A Situation Analysis Toolbox: Application to Coastal and Offshore Surveillance

Patrick Maupin, Anne-Laure Jousselme

R & D Defence Canada - Valcartier
2459 Pie XI North
Quebec, QC Canada, G3J 1X5
Patrick.Maupin@drdc-rddc.gc.ca

Hans Wehn, Snezana Mitrovic-Minic, Jens Happe

MacDonald, Dettwiler and Associates (MDA)
13800 Commerce Parkway
Richmond, BC Canada, V6V 2J3
hwehn@mdacoporation.com

Abstract – *In this paper, we present a toolbox to evaluate motion strategies in a realistic surveillance context for the purpose of enhancing decision support capabilities. Five components of the toolbox (Discretization, State Generation, State Searching, Behaviour Simulation and Visualization) implement the theoretical concepts put forward in previous works which outlined formal definitions of situation, situation awareness and situation analysis defined with the interpreted systems semantics.*

Keywords: Situation analysis, interpreted systems, visibility graphs, pursuit-evasion.

1 Introduction

Situation Awareness is essential to conduct decision-making activities. It is about the perception of the elements in the environment, the comprehension of their meaning, and the projection of their status in the near future [1]. Situation Analysis (SA) is defined as a process, the examination of a situation, its elements, and their relations, to provide and maintain a product, *i.e.* , a state of situation awareness for the decision maker [2]. This follows a standard intuition about levels 2 and 3 of the standard JDL Information Fusion model where, according to Steinberg and Bowman [3] Situation Assessment (level 2) is the "*estimation and prediction of entity states on the basis of inferred relations among entities*" whereas Impact Assessment (level 3) "*is usually implemented as a prediction, drawing particular kinds of inferences from Level 2 associations*".

The situation analysis process has to evaluate with both knowledge and uncertainty. A formalization is necessary if one is interested in the reproducibility of results, complex space-time problem solving, and in a language to represent and reason about dynamic situations. In [4], we proposed a formal model for situation analysis based on the interpreted systems semantics [5], a framework in which knowledge, information and uncertainty can be represented, combined, managed, reduced, increased, and updated.

The general problem of decision support can be sparsed into two principal tasks: situation analysis and planning. These two tasks are not easily separable since the actions carried out according to the plan defined during the task of planning modify the state of the environment and of the agents. This modifies the information to which the agents have access (their own or collective epistemic state) and thus their situational awareness. Our approach to situation analysis considers that the situation is created by the execution of a joint protocol for the agents P in interaction with the environment's protocol P_e, the latter including a possible opponent. The interpreted systems semantics will be used as the single specification language for both planning and situation analysis [6].

In this paper, we present a Situation Analysis Toolbox (SAT), which implements the theoretical concepts put forth in our previous works [7, 4, 6]. The general idea behind this toolbox is to build situations and analyse them. We use the formalism of the non-cooperative dynamic game theory [8] to model the situation. In its most general form it consists in a game between two teams having opposite goals, the players jointly seeking to maximize (resp. to minimize) a function of distance between targets. Differential game theory of Isaacs [9], dating back to the 1960s, made it possible to model the pursuit-evasion game - a simplified version of the problem of rendez-vous [10]. Isaacs' theory can be used to model many military problems by combining the traditional game theory of von Neuman and Morgenstein [11] (used especially in economy), variations calculus, and theory of optimal control. The interest to use such a framework for modelling the situations to be further analyzed is the existence of some analytically derivable optimal solutions, thus helping to measure the quality of solutions for simple cases. Another advantage of this theoretical framework is its great flexibility. Indeed, by slightly modifying the constraints of the basic game, a large range of situations can be easily modeled: collision avoidance, target tracking, or stowing.

SAT consists of five (5) components (sub-toolboxes): (1) the Discretization Toolbox, which allows an abstraction of a continuous environment into both visibility and navigation graphs, (2) the Behaviour Simulation Toolbox, which enriches the environment with containment probability maps, (3) the State Generation Toolbox, which builds a transition state system based on joint strategies of some agents constrained by the context defined in (1) and (2), (4) the State Searching Toolbox, which plays the role of a model checker and temporal logical formulas verification in the transition state system, and (5) the Visualization Toolbox, which renders states and graphs on screen.

The paper is organised as follows. Section 2 is dedicated to theoretical background including formal definition of a situation, situation awareness, and situation analysis framed into the interpreted system semantics, to basics of pursuit-evasion games. This background section, also describes a counter-smuggling scenario situated in Howe Sound (on Canada's West Coast, northwest of Vancouver) on which the SAT will be illustrated. Section 3, presents the Situation Analysis Toolbox and details of its five components together with their related theoretical concepts. Finally, Section 4 provides further potential use and applications of the SAT.

2 Theoretical Background

In the following Section, we will give formal definitions of situations, situation awareness and situation analysis as well as basics on visibility-based pursuit-evasion games that will be used to model the situations.

2.1 Interpreted Systems for Situation Analysis

The interpreted systems semantics has been introduced in [5] and proposed in [7, 4] as a formal language for situation analysis.

Let $\mathcal{A} = \{1, 2, 3, \ldots, n, e\}$ be a set of agents, where e is a special agent denoting the *environment*. Each agent is assumed to be in some *local state* l_i at a given time, encapsulating all the information the agent has access to. l_e encodes all the relevant information that is not encoded in the agents' local states. In particular, l_e can encode the objective state of the world, which is usually not attainable by the agents' perception and reasoning means. For SA applications, apart from the objective states of the world, l_e can contain maps of the environment, network information, or any other information describing the outside world. The agents' local states can encode partial or imperfect views of this outside world.

A *global state* s is an element of $S \subseteq L_1 \times \ldots \times L_n \times L_e$, where L_i is the set of all possible local states of agent i. A sequence of global states $s^1, s^2, \ldots$ is called *run* r over S and can be viewed as a function from time to global states. A *system* $\mathcal{R}$ is a set of runs, and (r, m) denotes a *point* in $\mathcal{R}$. If $r(m) = (l_1, \ldots l_n, l_e)$ is the global state at point (r, m) (the state of the system at time m in run r), then $r_e(m) = l_e$ and $r_i(m) = l_i$. A *round* m in run r takes place between time $m-1$ and time m.

Actions are the cause of changes in the system and are performed by the agents and the environment in rounds. Let ACT_i be the set of actions that can be performed by agent i, $i = 1, \ldots, n$ and let ACT_e be the set of actions performed by the environment. A *joint action* is an element of $ACT_e \times ACT_1 \times \ldots \times ACT_n$, *i.e.* a tuple $(\mathbf{a}_e, \mathbf{a}_1, \ldots, \mathbf{a}_n)$ of actions performed by the set of agents and the environment. Λ denotes the null action.

A *protocol* P_i for the agent i is a mapping from the set L_i of local states of the agent i to nonempty sets of actions in ACT_i, $i \in \mathcal{A}$. A *joint protocol* P is a tuple $(P_1, P_2, \ldots, P_n)$ consisting of the protocols of each of the agents i, $i = 1, \ldots, n$. Note that P_e, the protocol of the environment, is not included in the joint protocol, but it is rather a part of the context.

A *context* γ is a tuple (P_e, S_0, τ, Ψ) where P_e is a protocol for the environment, S_0 is a nonempty subset of S describing possible states of the system at the initiation of the protocol, τ is a transition function, and Ψ is an admissibility condition on runs. The environment's protocol P_e can be used to model the adversary's strategies or simply to model random errors or events in a given situation. A *transition function* τ assigns to each global state and each joint action the resulting global state obtained after performing the joint action. The transition function describes which actions can be performed from a given global state, as well as the effect of these actions. The *admissibility condition* Ψ on runs tells which ones are "legal". In practice, Ψ can be used to shrink down a large system, or to model fairness conditions. Formally, Ψ is a set of runs and $r \in \Psi$ if and only if r satisfies the condition Ψ. Note that the description of the behaviour of a system is contextual, *i.e.* , a joint protocol P is always described within a given context γ.

Let Φ be a set of primitive propositions, describing basic facts about the system. Formulas are built using the classical operators of propositional logic. The set of formulas is closed under $\neg$ and $\wedge$ (negation and conjunction). Hence, given two formulas ϕ and ψ, $\neg\phi$, $\phi \wedge \psi$ are also formulas. Let $\mathcal{L}(\Phi)$ denote the language of Φ, *i.e.* the set of well-formed formulas.

An *interpreted system* (IS) $\mathcal{I}$ consists of a pair $\langle \mathcal{R}, \pi \rangle$ where $\mathcal{R}$ is a system over a set S and π is an interpretation for the propositions in Φ over S, which assigns truth values (either true or false) to the primitive propositions at the global states. Thus, for every $p \in \Phi$ and state $s \in S$, we have $\pi(s)(p) \in \{false; true\}$.

We define a situation in terms of a transition state system in which the arcs are labeled by joint actions and nodes by global states. Formally, let $\mathcal{A}$ be a set of

$n+1$ agents including the environment, and let $\mathcal{I}$ be the interpreted system representing the joint protocol P of the n agents in the interpreted context (γ, π).

Definition 2.1 (Situation [4]) *A* situation *is the sub-system $\mathcal{I}_{(r,m)}$ of $\mathcal{I}$, that is the system representing P in the interpreted context $(\gamma_{(r,m)}, \pi)$ where $\gamma_{(r,m)} = ((r,m), P_e, \Psi, \tau)$.*

Situation analysis is the process by which the decision maker (or analyst) reaches a state of situation awareness which will then allow him (or her) to make decisions.

Definition 2.2 (Situation analysis [4]) Situation analysis *is the process of verifying properties of $\mathcal{I}$. If ϕ is a formula of $\mathcal{L}(\Phi)$ expressing such a property, then analysing the situation amounts to answering the question*

$$(\mathcal{I}, r, m) \vDash \phi \tag{1}$$

If ϕ is satisfied in $\mathcal{I}$ at (r,m), then the analyst of the system is said to be aware *of ϕ.*

If the verification algorithm (*e.g.* model checking) is sound (*i.e.* always gives correct answers), then the analyst has explicit knowledge of ϕ (see [4] for details).

Epistemic model checking is used as a technique for (1) studying the consequences of the joint execution of protocols on the mental states of the agents, (2) data mining allowing to analyze in a qualitative way the state transition systems thus obtained, and possibly (3) allowing to check the existence of certain equilibrium conditions of games in terms of epistemic group notions.

Awareness however, is not simply a special state of knowledge but also refers to a limited capacity of the agents. Considering the limited resources of agents, we proposed in [4] a definition of awareness inspired by [12] and [13]:

Definition 2.3 (Awareness [4]) *The local state of each agent i at point (r,m) includes both a local algorithm $\mathsf{Aw}_i = \mathsf{alg}_i(r,m)$ and local data $l_i = \mathsf{obs}_i(r,m)$. An agent i of $\mathcal{A}$ is* aware *of ϕ at a given point (r,m) in $\mathcal{I}$, which we denote by $A_i\phi$, iff it is able to compute the truth value of ϕ:*

$$(\mathcal{I}, r, m) \vDash A_i\phi \text{ iff } \mathsf{A}_i(\phi, l_i) \neq \text{“?”} \tag{2}$$

The interpretation of A_i is thus understood as capturing any constraints like time, memory, reasoning abilities, etc.

Definition 2.4 (Situation awareness [4]) *For an agent i of $\mathcal{A}$, the* situation awareness *at point (r,m) is the set of formulas of $\mathcal{L}(\Phi)$ about which the agent i is aware:*

$$\mathsf{Aw}_i(r,m) = \{\phi \in \mathcal{L}(\Phi) | (\mathcal{I}, r, m) \vDash A_i\phi\} \tag{3}$$

Situation awareness is thus defined in terms of states, *i.e.* , in terms of a set of points in the system. For a given agent, the situation awareness is provided by test evaluations on observations about the environment (the objective state of the world). Situation awareness is both an epistemic ability and a precondition for action [6]. This is a practical definition of knowledge.

2.2 Situations as pursuit-evasion games

We use the formalism of the non-cooperative dynamic game theory [8] to model the situation. Pursuit-Evasion (PE) games is a family of problems in which one group attempts to track down members of another group in an environment. The discrete formulation of the problem is also called graph searching and is due to Parsons [14]. Pursuer and evader agents are thus constrained to move within a graph whose nodes are possible positions (locations) and whose edges denote paths between two locations.

In its most general form, PE consists in a game between two teams having opposite goals, the players jointly seeking to maximize (resp. to minimize) a function of distance.

Visibility-based pursuit-evasion in graphs

We based our formulation on the work of Lavalle *et al.* on information spaces [15, 16]:

1. Let us consider an environment as a weighted (or labeled) graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of the vertices (or nodes), and $\mathcal{E}$ is the set of edges, $\mathcal{E} = \mathcal{V} \times \mathcal{V}$. The weight d on the edge represents the distance between the two vertices connected by the edge.

2. Suppose there are n pursuers $p_1, p_2, \cdots, p_n$ and one evader p_e, each of which can be at any vertex of $\mathcal{G}$. The positions of pursuer $p_i, i = 1, \cdots, n$ and evader e are v_i and v_e respectively. So the state of system is $(v_1, \cdots, v_n, v_e)$, and the state space is $X = \mathcal{V}^{n+1}$.

3. Each agent has a visibility sensor of sensing range r meaning that the agent can see a node if it is within its range. Let us denote by $r(v)$ the set of all the nodes that the agent can see at position v. The pursuers and the evader can have different visibility sensors with different sensing ranges. The *observation space* is the collection of subsets of $\mathcal{V}$.

4. Capture occurs when a pursuer and the evader are at the same position (node) at the same time.

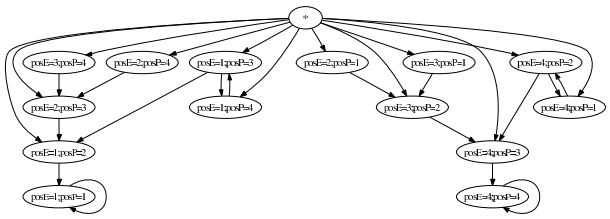
5. The speeds of pursuer p_i and evader are ω_i and ω_e respectively. For the sake of simplicity, we define the speed to be 1 for all the agents, meaning the agent moves from one node to another in one step, possibly accounting for edge's weight.

case of
  if PosE=PosP **do** Λ
  if PosP∈Neighbours (PosE) **do** move(farthest node)
  else Λ
  observe
end case

(a) Evader protocol P_e.

case of
  if PosP=PosE **do** Λ
  if PosE∈Neighbours(PosP) **do** move(PosE)
  else move(any adjacent node)
  observe
end case

(b) Pursuer protocol P_p.



(c) State transition system.

Figure 1: Strategies of two opponent players and corresponding state transition system in a simple case. Λ is the null action (do nothing).

6. A basic action $\mathbf{a}_i$ for agent i is to move from one vertex to an adjacent vertex along an edge.
7. The initial state of the pursuers is either set to some desirable nodes or determined randomly. For the evader, it is set randomly.
8. The evader's strategy P_e which is unknown to the pursuers, is (see Figure 1(a)):
   (a) If the pursuer is not visible, the evading agent stays on its actual position (reactive agent).
   (b) If the evader sees one of the pursuers, it moves to maximize its reward function. If we define the reward function as the distance between the closest pursuer and the evader $d(p_i, e)$, the strategy for the evader is to maximize the distance between itself and any pursuer. If there are many such nodes, the evader moves to minimal index node.
9. The pursuers' strategy P_p is (see Figure 1(b)):
   (a) If the evader is not visible to the pursuers, the pursuers move based on a depth first search algorithm.
   (b) If the pursuers can see the evader, they move to minimize their cost function $d(p_i, e)$.

Figure 1 shows examples of protocols for both pursuer P_p and evader P_e agents together with the corresponding state space. A very simple case consisting of only one pursuer, one evader and 4 possible positions (4 nodes in the navigation graph) is shown, since the state transition system becomes rapidly huge and consequently difficult to visualise. As it will be introduced in Section 3.1, neighbours are defined in two ways: (1) visibility graph and (2) navigation graph. In the latter case, a pursuer moving into a neighbouring cell with an evader catches it, while in the former, it may not.

Local states encode physical terrain positions and hence the possible states of the agents can be stacked up the map, allowing convenient illustrations (see for example Figure 5(b)).

2.3 Counter-smuggling Vignette

We illustrate the SAT on a counter-smuggling vignette shown in Figure 2. In this figure, several boats manned by smugglers (the hostile agents) operate in Howe Sound (close to Vancouver). They attempt to go from their hiding places to target locations where their illegal goods can be exchanged. These agents try to avoid obstacles such as islands, and avoid colliding with each other. They also try to keep a distance away from the neutral agents such as ferries and pleasure crafts to avoid being seen and perhaps cause suspicion. More

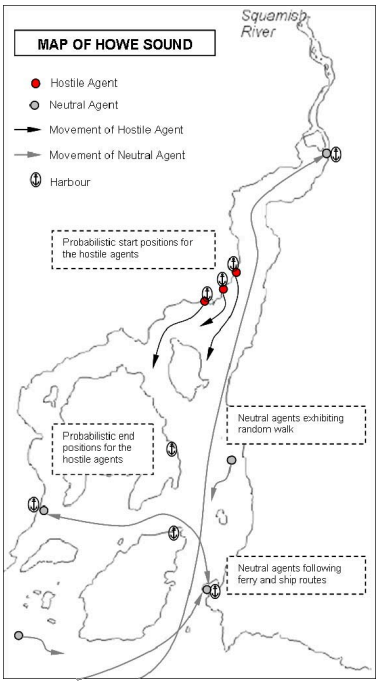


Figure 2: A smuggling operation has been reported in Howe Sound (north-west of Vancouver on Canada's West Coast). Can we guarantee that the smugglers will be detected ?

specifically, the neutral agents can be ferries or ships that follow a specific route, or boats that sail in any

random direction. The locations of the home and target harbours are determined probabilistically.

3 Situation Analysis Toolbox

The SAT software is intended to provide tools for generating situations as well as analysing them. A key component of a situation is its context, which will be mainly represented here by the terrain on which the situation takes place. The area of interest is represented in the form of a polygon (*i.e.* boundary of the search area) with holes (which represent obstacles that the agents can neither see nor pass through). The software is divided into five sub-toolboxes, which serve different functions: (1) Discretization Toolbox, (2) State Generation Toolbox, (3) State Searching Toolbox, (4) Behaviour Simulation Toolbox and (5) Visualization Toolbox.

The high-level data flow diagram given in Figure 3 shows how these components are connected: First, the

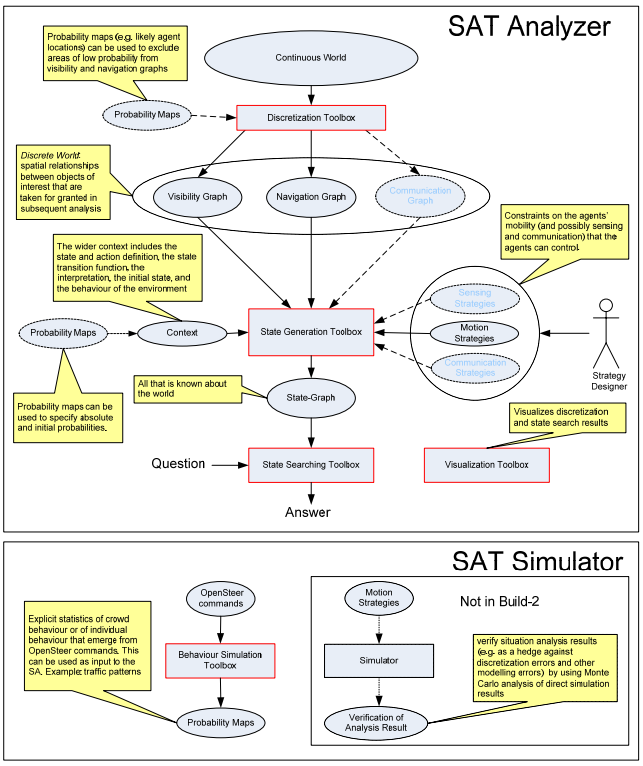


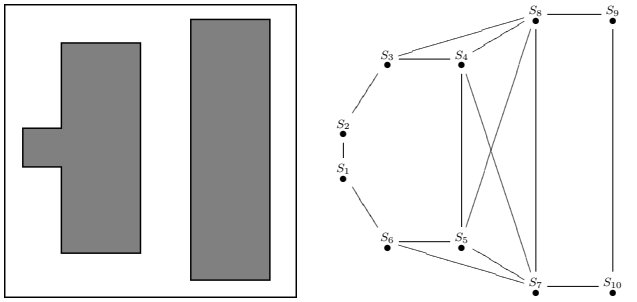
Figure 3: The Situation Analysis Toolbox.

Discretization Toolbox provides an abstraction of the terrain into a visibility graph and a navigation graph, which will then serve as the basis for defining a part of the context γ and the set of possible global states S for the system. Another component of the context comes from the Behaviour Simulation Toolbox, which provides agent containment probability maps of the area under consideration. The State Generation Toolbox inputs both the context $\gamma = (P_e, S_0, \tau, \Psi)$ and the strategies of the agents $P = [P_1, \ldots, P_n]$ and outputs a set of possible states. The situation thus built (*i.e.* , the transition state system) can then be analysed through the State Searching Toolbox, whose purpose is to answer logical queries of interest. The Visualization Toolbox provides graphical user interface for the SAT.

3.1 The Discretization Toolbox

The main task of the Discretization Toolbox is to provide discrete representations of the real-world continuous search space geometry, suitable for situation analysis in the context of surveillance problems. The toolbox supports discrete representations in the form of either convex polygons (cells) or points, shown in Fig. 4. Two main graphs are computed by the Discretization Toolbox: a visibility graph and a navigation graph.

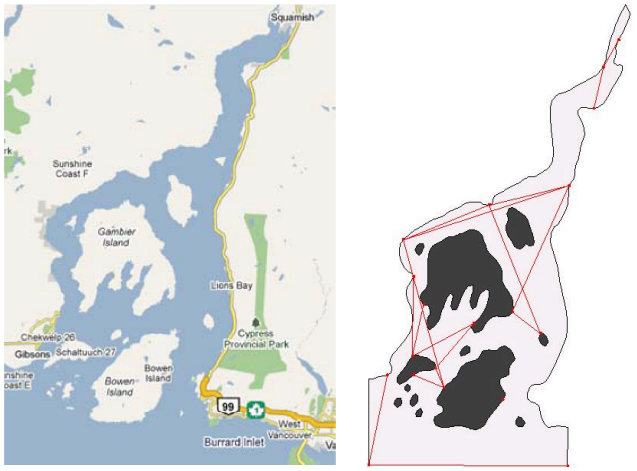
The *visibility graph* of a set of points Q in a polygonal environment is a graph $\mathcal{G}_V(Q)$ whose nodes are the points in Q and in which two vertices are adjacent if



(a) Simple example of a polygonal environment with two holes (obstacles).

(b) Corresponding visibility graph.

Figure 4: Visibility graph computed for a polygonal environment.



(a) Map of Howe Sound,.

(b) 19 patrols (guards) with unlimited range sensors cover Howe Sound. Lines show visibility between guards.

Figure 5: GIS map and associated visibility graph.

they are visible from each other in the environment. An example is shown in Figure 4 for the reflex vertices in a simple polygonal environment, and in Figure 5 for the counter-smuggling scenario. The Discretization Toolbox provides several options for selecting the set Q from among the points in the polygonal environment. The *navigation graph* $\mathcal{G}_N$ defines the transition function τ for the agents . In an agent's navigation graph, the vertices are either points or cells, and two cells/points are connected if an agent can move directly from the one to the other in one time step. As a rule, the navigation graph is a subgraph of the visualization graph as two adjacent points/cells are always visible from one another. The agent's knowledge of the environment is

Algorithm 1 Exploration algorithm of Pellier and Fiorino [17]: Explore(edb, p, γ_e)

Require:

- edb the depth exploration bound
- p the current position of the pursuer

Ensure:

- p the current position updated
- γ_e the exploration path

```
1:  i := 0; γ_e := ∅; γ*_e := ∅; N*_n := Unexplored(G_n);
2:  while i < edb and N*_n ≠ ∅ do
3:      N*_n := Unexplored(G_n)
4:      for all V_c ∈ N*_n do
5:          γ*_e := ComputeMotion(G_n, p, V_c);
6:          ExecuteMotion(γ*_e);
7:          Update(G_n,VisibleVerticesFrom(V_c));
8:          γ_e := γ_e + γ*_e;
9:          MarkExplored(G_n, V_c);
10:         p := V_c;
11:     end for
12:     i++;
13: end while
```

built through an exploration algorithm whose aim is to learn the environment while exploring the navigation graph. Several algorithms such as the one described in [17] and shown in Algorithm 1 are available.

3.2 The State Generator

The State Generation Toolbox strives to be an implementation of the concepts and abstractions introduced in the first part of this paper. It incorporates the concepts of state, agent, action, transition function, and strategy. It provides a state graph which represents an interpreted system $\mathcal{I}$ for the purpose of testing agent behaviour strategies. This module takes as main inputs the abstraction of the terrain under the form of both the visibility and navigation graphs computed by the Discretization Toolbox. Other inputs are: the agents' local states definition l_i which may include other parameters than the positions, the agents's protocols P_i, the context $\gamma = (P_e, S_0, \tau, \Psi)$ and the interpretation function π which indicates whether some Boolean formulas ϕ are true in global states s. Given a state, the State Generation Toolbox outputs a set of successor states. Figure 6

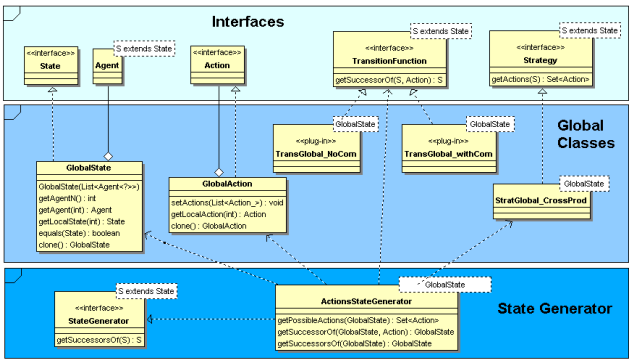


Figure 6: Classes comprising the global State Generator.

shows the classes comprising the global state generator. The main class is a generic state generator that maps the current global state into a successor when given a global action. At this level of abstraction, the state generator can be used generically by the model checker to implicitly generate the state graph and search it to answer Computation Tree Logic (CTL) questions. The model checker can stay ignorant of the particular type of state or action involved, and thus stays generic.

3.3 The State Searching Toolbox

The main task of the State Searching Toolbox is to provide answers to formal queries by searching the state graph $\mathcal{I}$. This module takes as input the state transition function τ and a query ϕ, and outputs the query result. Although basic answers to queries on state transition systems are Boolean (either true or false), degrees can also be computed ranging from probably true

AX ϕ	ϕ in all next states.
EX ϕ	ϕ in at least one next state.
A $[\phi$ **U** $\psi]$	on all paths, ϕ until ψ.
E $[\phi$ **U** $\psi]$	on at least one path, ϕ until ψ.
AF ϕ	On all paths, in some future state, ϕ.
EF ϕ	On at least one path, in some future state, ϕ.
AG ϕ	On all paths, in all future states, ϕ.
EG ϕ	On at least one path, in all future states, ϕ.

Table 1: CTL operators.

to probably false. Also, additional statistical measures can be computed such as: Number of states where ϕ is true/false, Number of paths where ϕ is true/false, Length of path where ϕ is true/false, Length of path before encountering a counterexample, Average length of path, Ratio of states/paths. The formal language of queries is CTL, which is propositional logic over the states in a state graph, augmented by the list of operators shown in Table 1.

The State Searching Toolbox plays the role of a model checker and thus serves as the situation analysis tool proper. Figure 7 shows an instance of the

counter-smuggling vignette. The previously computed restricted search space is discretized into cells, of which

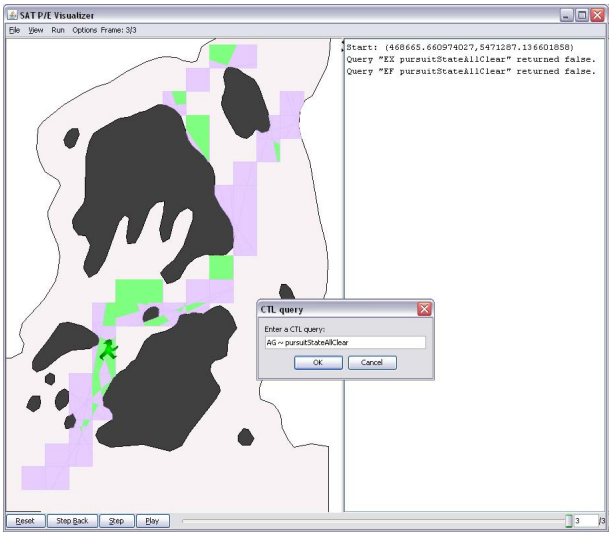


Figure 7: Executing queries using the Visualizer Toolbox for the counter-smuggling vignette.

the green cells are used to compute the visibility graph. In a first step, (both the purple and green) cells have been selected through a threshold on the probability map (see for example Fig. 9(b)). In a second step, a subset of the thresholded cells is further selected and colored green. The green cells are sufficient to cover all thresholded cells using infinite-range omnidirectional sensors. The green guy marks the start position of the pursuer, who is supposed to systematically search for evaders in the gallery. The predicate `pursuitStateAllClear` used in the query ascertains that (all cells in) a given state is (are) clear of evaders—or equivalently, that any evader present in the search area has been caught. The results of past CTL queries are shown on the right-hand-side of the screen. In this case, they return false because the topology of the area allows an evader to hide behind islands and avoid detection, no matter how the pursuer moves. Multiple pursuers would be necessary to completely sweep the area and guarantee detection.

3.4 The Behaviour Simulation Toolbox

The main task of the Behaviour Simulation Toolbox (see Figure 8) is to extract probability maps for agent containment from OpenSteer[1] multi-agent simulations. The OpenSteer is a library implementing "steering behaviours" originally designed by Craig Reynolds [18] to model coordinated motion of animals such as flying birds. Given the description of the scenario in Section 2.3, the Behaviour Simulation Toolbox allows the generation of a map of the most likely locations for finding

[1] `http://opensteer.sourceforge.net`

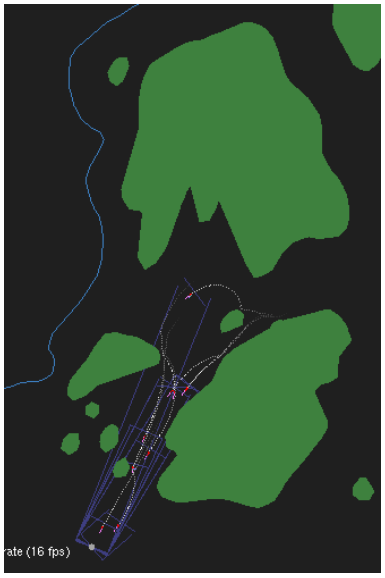


Figure 8: Screen shot of the Behaviour Simulation Toolbox for the counter-smuggling vignette.

the smugglers. The main input to the Behaviour Simulation Toolbox is a GIS map corresponding to the area of interest. The output is a probability map, *i.e.* a set of polygonal cells marked up with their probability of agent containment (Fig. 9).

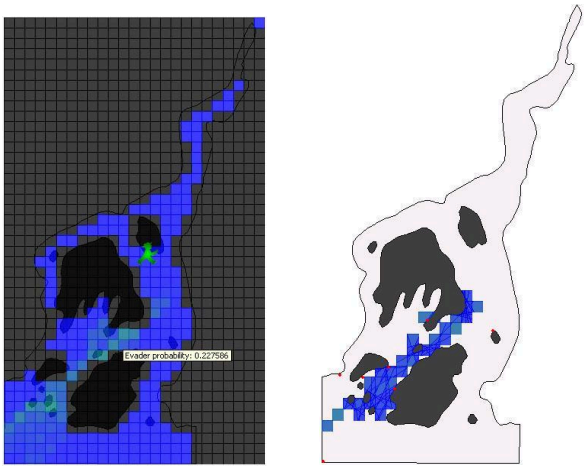
3.5 The Visualization Toolbox

The Visualization Toolbox allows the user to run and visualize the following: discretization of a map into either cells or points (guards) (Figure 5(b)), input of CTL queries for the state search, display of the results of the query, and visualization of the graph search (Figure 7).

The state generator and the visualizer currently support the following problem types [19]: the pursuit-and-evade problem, the sensor placement problem, and the exploration problem.

4 Conclusions

We presented a Situation Analysis Toolbox implementing formal notions of situation analysis based on epistemic transition state systems. Composed of 5 subtoolboxes, the SAT builds a situation based on an abstract version of the environment as a visibility graph and on the execution by a set of agents of a joint strategy derived from pursuit-evasion game theory. The context is further enriched with probability maps of presence of agents, built through modeling emerging behaviour. Using the situational definitions and formalisms, SAT affords modeling, evaluation, and representation of dynamic environments



(a) Probability map of evader presence. Lighter, greenish colours represent higher probability.

(b) Restricted search space: 0.1 probability threshold.

Figure 9: Probability map output by the Behaviour Simulation Toolbox.

Acknowledgments

The authors want to thank Grace Lin for programming on the behaviour simulation toolbox.

References

[1] Mica R. Endsley and Daniel J. Garland. *Situation Awareness Analysis and Measurement*. Lawrence Erlbaum Associates, Publishers, Mahwah, New Jersey, 2000.

[2] Jean Roy. From data fusion to situation analysis. In *Proceedings of the 4th International Conference on Information Fusion*, volume II, pages ThC2–3 – ThC2–10, Montreal, Canada, 2001.

[3] A. N. Steinberg and C. L. Bowman. Revisions to the JDL data fusion model. In D. L. Hall and J. Llinas, editors, *Handbook of Multisensor Data Fusion*, The Electrical Engineering and Applied Signal Processing Series, chapter 2, pages 2–1 to 2–19. CRC Press, Boca Raton, 2001.

[4] Patrick Maupin and Anne-Laure Jousselme. Interpreted systems for situation analysis. In *Proceedings of the 10th International Conference on Information Fusion*, Quebec City, Canada, 9-12 July 2007.

[5] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about knowledge*. The MIT Press, Cambridge, MA, 2003.

[6] Anne-Laure Jousselme, Patrick Maupin, Christophe Garion, Laurence Cholvy, and Claire Saurel. Situation awareness and ability in coalitions. In *Proceedings of the 10th International Conference on Information Fusion*, Quebec City, Canada, 2007.

[7] Patrick Maupin and Anne-Laure Jousselme. A general algebraic framework for situation analysis. In *Proceedings of the 8th International Conference on Information Fusion*, Philadelphia, PA, USA, July 2005.

[8] Tamer Basar and Geert Jan Olsder. *Dynamic noncooperative game theory*, volume 160 of *Mathematics in Science and Engineering*. Academic Press, 1999.

[9] R. Isaacs. *Differential games*. Wiley, New York, 1965.

[10] Steve Alpern and Shumel Gal. *The Theory of Search Games and Rendezvous*. Kluwer Academic Publishers, 2002. 336 pages.

[11] J. von Neuman and O. Morgenstein. *Theory of Games and Economic Behaviour*. Princeton University Press, Princeton, 1944.

[12] Ronald Fagin and Joseph Y. Halpern. Belief, awareness, and limited reasoning. *Artificial Intelligence*, 34(1):39–76, 1988.

[13] J. Y. Halpern, Y. Moses, and M. Y. Vardi. Algorithmic knowledge. In *Proc. of the 5th Conference on Theoretical Aspects of Reasoning about Knowledge (TARK'94)*, pages 255–266. Morgan Kaufmann, 1994.

[14] T. D. Parsons. *Theory and Applications of Graphs*, chapter Pursuit-evasion in a graph, pages 426–44. Springer-Verlag, 1976.

[15] Steven LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[16] Leonidas J. Guibas, Jean-Claude Latombe, Steven LaValle, David Lin, and Rajeev Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry and Applications*, 9(5):471–494, 1999.

[17] D. Pellier and H. Fiorino. Coordinated exploration of unknown labyrinthine environments applied to the pursuit evasion problem. In *Proceedings of the fourth international joint conference on Autonomous Agents and Multiagent Systems*, pages 895–902. ACM Press New York, NY, USA, 2005.

[18] Craig W. Reynolds. Steering behaviors for autonomous characters. In *Proc. of Game Developers Conference*, 1999.

[19] Ibrahim Volkan Isler. *Algorithms for Distributed and Mobile Sensing*. PhD thesis, University of Pennsylvania, 2004.