Coordinated Exploration of unknown labyrinthine environments applied to the Pusruite-Evasion problem

Damien Pellier, Humbert Fiorino

Coordinated exploration of unknown labyrinthine environments applied to the Pursuit-Evasion problem

Damien Pellier
Laboratoire Leibniz (CNRS - INPG - IMAG)
Equipe MAGMA
46, avenue Félix Viallet
F-38031, Grenoble
Damien.Pellier@imag.fr

Humbert Fiorino
Laboratoire Leibniz (CNRS - INPG - IMAG)
Equipe MAGMA
46, avenue Félix Viallet
F-38031, Grenoble
Humbert.Fiorino@imag.fr

ABSTRACT

This paper introduces a multi-robot cooperation approach to solve the "pursuit evasion" problem for mobile robots that have omni-directional vision sensors in unknown environments. The main characteristic of this approach is based on the robots cooperation by sharing knowledge and making them work as a team: a complete algorithm for computing robots motion strategy is presented as well as the deliberation protocol which distributes the exploration task among the team and takes the best possible outcome from the robots resources.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence-Coherence and coordination - Intelligent agents - Multiagent systems

General Terms

Algorithms

Keywords

Distributed robotics, coordination, pursuit-evasion problem, multi-agent systems

1. INTRODUCTION

The domain of cooperative robotics is acquiring a prominent interest in many applications such as drones or Unmanned Autonomous Vehicles formations [22], mobile robots carrying out transportation tasks [1], exploring the environment [11], interacting with people [6] etc. Indeed, this domain is well adapted to implement and validate cooperation approaches. To that end, many different approaches have been proposed ranging from reactive behaviors [2] to deliberative protocols [1].

In this paper, we consider the following problem: a group of robots have to explore a simulated labyrinthine environment in a way such that, if an intruder is hidden within, necessarily it would be found and could not escape. Such a problem has been already studied in mainly two different aspects.

On the one hand, the "prey predators" problem has been proposed for the first time by [3]: the prey and the predators (i.e. the *agents*) share a common environment which is represented by a mere grid. The hunt is simulated and the agents that are autonomous processes can move horizontally or vertically all the time. Quite obviously, the *goal* of the prey is to escape as long as possible whereas the predators have to capture the prey as fast as possible. Therefore, the "prey-predators" problem is an interesting testbed for competing agents and coordination protocols.

Indeed, because each agent does not perceive all the environment, and can have incomplete or inconsistent knowledge about the other agents, many challenging issues have to be tackled: which information has to be communicated? to whom? when? how can the predators constrain the prey movements? how can they elaborate a common strategy and behave as a *team*? otherwise, can the predators exclusively rely on reactive behaviors? which is the best approach in terms of performances, implementation, communication hazards (communication bottleneck, deadlocks etc.) ? Unfortunately, the environment considered in those experiments is too simple.

On the other hand, the "pursuit-evasion" problem is based on an environment made up of many obstacles. It is postulated that an intruder may be hidden within this environment, and a pursuer must flush it out of its hiding place. Therefore, a solution is a path ensuring that whatever movement is realized by the intruder, finally it will be uncovered by the pursuer. This problem has been tackled in many ways such as game theory [16], graph theory [4, 12, 17, 18] etc. As far as robotics are concerned, the "pursuit-evasion" problem has been introduced by [21]. Since then, several works have been undertaken: in [15, 7], robots fitted with one detection beam must keep watch on a grid environment with one exit; in [10, 9], two keepers move along the borders of a labyrinthine environment without isolated obstacles (i.e., that are not linked to a border) and have to keep in touch constantly with their sensors. The "pursuit-evasion" problem has also been looked at more generally in [5, 13, 14, 21]: this time, there are less restrictions on the environment (2D polygonal or curved environment) and the pursuers are fitted with omni-directional detection beams. More recently, [20] tackles the "pursuit-evasion" problem in unknown polygonal environment.

However, either the environment can be explored by only one pursuer and then the algorithm provides the path ensuring that the intruder (its velocity can be arbitrary high) will be uncovered, or, because of the environment topology, several pursuers are needed

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

and then the algorithm provides the path that has to be followed by *one* pursuer completed by other robots that must remain stationary in their assigned observation posts. This is due to the intruder's ability to move under cover from a hiding place to places already explored by the pursuer. The main inconvenient of this approach is that many robots which may help to explore the environment and hence their resources, are underemployed.

In order to avoid the drawbacks of the former approaches, we present a cooperation protocol which can be applied to the robotics that allows several pursuers to coordinate their exploration and jointly look after the intruder: they operate like a team and the number of pursuers is "minimized" as much as possible. In section 2, the pursuit evasion problem is defined. In section 3, a complete algorithm for several pursuers based on a cooperation protocol is presented.

2. PROBLEM DEFINITION

In this paper, we assume that the pursuers manoeuvre within a 2D simulated polygonal environment; their vision is omni-directional and they do not know the environment. A simple environment and a possible robot motion strategy is shown in figure 1.

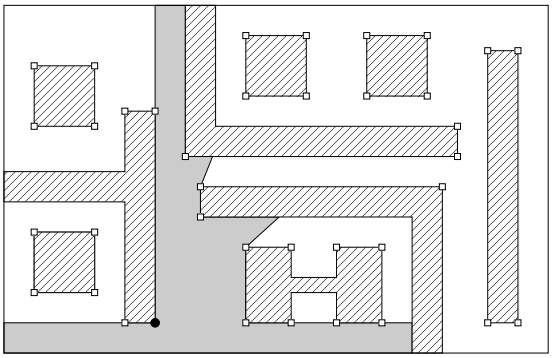

Figure 1: Example of labyrinthine environment: the black dot shows the agent position, the gray area the agent perception, the dashed areas the obstacles and the white squares the critical points of the environment .

The pursuers and the intruder are represented by points in the 2D Euclidean space. Let F be the free space (obviously, the pursuers and the intruder belong to F). Let $e(t) \in F$ be the position of the intruder at time $t \geq 0$. We suppose that

$$e : [0, \infty[\rightarrow F$$

is a continuous function and that the intruder is able to move as fast as it wants: its initial position $e(0)$ and its trajectory e are not known by the pursuers. Let $\gamma^i(t)$ be the position of the ith pursuer at time $t \geq 0$. $\gamma^i : [0, \infty[\rightarrow F$ is the continuous trajectory of the i^{th} pursuer. Let γ be the trajectories set of the N pursuers :

$$\gamma = \{\gamma^1, \ldots, \gamma^N\}$$

For all point $q \in F$, let $V(q)$ be the set of all visible points from q in F (i.e., all the segments joining q to a point of $V(q)$ strictly belong to F). The trajectory or "watch" γ is a solution if, for all continuous function $e : [0, \infty[\rightarrow F$, $t \in [0, \infty[$ such that $e(t) \in V(\gamma^i(t))$ (with $i \in \{1, \ldots, N\}$) exists. This implies that the intruder cannot escape; at a given time, necessarily, it will be uncovered.

The problem definition shows two of the main difficulties that have to be dealt in the pursuit evasion problem. The first one is to find a motion strategy $\gamma^i(t)$ with $i \in 1, \ldots, N$ such that $\gamma^i(t)$ must all the time consider all the possible moves of the intruder in order that the intruder cannot move under cover from hiding places to placed already explored by the pursuers. This difficulty is illustrated in figure 2. The second difficulty is to compute the minimum number of pursuers $H(F)$ needed to ensure that the intruder will be discovered. According to [8], in known environments, $H(F)$ computation is NP-Hard. It was proved that

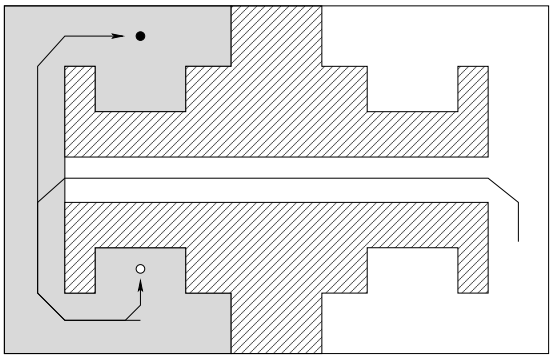

Figure 2: An intruder (white dot) can move under cover from a hiding place to places already explored (gray area) by the pursuer (black dot).

$$H(F) = O(\sqrt{h} + log\ n)$$

in the worst case in an environment F where n is the number of edges and h the number of simply connected areas of F (i.e, $log\ n$ pursuers to triangulate F and $\sqrt{h}$ pursuers to split F in simply connected areas). On the contrary, we assume that the environment is unknown. Therefore, we try first of all to solve the pursuit-evasion problem with one pursuer. In some circumstances that we will detail later, the pursuer cannot follow the exploration without assistance. Then, additional pursuers are added and we make them work as a team in order to use as best as possible the available resources and avoid to add pursuers unnecessarily.

3. PURSUIT-EVASION ALGORITHM

The task of the pursuers is to detect the intruder in the environment, but they cannot execute this task without knowing this environment. That is why, the pursuit algorithm interleaves two steps, an exploring one and a pursuing one. The exploration steps build a topological representation of F as a graph based on the possible moves of the pursuers. During the exploration process the status of the pursuit is also maintained for efficiency reasons. The pursuing steps use the representation of the environment gathered during the exploration steps so as to compute a motion strategy which guarantees that all the environment has been explored and the intruder did not escape. The pursuit algorithm is based on two graphs, the navigation graph G_n and the pursuit graph G_p.

3.1 Navigation Graph

We assume there are specific vertices of the free space F, called critical vertices that are significant in order to solve the pursuit-evasion problem. Those points are as follows: a vertex of F is considered as *critical* if and only if the angle formed by this adjacent edges is superior to π. Intuitively, if we could put a pursuer with an omni-directional vision on each critical vertex, there would have no place where the intruder could hide in F (see figure 3).

Hereafter, we assume that the pursuers are able to detect the critical vertices of the environment, and that they move from one critical vertex to another one. Some sensing capability is required, of

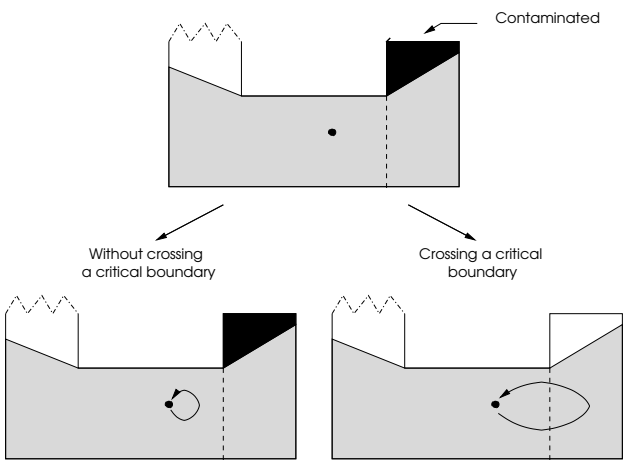

Figure 3: A critical event occurs when edges visibility changes. A critical point is a specific point of the critical boundary.

course, to solve the problem. The pursuer is equipped with a sensor which gives the location of discontinuities in depth information. It is assumed that the pursuers have a kind of edge detector that can detect each discontinuities, and returns their direction relative to the pursuer to compute the vertices angles of the environment.

Moreover, in order to encode the different moves that pursuers can execute into F, we describe F as a *navigation graph*

$$G_n = (N_n, E_n)$$

with N_n the set of critical vertices V_c of F and E_n the set of edges that represent visibility relations between critical vertices. Initially, the environment is entirely unknown and the visibility graph G_n is empty.

3.2 Pursuit Graph

To solve this problem, pursuers have to know the global state of the pursuit S_{gp} i.e. which parts of the environment represented by G_n are considered as CLEAR or CONTAMINATED (when an intruder can be hidden within) according to their current position $\gamma(t)$. Hence, a *pursuit state* S_p is made up of a list of critical vertices $V_c \in N_n$, each of these vertices being labeled CLEAR or CONTAMINATED.

Suppose a moment that the environment is completely known. A complet visibility graph is shown in figure 5.a. This graph corresponds to the environnement shows in figure 4. It is possible to compute a *pursuit graph* (see figure 5.b) that characterizes all the different pursuit states that can be reached from G_n. Each node of G_p is a pursuit state S_p and the edges represent the vertices where the pursuers must go to reach the next pursuit state. As we will present in the next section, the environment is split in subparts to reduce the problem complexity when a pursuer does not find a solution alone. Therefore, we define the global pursuit states S_{gp} as the union of the different pursuit state reached by each pursuer on each subpart of the environment.

The initial state of G_p is calculated according to the starting position p of the exploration (arbitrary chosen) and the pursuit global state to avoid the computation of unused nodes. For each critical vertex V_c of the considered node, the CLEAR label is given if and only if the vertex V_c is visible from p and the adjacent segments of V_c are entirely visible from p. For each possible transition $T = \{t_1, \ldots, t_n\}$ (where n is the number of critical vertices visible from p), a new node is generated by applying this procedure.

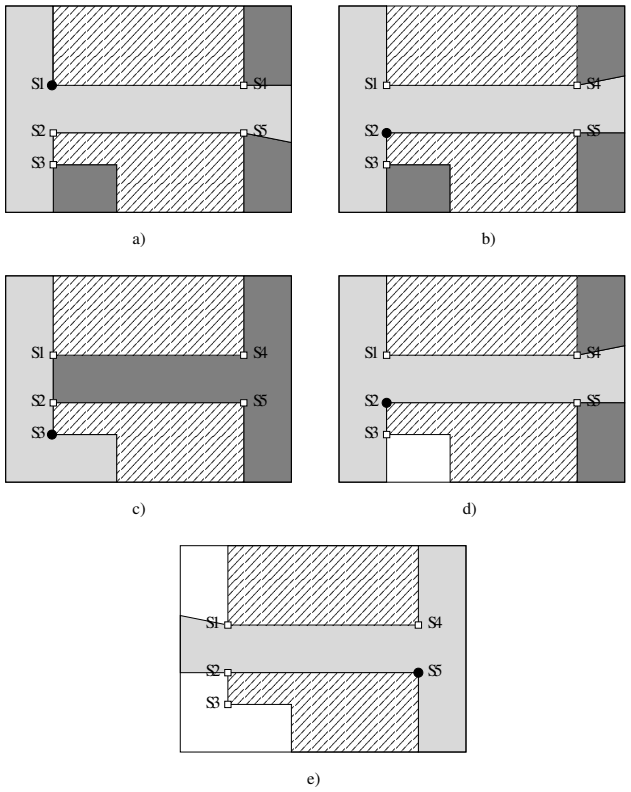

Figure 4: The agents are represented by black dots, the light gray areas show the robot perception, the white squares represent the critical points label from S_1 to S_5, the white areas show the decontaminated part of the environment whereas the dark areas show the contaminated ones.

However, when generating these new nodes, it must be verified that the former cleared critical vertices remain cleared in the following node. Therefore, the following condition must be assessed: i) for all cleared critical vertices in the current node, there do not exist a path from a contaminated vertex to one of these vertices; ii) if that path exists, the "threatened" vertices remain visible during the transition t_i. If those conditions are verified then the vertices remain cleared (otherwise they become contaminated). The transition is labeled with the distance previously calculated into G_n. This step is applied recursively on all new nodes. Thus, because all possible transitions are explored, it is guaranteed that, if a solution exists, this solution is found by the algorithm. And, a solution is found when a pursuit node S_p is obtained when building up G_p such that for all $V_c \in S_p$, V_c is labeled CLEAR.

3.3 Exploration and Pursuit Algorithms

In this section we present the exploration and the pursuit process that are interleaved. Initially, the navigation graph G_n is empty. Thus, the goal of the exploration step is to build a partial map of the environment on which the pursuit algorithm will be applied.

On the one hand, the exploration algorithm is as follows: The pursuer gets the critical vertices from its initial position and initializes G_n with this position (see algorithm 1). To be sure that an intruder will be discovered, the exploration algorithm must exhaustively cover the environment. Therefore, to ensure that no critical vertex will be missed by the exploration process, each node is marked when it is visited. At each step of the exploration, the

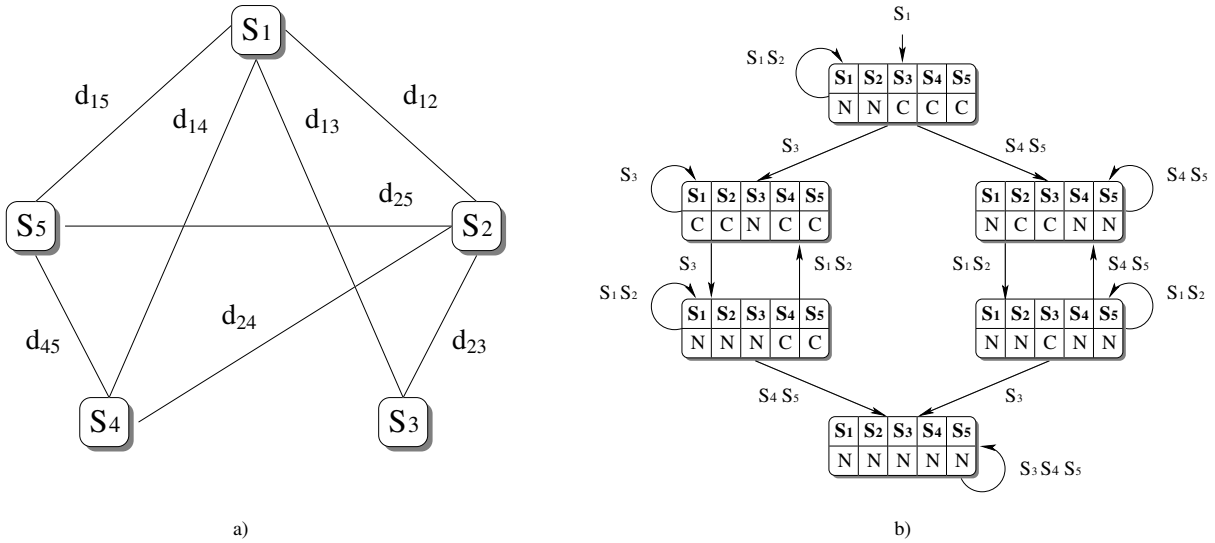

Figure 5: a) shows the navigation graph and b) the pursuit graph corresponding to figure 4: "N" means that the vertex is cleared; "C" means contaminated.

Algorithm 1: Explore((edb, p, γ_e)

Input: : - *edb the depth exploration bound.*
- *p the current position of the pursuer.*

Output:: - *p the current position updated.*
- γ_e *the exploration path.*

```
1  i := 0; γe := ∅; γe* := ∅; Nn* := Unexplored(Gn);
2  while i < edb and Nn* ≠ ∅ do
3      Nn* := Unexplored(Gn);
4      foreach Vc ∈ Nn* do
5          γe* := ComputeMotion(Gn, p, Vc);
6          ExecuteMotion(γe*);
7          Update(Gn, VisibleVerticesFrom(Vc));
8          γe := γe + γe*;
9          MarkExplored(Gn, Vc);
10         p := Vc;
11     end
12     i++ ;
13 end
```

pursuer explores the vertices previously discovered. Let N_n^* be the nodes of G_n that have not been visited at a given exploration step (i.e. they have been seen but not visited) and edb the depth exploration bound. Therefore, edb allows to set to what extent the exploration is interleaved with the pursuit.

For each $V_c \in N_n^*$ (i.e., the set of nodes not visited yet) the pursuer computes the shortest path γ_e^* to reach it. Once the pursuer reached it, it gets the list of the critical vertices visible from its new position: the navigation graph is updated with those vertices. Then, there is two possibilities: if one of these vertices is already in G_n, an edge is added between the current position node and this node; otherwise, if a new critical vertex has been discovered, it is added to the navigation graph as not visited and it is connected to the current position node. Finally, the current position node is marked visited. When all the node N_n^* have been explored, the depth of the exploration is incremented. The exploration algorithm ends when the depth exploration bound is reached or if there is no more nodes to explore. Then, the algorithm returns the exploration motion γ_e which is the concatenation of each γ_e* (see algorithm 1).

On the other hand, the pursuit algorithm is as follows (see algorithm 2): the pursuer is now able to seek the intruder in the previously explored part of the environment. First of all, the pursuit process updates the pursuit graph G_p according to the exploration path γ_e to know its current pursuit state. Indeed, for each vertex crossed by γ_e (recall that we assume that the pursuers move from critical vertex to critical vertex), a pursuit state is built. Then the global pursuit state S_{gp} is updated. If the current state is a final state (i.e., for all $V_c \in S_p$, V_c is labeled CLEAR), then a partial solution is found (i.e. a solution for this part of the environment) and the exploration process can be resumed. Practically, finding a partial solution by following the exploration path is a particular case.

Generally, the pursuer must build the pursuit graph G_p associated to this subpart of the environment as explained in section 3.2. If it exists at least one final node in G_p then a partial solution is found: the pursuer computes the pursuit motion from G_p by applying the Dijkstra algorithm and the global pursuit state S_{gp} is updated. If no final state corresponding to the considered subpart of the environment is found, at least one more pursuer is needed. This additional pursuer is added in order to reduce the complexity of the explored environment by splitting it (SPLITENVIRONMENT()). First of all, it chooses to withdraw a node $n_c \in G_n$ such that the degree of n_c is the highest of G_n.

Then, the pursuer removes from the navigation graph G_n n_c but also all the visible nodes from n_c. The pursuer knows that it needs a guard staying on this node to split the environment into parts that are represented by the different connected components of the navigation graph (denoted CCG_n in algorithm 2). Then, each part of the environment can be explored independently. Indeed, leaving a stationary pursuer on the removed node n_c guarantees that an intruder cannot move from a part of the environment to another one. If this method fails to share the environment into independent components, then we obtain one connected component but the guard is still needed on n_c: the environment that remains to be explored (the obtained connected component) is reduced. Then, the pursuit and the splitting procedures are resumed recursively on each obtained component (eventually the only one) until the pursuit-evasion prob-

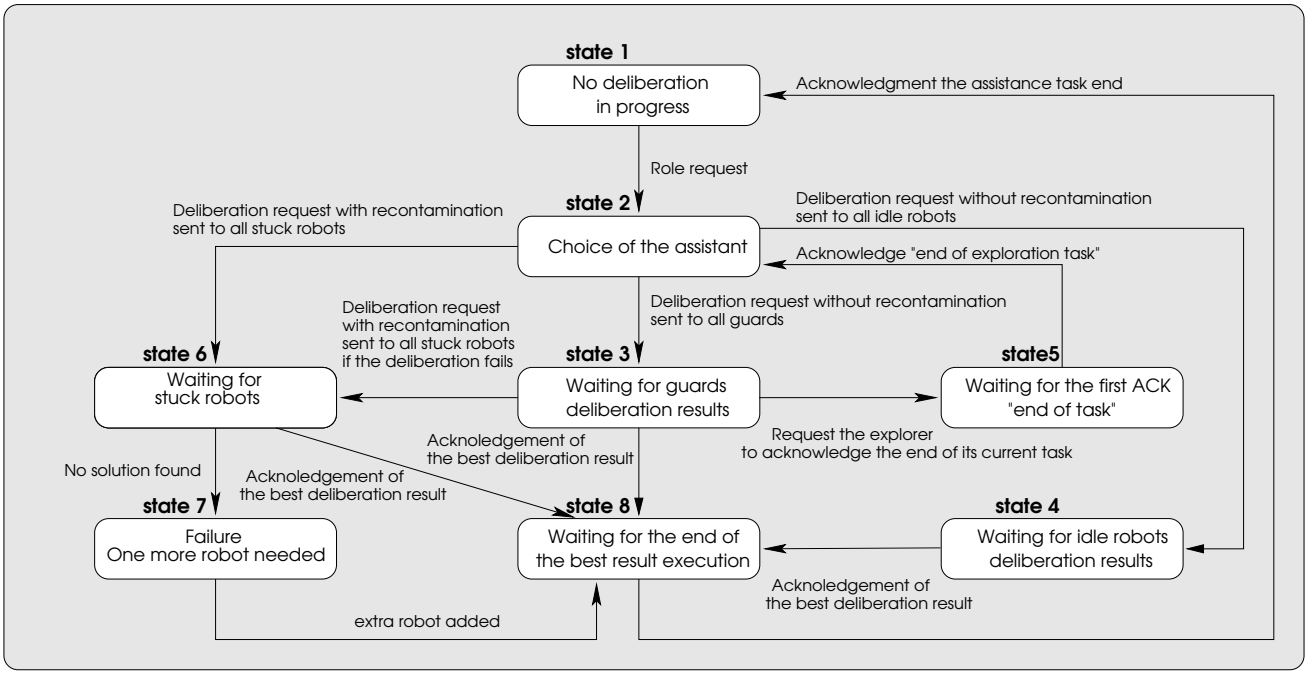

Figure 6: Deliberation protocol: side of the assisted robot.

lem can be solved independently on those components.

Note that the exploration (EXPLORE()) and DELIBERATION() ares sub-procedures of the pursuit. DELIBERATION() is explained in the next section.

3.4 Cooperation: Delegation Protocol

The cooperation is implemented through the delegation protocol: first, a pursuer tries to solve the pursuit-evasion problem alone. The figure 6 shows the delegation protocol side of the assisted robot and the figure 7 shows the delegation protocol side of the assistant robot. If it fails, other teammates are added to provide assistance. Then the team works as follows: some pursuers are stationary in order to split the environment while the others explore and seek the intruder within the induced subparts of the environment. When a subpart is cleared, the explorer and the stationary pursuers are reallocated to another part of the environment.

This protocol is based on four different "roles" that can be played by the pursuers while exploring the environment (these roles change during the labyrinth exploration):

- *explorer*: the pursuer explores an assigned part of the environment;
- *guard*: the pursuer is used to share the environment into parts so that an intruder cannot move from one part to another one;
- *idle pursuer*: the pursuer is idle and it can be required for a new task (i.e. it ended previous assigned tasks);
- *stuck pursuer*: this pursuer needs help to continue its exploration. It cannot move without risking a recontamination of previously explored areas.

Therefore, the first step of the deliberation protocol is to collect the pursuers roles (see figure 6, state 1). A stuck pursuer runs this protocol in order to estimate the team capabilities and to request assistance. When this pursuer receives all the teammates roles, it

Algorithm 2: Pursuit(G_n, S_{gp}, edb)

Input: : - G_n *the part of the environment to guard.*
- S_{gp} *the global status of the pursuit.*
- edb *the depth of exploration.*

```
1  G_p := ∅; γ_e := ∅; γ_p := ∅; p := ∅; n_c := ∅;
2  if G_n == ∅ then
3  |  p := InitialPosition();
4  |  Update(G_n, VisibleVerticesFrom(p));
5  end
6  while ∃ V_c ∈ G_n | V_c ∈ Unexplored(G_n) do
7  |  Explore(G_n, edb, p, γ_e);
8  |  S_gp := UpdatePursuitState(G_n, γ_e);
9  |  if ∀ V_c ∈ S_p, V_c == Clear then
10 |  |  return ;
11 |  end
12 |  else
13 |  |  ComputePursuitGraph(G_n, p, S_gp, G_p);
14 |  |  if ∃S_p ∈ G_p | ∀V_c ∈ S_p, V_c == Clear then
15 |  |  |  ComputePursuitGraph(G_n, p, S_gp, G_p);
16 |  |  |  ExecuteMotion(γ_p);
17 |  |  |  S_gp := UpdatePursuitState(G_n, γ_p);
18 |  |  end
19 |  |  else
20 |  |  |  SplitEnvironment(G_n, CCG_n, n_c);
21 |  |  |  Deliberation(n_c);
22 |  |  |  foreach cc ∈ CCG_n do
23 |  |  |  |  Pursuit(cc, S_gp, edb);
24 |  |  |  end
25 |  |  end
26 |  end
27 end
```

can assess which pursuers can help it (see figure 6, state 2). Three cases are possible:

- if at least one pursuer is idle, the stuck pursuer informs the idle pursuers of its need for assistance because they can be immediately reallocated without any risk of recontamination (see figure 6, state 4);
- if all pursuers are busy (i.e, guard or explorer) but there is at least one guard, the stuck pursuer sends to the guards the visible vertices it would like to be watched and the global state of the team (see figure 6, state 3). To obtain this global state, the stuck pursuer requests the local state of each teammate (i.e. the current node of their pursuit graph) and merges all these states. With this information each guard is able to compute all the possible moves it can do in order to assist the stuck pursuer.

  They must solve two different kinds of problems: ensure the division of the environment and answer to the assistance requests. More precisely, two possibilities must be considered:

  - the guard finds that all the visible vertices from its position are cleared and thus, it becomes an idle pursuer. Indeed, if this condition is verified, the guard is sure that its surveillance task is finished. It has no reason to watch vertices that are cleared because the algorithm for one pursuer ensures that no intruder can be located into a cleared part of the environment. Therefore, it can leave its current position and calculate the best path to reach a vertex from which it can carry out the requested assistance task;
  - The guard finds that at least one visible vertex from its position is contaminated. It cannot be reallocated to explore a sub-part of the environment. But, in some cases, it can find a path that drives it to a vertex from which it can carry out the requested assistance task *and* fulfill its current guard task;
- if all pursuers are stuck (see figure 6, state 6), the last stuck pursuer (that finds that all its teammate are also stuck) builds the global state and broadcasts this information with its need for assistance. In order to avoid a deadlock, at least one stuck pursuer must give up its assistance request and its planned exploration task in order to provide assistance to one of its teammate. Thus, the areas it had previously explored must be considered as re-contaminated.

  Then, this pursuer becomes idle and provides assistance to the last stuck pursuer. To that end, it calculates the cost of its assistance in terms of trajectory distance and number of re-contaminated critical vertices. Furthermore, the protocol must ensure that the team will not enter into an infinite loop. For instance, assume that two areas A and B have to be cleared: if clearing the A area implies to contaminate the B area, it must be ensured that clearing B does not imply to re-contaminate A. The infinite loop is avoided by recording information about the previously cleared and re-contaminated areas. If such a loop is detected, then at least one more pursuer is needed.

After this step, each pursuer delivers its answer to the assistance request (i.e. "refused" or "accepted" request and, in the former case, the cost of the assistance task) to the stuck pursuer that chooses the best solution according to distance and number of re-contaminated vertices criteria.

However, the cost of the assistance task can take into account the pursuers' specificities in terms of available resources. That is to say, their ability to answer to assistance requests does not only depend on their roles (guard, explorer, idle or stuck pursuer) but also on their available energy etc. At this step two cases are possible:

- The deliberation converges toward one or more solutions. The best solution is chosen according to the previously introduced criteria. The chosen pursuer receives an acknowledgment that informs it to start the assistance task. The other pursuers receive a no-acknowledgment to indicate that their solutions were not accepted.

  Finally, when the assistant pursuer finishes its task, it delivers to the stuck pursuer an "end of task" message. The former stuck pursuer can resume its exploration task and becomes an explorer;
- The global deliberation fails. No solution is available:
  - if no solution was found even with recontamination (a stuck pursuer abandons its exploration task), the team fails and cannot solve the problem with the number of pursuer involved into the environment. Necessarily, at least one more pursuer is needed. This pursuer is added to the team and the deliberation protocol is resume in order to find the minimal number of pursuers needed to converge toward a solution (see figure 6, state 7);
  - if at least one pursuer is an explorer (its exploration task is in progress but sooner or later it will become idle or stuck), potentially it can decontaminate an area and thus allow guards to become idle pursuers. Thus, the failure is not effective until there is no more explorer.

    Anyway, they must inform the stuck pursuer of the end of their current tasks. Indeed, the global state may have evolved favorably. A new round of deliberation is initiated (in the worst case, the number of additional deliberation rounds is equal to the number of explorers found during the first round) and can lead to a solution (see figure 6, state 5);
  - if the team is only made up of guards and stuck pursuers and no solution was found by the guards, the last stuck pursuer broadcasts to the other stuck pursuers a deliberation request. Thus, the protocol returns to the state 6 previously described.

4. REMARKS AND DISCUSSION

Tasks parallelization. When a pursuer is stuck, it tries to split the environment into smaller parts to solve the problem. The current node where the pursuer is located and all its adjacent nodes are removed from the navigation graph. This operation can lead to split the navigation graph in many not connected components. If a guard stays at the current pursuer location, the pursuer team can choose to parallelize the exploration by introducing new pursers or available pursuers to explore each parts of the environments (i.e. the navigation graph components obtained after the cut). The figure 8 shows a simple example where the parallelization is possible between two pursuers.

Guard coordination. The role of the guards is to stay at a location to keep a look to potential critical vertices of the environment. Consider a guard that must stay at a location to

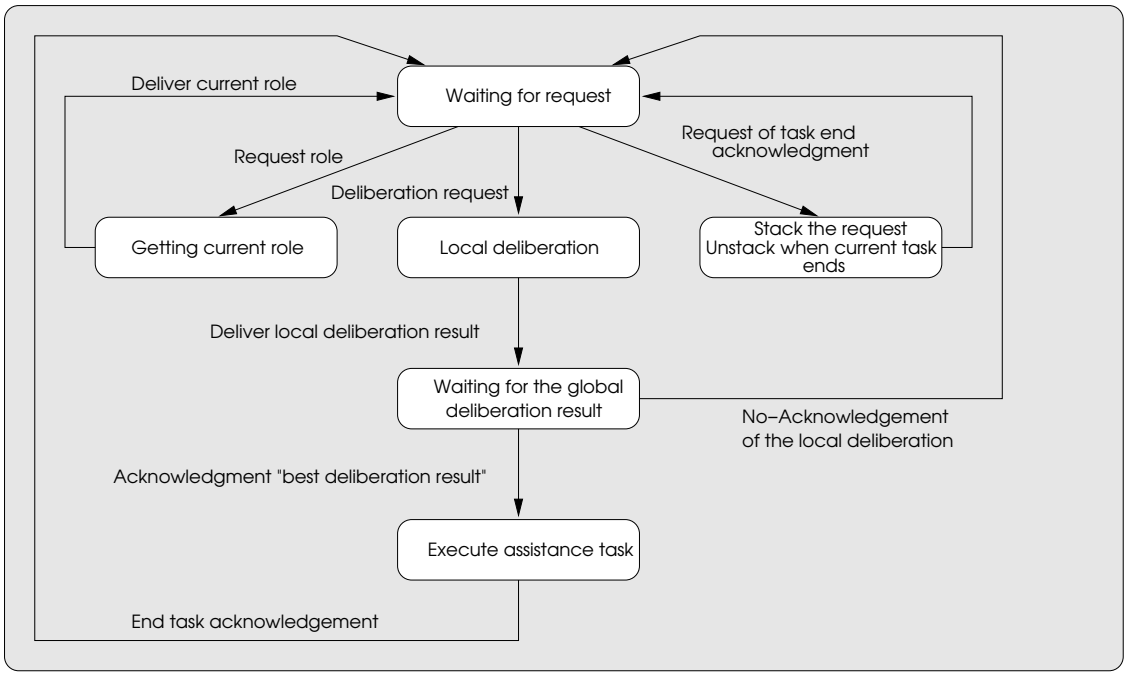

Figure 7: Deliberation protocol: side of the assistant robot.

monitor some critical vertices. When a pursuer cleans a part of the environment, the vertices that must be monitored by the guard can become clean. These vertices must no more be monitored by the guard. Therefore, the guard can move to help another stuck pursuer but must always monitor the contaminated vertices. This case is shown by the simple example in figure 9.

Implementation. The implementation was carried out with JAVA language on the multi-agent platform JADE. The implementation allows to distribut agents on many hosts and shows the motion strategies of the agents on a graphical console.

5. CONCLUSION

As presented, this cooperation protocol is adapted for an unknown environment. The construction of the graphs can be done iteratively: each time new critical vertices are discovered during the exploration, they are added to the navigation and pursuit graphs respectively. When a pursuer is stuck, the team can still run a deliberation about the known environment. Further deliberations are undertaken as the known environment grows. Unlike [19] where the algorithm provided works for only one pursuer and needs two successive steps (environment cartography and then search for an intruder), the simultaneous discovery and exploration of the environment can lead to decrease the covered distances.

Furthermore, recall that when a pursuer is stuck it tries to split the environment into smaller parts to solve the problem. When a splitting of the environment occurs, the chosen cut node and all its adjacent nodes are removed from the navigation graph. Indeed, those nodes can be removed of the pursuit process because a guard pursuer stays at the cut node position. Thus, if an intruder tries to go over these locations it will be discovered. Of course, the nodes not yet visited cannot be removed from the navigation graph without risk of forgetting parts of the environment still unexplored. The splitting of the environment is run recursively on each obtained components until the component can be cleared by a single pursuer.

The splitting of the environment leads to distribute the algorithm complexity. Suppose a moment that the whole environment is known. The construction of the pursuit graph is $O(2^n)$ with n the number of critical vertices of the environment. Consider now that the pursuit algorithm split the environment into k parts, the complexity is "reduced" to $O(k2^{n/k})$. Moreover, the complexity is tractable because practically the number of critical vertices remains small. This is a fortiori verified when exploring and constructing the graphs simultaneously. The protocol complexity (i.e. corresponding to the number of exchanged messages) is $O(n^2)$ in the worst case where n is the number of pursuers.

Finally, in comparison to previous works such as [14], this cooperation protocol allows to minimize the number of pursuers by making them work as a team, distributing and sharing the exploration among all the pursuers. It is based on a "least commitment" strategy: indeed, extra pursuers are added if and only if cooperation fails and assistance is successively sought among the least constrained pursuers (idle, guard, explorer and then stuck pursuers). This leads to a better use of the pursuers' resources, and an increase of performances and robustness due to the parallelization.

6. REFERENCES

[1] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi robot cooperation in the Martha project. In *IEEE Robotics and Automation Magazine*, volume 5(1), 1998.

[2] T. Balch and C. Arkin. Behavior-based formation control for multirobot teams. In *IEEE Transaction on Robotics and Automation*, volume 14(6), pages 926–939, 1998.

[3] M. Benda, V. Jagannathan, and R. Dodhiawala. On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS-G2010-28, Boeing Advanced Technology Center, Boeing Computing Services, July 1986.

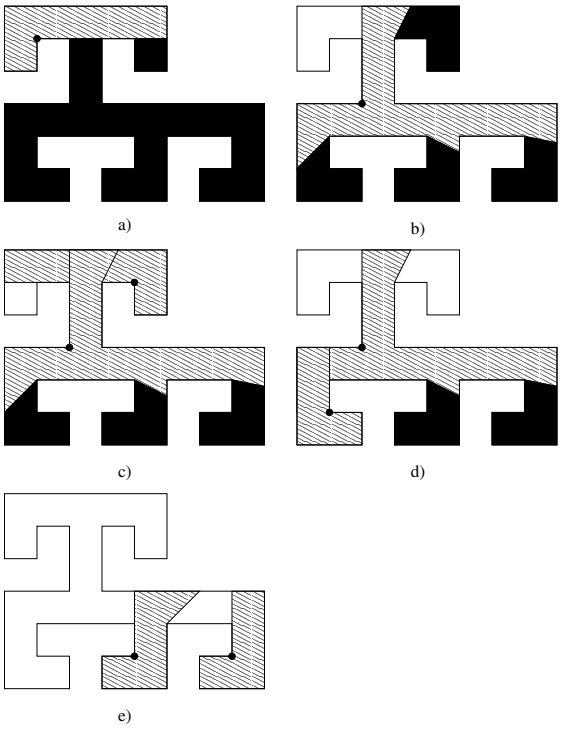

Figure 8: Two robots with a coordination and parallelization tasks.

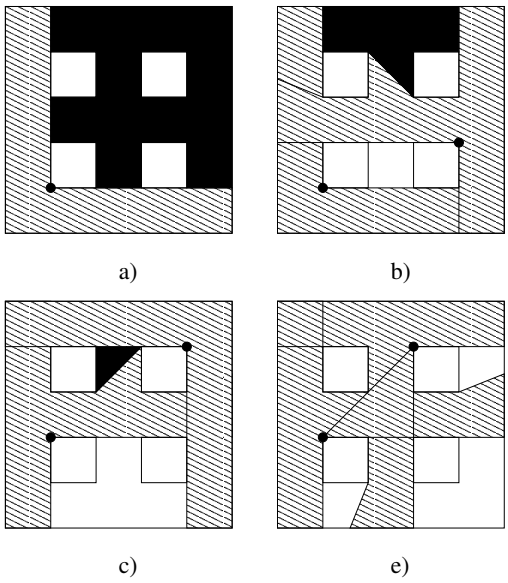

Figure 9: A deliberation example between a stuck robot and a guard.

[4] D. Bienstock and P. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12:239–245, 1991.

[5] D. Crass, I. Suzuki, and M. Yamashita. Searching for a mobile intruder in a corridor - the open edge variant of the polygon search problem. *International Journal of Computational Geometry and Applications*, 5(4):397–412, 1995.

[6] A. Drogoul and S. Picault. "the MICRobES project, an experimental approach towards open collective robotics". In *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS'2000)*, 2000.

[7] B. Gerkey, S. Thrun, and G. Gordon. Visibility-based pursuit-evasion with limited field of view. In *Proceedings of the National Conference on Artificial Intelligence (AAAI 2004)*, San Jose, California, 2004.

[8] L. Guibas, J. C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry and Applications*, 9:471–493, 1999.

[9] P. C. Heffernan. An optimal algorithm for the two guards problem. In *Proceedings of ACM Symposium on Computational Geometry*, pages 348–358, 1993.

[10] C. Icking and R. Klein. The two guards problem. In *Proceedings of ACM Symposium on Computational Geometry*, pages 166–175, 1991.

[11] S. Lang and B. Y. Chee. Coordination of behaviours for mobile robot floor cleaning. In *Proc. IEEE International Conference on Intelligent Robots and Systems*, pages 1236–1241, 1998.

[12] A. S. Lapaugh. Recontamination does not help to seach a graph. *J. ACM*, 40(2):224–245, April 1993.

[13] S. M. LaValle and J. Hinrichnsen. Visibility-based pursuit-evasion: The case of curved environments. In *Proc. IEEE International Conference on Robotics and Automation*, 1999.

[14] S. M. LaValle, D. Lin, L. J. Guibas, J. C. Latombe, and R. Motwani. Finding an unpredictable target in workspace with obstacles. In *Proc. IEEE International Conference on Robotics and Automation*, pages 737–742, 1997.

[15] J. H. Lee, S. M. Park, and K. Y. Chwa. Searching a polygonal room with a door by 1-searcher. *International Journal of Computational Geometry and Applications*, 10:201–220, 2000.

[16] R. Levy and J. S. Rosenschein. A game theoretic approach to distributed artificial intelligence and the pursuit problem. *SIGOIS Bull.*, 13(3):11, 1992.

[17] B. Monien and I. H. SudBorough. Min cut is NP-complete for edge weighted graphs. *Theoretical Computer Science*, 58:209–229, 1988.

[18] T. D. Parsons. Poursuit-evasion in a graph. In *Theory and Application of Graphs*, pages 426–441, Springer-Verlag, Berlin, 1976.

[19] S. Rajko and M. LaValle. A pursuit-evasion BUG algorithm. In *Proceedings IEEE International Conference on Robotics and Automation*, 2001.

[20] S. Sachs, S. LaValle, and S. Rajko. Visibility-based pursuit-evasion in an unknown planar environment. *The International Journal of Robotics Research*, 23(1):3–26, 2004.

[21] I. Suzuki and M. Yamashita. Searching for mobile intruder in a polygonal region. *SIAM Journal on computing 21*, 21(5):863–888, 1992.

[22] R. Vidal, S. Rashid, C. Sharp, O. Shakernia, J. Kim, and S. Satry. Pursuit-evasion games with unmanned ground and aerial vehicles. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, 2001.