, but these are not permitted since 21 has already been visited in sub-grid S_1. Precedence constraints forbid other paths from 13 to 24 and 32. From 21, 8 and 16 can be reached through 22, 14, 6, 7, 8, and 16. Points 24 and 32 can be reached through 22, 30, 31, 32, and 24. As 21 is a critical point, Π is verified over G and the problem is solved.

4. RESULTS & DISCUSSION

In this section, we study the performance of Decomposition algorithm for different scenarios and we compare it with the Reference algorithm (see 2.3). For a possible moves from a cell (here $a = 3$), the complexity of Reference algorithm with respect to grid size w (width) x l (length) is $a^{w.l}$ and the complexity of Decomposition is $2xa^{w.l/2}$. We focus on the impact in memory requirement and time execution.

4.1 Comparison methodology

From preliminary studies about memory behavior, testing a set of grid cases, it was decided to use grid sizes of 4x16 and 4x32. The following test base was built for 4x16 size (size I): 10 4x16 random grids for which Π is not verified, 11 4x16 grids for which Π is verified with a solution involving a potential point at order 1, including the case 'allA' with action A assigned to all cells, and 10 others with solution at order 2. For orders 1 and 2, except 'allA', the

Table 1. Comparison in Memory (M) and Execution Time (ET) between Reference algorithm and Decomposition algorithm

Classe	Scenarios	Reference		Decomposition	
		ET (s)	M(KB)	ET (s)	M(KB)
Order 1	I.1	3.1	14,936	2	84
	I.2	2.9	16,432	2	84
	I.3	1.6	92	2	84
	I.4	3.0	11,572	2	84
	I.5	3.6	16,448	2	84
	I.6	3.5	14,928	2	84
	I.7	4.7	18,588	2	84
	I.8	5.2	19,876	2	84
	I.9	3.9	18,356	2	84
	I.10	5.0	28,724	2	92
	all A	412.8	2760,436	3.3	18,756
Order 2	I.1-10	1.6	84	4.6	84
No solution	I.1	2.0	12,024	10	84
	I.2	1.6	84	10	84
	I.3	1.6	92	10.4	84
	I.4-8,10	1.6	84	10.4	84
	I.9	1.6	84	9.4	84
Order 1	II.1	986.72	7191,716	3.6	17,604
	II.2	667.0	¿28305,268	8.6	56,712
	II.3	4738.2	29011,768	4.0	39,756
	II.4	5212.2	34719,696	4.5	30,540
Order 2	II.1	1575.7	18623,220	97.4	192,928
	II.2	2782.7	25417,372	160.6	346,444
	II.3	2618.6	21659,564	160.5	346,492
	II.4	6211.8	86793,712	578.9	346,464

grids were produced thanks to an ad-hoc method (mixing cells with pre-determined actions and cells with random chosen actions). For size 4x32 (size II), 4 cases were defined at order 1 and 4 other cases for order 2. We call each grid of the test base a scenario. We group scenarios in classes according to their *a priori* difficulty for solving by decomposition.

The results of tests are given in the table 1. These results were obtained with UppAal verifier in Breadth First Search (BFS) mode. This mode was used because it allows a fair comparison with the basic method and because it should exhibit less variation in performance between queries for a given grid.

In order to analyse the performance results, it is useful to consider the following points:

- With the Reference algorithm on grids of width 4×16, 4^2 reachability queries are always addressed to the UppAal model checker.
- With the Decomposition algorithm, queries are made on smaller grids but are more numerous. For solution at first order, 20 to 32 queries need to be addressed to the verifier: 16 reachability queries in sub-grid S_1 and between 4 to 16 reachability queries for sub-grid S_2. Solution at greater orders require still more queries.
- UppAal model checker uses on-the-fly searching technique: it stops when the first solution of the reachability property is found. That is why, it does not generate all the states graph. Nervertheless, in BFS order, the solution is far from the root of the graph because it can be found only at east border of the grid. This makes a difference for Decomposition algorithm which works on half-grids.
- The size of states graph depends on the distribution of actions in the grid.
- UppAal allocates an initial memory bloc of size 84 KB when starting to execute query [2] .

In table 1, the columns named ET (Execution Time) give the total time required by the all needed queries. The columns named M (Memory) give the required memory by the query that required maximum memory.

4.2 Memory performances

From all the test base, it seems that the Decomposition algorithm succeeds in reducing memory requirements. For the "all A" case, which is a worst case for order 1, the reduction is, as expected, spectacular.

In some cases like scenario I.3 in order 1, memory saving is less apparent. Our hypothesis is that the complexity of the path generation of the initial grid is generated by one half of the grid, whereas the rest of the grid has a moderate effect on stored paths. In that case, the decomposition method should have a lower effect on complexity with regards to the basic reference method.

For size I, at second order and for last class (Reference algorithm finds that Π is false), the Decomposition algorithm offers comparable memory requirements as Reference.

The 8 scenarios of size II (4x32 grid) were tested in order to demonstrate the effect of the exponential complexity according to grid size. The memory requirements with Reference algorithm is between 7Go and 87 Go whereas the Decomposition algorithm could solve all cases, with memory requirement below 0.4 Gb.

In conclusion, the Decomposition algorithm succeeds in reducing memory requirements and it allows to address grids of bigger sizes than the Reference. Moreover, the Decomposition methodology has potential for grids of bigger size than those used in the experiments. Indeed, the Decomposition principle can be applied recursively until finding the granularity that makes problem solvable.

4.3 Time performances

At first order, the Decomposition algorithm solves Π with 20 to 32 queries. The average query time is 0.10s for 4x16 grids and 12.46s for 4x32 grids. The Reference algorithm requires 16 queries, which have an average execution time of 0.22s for 4x16 grids and 206.1s for 4x32 grids.

At the second order, the Decomposition Algorithm solves the problem using 46 queries in each scenario case for 4x16 (size I) grids with an average query time of 0.1s, which is the minimum resolution for measurement of elapsed time. For these cases, the Reference algorithm is better than the Decomposition algorithm in execution time. For 4x32 (size II) grids, Decomposition performs better than Reference.

For the no solution class (size I), the Decomposition algorithm requires more execution time than the Reference algorithm and it solves the problem (Π being false) using [94 - 104] queries.

[2] *https : //www.it.uu.se/research/group/darts/uppaal/press/uppaal − 3.4.shtml*

5. CONCLUSION AND FUTURE WORK

In this paper, it was investigated how model-checking can be applied to address spatial properties for mobile robots. We have proposed a decomposition methodology that is specific to the spatial nature of the problem and that decomposes both modeling and verification. The structure and motivations of the Decomposition algorithm were explained. The experimental results provided suggest that the decomposition algorithm is efficient in reducing memory requirement in comparison to a basic reference method. In some cases, the decomposition algorithm also reduces model-checking execution time.

The decomposition principle can be applied recursively, which offers potential to solve large grids. The Decomposition methodology could also be developed further, by dividing the initial grid in n sub-grids (S1,.., Sn) instead of 2. This enhanced decomposition methodology would apply a similar decomposition technique for S1 and Sn. Then, it would search for a path from a potential point u on east border of S1 to a potentially critical point v on west border of Sn. This enhanced approach may be more interesting than applying recursively decomposition methodology because it would reduce the number of queries from n.C to 2.C+k where C is the number of queries in a sub-grid and k is the number of queries for searching a single path from u to v Another further step would be to develop methods that would be applicable to both safety and reachability properties. The theory of Timed games (Pnueli et al. (1998)) also seems relevant for supporting control design for mobile robots acting on a spatial grid.

ACKNOWLEDGMENT

This work has received the support of French National Research Agency under the grant number ANR-14-CE27-0004 attributed to AdAP2E project.

REFERENCES

Al-A'Abed, M., Majali, T., Omar, S., Alnawaiseh, A., Al-Ayyoub, M., and Jararweh, Y. (2015). Building a prototype for power-aware automatic parking system. *Proceedings of 2015 IEEE Int. Renewable and Sustainable Energy Conference, IRSEC 2015*.

Alur, R., De Alfaro, L., Henzinger, T.A., and Mang, F.Y. (1999). Automating modular verification. In *Int. Conf. on Concurrency Theory*, 82–97. Springer, Berlin, Heidelberg.

Alur, R. and Dill, D.L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126, 183–235.

Bechar, A. and Vigneault, C. (2016). Agricultural robots for field operations: Concepts and components. *Biosystems Engineering*, 149, 94–111.

Bengtsson, J. and Yi, W. (2004). *Timed Automata: Semantics, Algorithms and Tools*, 87–124. Springer, Berlin, Heidelberg.

Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., and Juan, A.A. (2015). Rich vehicle routing problem: Survey. *ACM Computing Surveys (CSUR)*, 47(2), 32.

Claes, D., Robbel, P., Oliehoek, F.A., Tuyls, K., Hennes, D., and van der Hoek, W. (2015). Effective approximations for multi-robot coordination in spatially distributed tasks. In *Proceedings of the 2015 Int. Conf. on Autonomous Agents and Multiagent Systems*, 881–890. International Foundation for Autonomous Agents and Multiagent Systems.

Clarke, E., Talupur, M., Touili, T., and Veith, H. (2004). Verification by Network Decomposition. In P. Gardner and N. Yoshida (eds.), *CONCUR 2004 - Concurrency Theory*, 276–291. Springer Berlin Heidelberg.

Hélias, A., Guerrin, F., and Steyer, J.P. (2008). Using timed automata and model-checking to simulate material flow in agricultural production systems - application to animal waste management. *Computers and Electronics in Agriculture*, 63(2), 183–192.

Herrera, D., Tosetti, S., and Carelli, R. (2016). Dynamic modeling and identification of an agriculture autonomous vehicle. *IEEE Latin America Transactions*, 14(6), 2631–2637.

Kayacan, E., Kayacan, E., Ramon, H., and Saeys, W. (2013). Modeling and identification of the yaw dynamics of an autonomous tractor. *9th Asian Control Conference, ASCC 2013*.

Koo, H.M. and Mishra, P. (2006). Functional Test Generation Using Property Decompositions for Validation of Pipelined Processors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '06, 1240–1245. European Design and Automation Association, 3001 Leuven, Belgium.

Largouët, C., Cordier, M.O., Bozec, Y.M., Zhao, Y., and Fontenelle, G. (2012). Use of timed automata and model-checking to explore scenarios on ecosystem models. *Environmental Modelling & Software*, 30, 123–138.

Larsen, K.G., Larsson, F., Pettersson, P., and Yi, W. (2003). Compact data structures and state-space reduction for model-checking real-time systems. *Real-Time Systems*, 25(2-3), 255–275.

Larsen, K.G., Pettersson, P., and Yi, W. (1997). Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2), 134–152.

Leahy, K., Jones, A., Schwager, M., and Belta, C. (2015). Distributed information gathering policies under temporal logic constraints. In *2015 54th IEEE Conf. on Decision and Control (CDC)*, 6803–6808. IEEE.

Lenain, R, c. (2014). "anr adap2e project reference anr-14-ce27-0004 adaptive autonomous production platform for environment". URL `"http://www.agence-nationale-recherche.fr/?Project=ANR-14-CE27-0004"`.

Pnueli, A., Asarin, E., Maler, O., and Sifakis, J. (1998). Controller synthesis for timed automata. In *Proc. System Structure and Control. Elsevier*. Citeseer.

Souissi, O., Benatitallah, R., Duvivier, D., Artiba, A., Belanger, N., and Feyzeau, P. (2013). Path planning: A 2013 survey. *Proceedings of 2013 Int. Conf. on Industrial Engineering and Systems Management, IEEE - IESM 2013*.

Vanwalleghem, T., Jimnez-Hornero, F., Girldez, J., and Laguna, A. (2010). Simulation of long-term soil redistribution by tillage using a cellular automata model. *Earth Surface Processes and Landforms*, 35(7), 761–770.

Wainer, G.A. and Giambiasi, N. (2001). Application of the cell-devs paradigm for cell spaces modelling and simulation. *Simulation*, 76(1), 22–39.