A MULTI-AGENT APPROACH TO ENVIRONMENT EXPLORATION

DARIO MAIO *
e-mail: dmaio@deis.unibo.it

and

STEFANO RIZZI *
e-mail: srizzi@deis.unibo.it

ABSTRACT

Exploration is a central issue for autonomous agents which must carry out navigation tasks in environments of which a description is not known a priori. In our approach the environment is described, from a symbolic point of view, by means of a graph; clustering techniques allow for further levels of abstraction to be defined, leading to a multi-layered representation. In this work we propose an unsupervised exploration algorithm in which several agents cooperate to acquire knowledge of the environment at the different abstraction levels. All agents are equal and pursue the same local exploration strategy; nevertheless, the existence of multiple levels of abstraction in the environment representation allows for the agents' behaviour to differ. Agents carry out exploration at different abstraction levels, aimed at reproducing an ideal exploration profile; each agent dynamically selects its exploration level, based on the current demand. Inter-agent communication allows for the agents to share their knowledge and to record acquaintances of the other agents. A communication protocol for organizing teams of agents is provided.

Keywords: Autonomous Agents, Cooperation, Distributed Artificial Intelligence, Environment Exploration, Layered Knowledge Representation, Multi-Agent Systems

1. Introduction

Autonomous agents are mobile versatile machines capable of interacting coherently with an environment and executing a variety of tasks in unpredictable conditions [7, 20]. Most activities for an autonomous agent involve planning the cheapest path which allows for one or more (possibly inter-related) goals to be achieved while avoiding collisions with obstacles, other agents or people; this navigation capability necessarily relies on a topological and metric description of the environment.

In our work we consider the case where the agent is given no a priori knowledge, so that it must learn the description of the environment on-line by exploring

* DEIS - Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy

it and interpreting sensor data. However, we assume that some meta-knowledge concerning the type of environment to be explored is available. Firstly, we assume that the descriptions of typical sensory patterns present in the environment are given; the selection of patterns corresponding to distinctive or significant categories of objects and places enables *landmarks* to be recognized, when in view, through a sensor-based classification algorithm [4, 13, 21, 10, 15]. Examples of landmarks are computers and telephones in office environments, medical equipment and receptions in hospital environments. Secondly, we assume that characterization of semantically significant *clusters* of objects or places is possible; agents recognize cluster borders by sensing the passageways between adjacent clusters. Example of clusters in office environments are rooms and floors, identified by recognizing doors and stairs, respectively.

Based on these assumptions, we proposed in [5, 16] a multi-layered architecture for representing the environmental knowledge to be used by an autonomous agent for navigation. In this architecture the environment is described, from a symbolic point of view, by means of a hierarchy of graphs defined by applying clustering techniques, starting from a graph of landmarks and inter-landmark paths (*routes*). In section 2. the concepts and formalisms necessary in the context of this paper are outlined, and the impact of layering on navigation-oriented applications is discussed.

The lack of a priori knowledge of the environment makes the problem of on-line exploration more relevant. In order to be ready to carry out navigation tasks as soon as possible, the agents should rapidly acquire a topological and metric description of the whole environment; an agent which knows the description of a single room in great detail will be less useful, for most tasks, than an agent which knows less about each room but has a general picture of the arrangement of the rooms in departments and floors. In section 3. we show how the existence of multiple levels of abstraction in the environment representation allows for different exploration profiles to be defined.

In [17] we proposed an algorithm for supervised multi-agent exploration, where the supervisor dynamically assigns each agent the task of exploring the environment at a specific abstraction level, and coordinates the agents assigned to the same level. Since the supervisor has, at any time, an exact picture of the exploration progress, the supervised architecture allows for an accurate scheduling of the resources. On the other hand, the supervised architecture is based on a point-to-point communication model which is not always feasible and in any case leads to high communication costs. Besides, since knowledge of the environment is stored in the supervisor, the existence of several agents is not exploited to achieve fault tolerance.

In [19] we proposed a multi-agent approach to exploration, in which several agents cooperate to acquire knowledge of the environment at the different abstraction levels; no external supervisor/coordinator is required. In this work we refine and extend the unsupervised approach to exploration. All the agents employed are equal and pursue the same local exploration strategy, inspired by Tremaux's graph-exploration algorithm. On the other hand, the hierarchical representation of the environment enables the agents to diversify their behaviour by committing themselves to exploration at different abstraction levels, aimed at reproducing an ideal exploration profile.

Each agent has an agenda, used for keeping track of the places seen (by the agent

itself or by other agents) but not explored. When the graph-exploration algorithm cannot be applied, due for instance to a one-way route, the agenda is first consulted locally (within a neighbourhood of the place where the agent currently is), then vertically (within a neighbourhood distributed on the higher abstraction levels), and finally globally (throughout the whole map).

The agents communicate by broadcasting messages; a message sent from an agent is received only by the agents who are currently placed within a circular area centred in the sender. Messages are aimed at acquaintanceship: to this end, each agent keeps the other agents as best informed as possible about where it is, what it is doing and where it is about to go. Messages are also aimed at knowledge sharing: fault tolerance is highly improved by having each piece of knowledge shared by several agents.

A mechanism of team formation is provided. When an agent discovers a new cluster, it may call for other agents to come and explore that cluster. By comparing the costs paid by the agents who answered the call to interrupt their activity and move to the new place, the caller forms an exploration team.

In section 4. the main features of the exploration script are outlined, focusing in particular on communication protocols, team formation and knowledge sharing. In section 5., performance of the exploration algorithm is evaluated in terms of efficiency, adherence to an optimal exploration profile, communication overhead and fault tolerance. In section 6. we discuss to what extent our algorithm could be effectively employed to carry out exploration of large information spaces such as the world wide web. The complete exploration algorithm is outlined in the Appendix.

2. Knowledge architecture

The representation adopted for the environmental knowledge has a critical role in making the formulation of queries about the objects in the environment and the formulation of planning problems more flexible, and in simplifying their resolution. The architecture for knowledge representation that this work is based on, is specifically oriented to navigation in structured environments. We say an environment is *structured* if a number of categories of objects and places that can be encountered in it are described a priori. We call *landmarks* the objects and places belonging to a subset of categories which are regarded as distinctive or significant.

According to many cognitive scientists, a cognitive map is organized into successive layers at different abstraction levels [11]. The architecture we propose is organized in *knowledge layers* determined by the structure of the environment and by the tasks which must be carried out. Each layer can be thought of as a view of the environment at a specific abstraction level; it includes only those details of the environment which are significant for a specific family of tasks or sub-tasks, and represents them in the most suitable formalism [16].

A layered representation of the environment is semantically richer than a "flat" representation. Consider, for instance, a consultant system for planning visits to a city or a museum. In these applications, the language for user-machine interaction should allow for constraints to be stated as precisely as possible ("one-hour shopping downtown (consider the shop hours), walk in the park before sunset, be back at

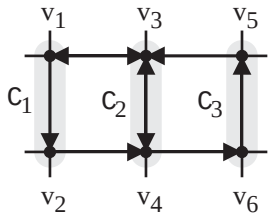


Figure 1: A clustering on a simple directed graph (arcs are shown by arrows).

airport by 19.00"). These natural-language requirements are formulated at different levels of abstraction, and correspond to formal constraints for path planning to be expressed on different knowledge layers.

The acquisition of the environment description takes place in most cases analogically, for instance from a set of sensor measures or a map. This view of the environment can hardly be exploited directly to carry out complex tasks, so knowledge must be reorganized and interpreted by abstracting one or more layers, each suitable for a specific task, from the low-level analogical description. Each of these layers may in turn generate other layers for other tasks, by means of a procedure of progressive abstraction which creates a taxonomy of layers. Three abstraction primitives can be used to derive a new layer from an existing one: classification, aggregation, generalization.

In this section we introduce the restricted formalism needed in this paper, which concerns representation of layers abstracted by aggregation only.

Let $L^{(0)}$ and $R^{(0)}$ be, respectively, the sets of landmarks and directed inter-landmark paths (*routes*) experienced at a given time. We define as *symbolic layer* the (weakly connected) directed graph $\mathcal{L}^{(0)} = (L^{(0)}, R^{(0)})$; each arc is labelled with the *cost* paid when covering the corresponding route. In this work we will assume that the cost of a route measures the physical distance covered by an agent which follows that route.

Given a directed graph $\mathcal{G} = (V, A)$, with V a set of vertices and A a set of arcs, we denominate with *clustering* a partition of the vertices and arcs of $\mathcal{G}$ into a set of clusters and a set of bridges. A *cluster* is a connected sub-graph of $\mathcal{G}$. The *bridge* between two clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ is the set of the arcs of $\mathcal{G}$ which connect a vertex of $\mathcal{C}_i$ to a vertex of $\mathcal{C}_j$. All clusters and bridges are disjointed.

Consider for instance the simple graph in Figure 1, defined as

$$\begin{aligned}\mathcal{G} \;=\; & (\{v_1, v_2, v_3, v_4, v_5, v_6\},\\ & \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_1), (v_3, v_4), (v_4, v_3), (v_4, v_6), (v_5, v_3), (v_6, v_5)\})\end{aligned}$$

A possible clustering is defined by the three clusters

$$\begin{aligned}\mathcal{C}_1 &= (\{v_1, v_2\}, \{(v_1, v_2)\}),\\ \mathcal{C}_2 &= (\{v_3, v_4\}, \{(v_3, v_4), (v_4, v_3)\}),\\ \mathcal{C}_3 &= (\{v_5, v_6\}, \{(v_6, v_5)\})\end{aligned}$$

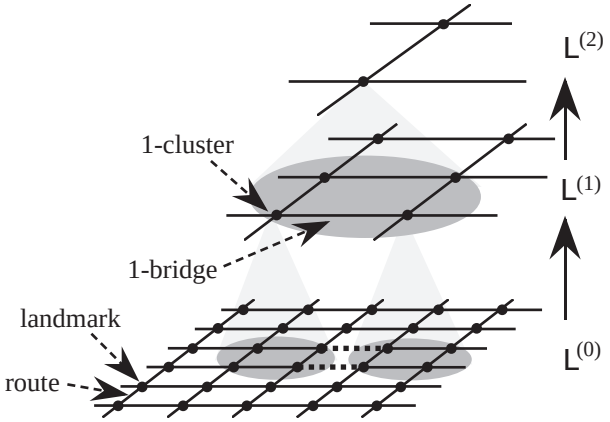


Figure 2: Definition of clustered layers. The two grey circles at the bottom are subgraphs of the symbolic layer, represented in the 1-clustered layer as vertices (1-clusters). The 1-bridge connecting the two 1-clusters corresponds, in $\mathcal{L}^{(0)}$, to the set of the two dashed routes.

and by the four bridges

$$\begin{aligned}(\mathcal{C}_1,\mathcal{C}_2) &= \{(v1,v3),(v2,v4)\},\\ (\mathcal{C}_2,\mathcal{C}_1) &= \{(v3,v1)\},\\ (\mathcal{C}_2,\mathcal{C}_3) &= \{(v4,v6)\},\\ (\mathcal{C}_3,\mathcal{C}_2) &= \{(v5,v3)\}\end{aligned}$$

We call *1-clusters* and *1-bridges* the clusters and bridges determined by clustering the symbolic layer $\mathcal{L}^{(0)}$. The directed graph whose vertices and arcs correspond, respectively, to the 1-clusters and the 1-bridges is called the *1-clustered layer* and denoted with $\mathcal{L}^{(1)}$. The 1-clustered layer defined by the clustering in Figure 1 is the graph

$$\mathcal{L}^{(1)} = (\{\mathcal{C}_1,\mathcal{C}_2,\mathcal{C}_3\},\{(\mathcal{C}_1,\mathcal{C}_2),(\mathcal{C}_2,\mathcal{C}_1),(\mathcal{C}_2,\mathcal{C}_3),(\mathcal{C}_3,\mathcal{C}_2)\})$$

The 1-clustered layer may in turn be clustered, generating a 2-clustered layer, and so on; a hierarchical clustering can thus be progressively defined, producing a hierarchy of graphs whose root is $\mathcal{L}^{(0)}$ (see Figure 2). In general, we name *k-clustered layer* and denote with $\mathcal{L}^{(k)}$ ($k = 1,..n$, where n is the maximum clustering level) the graph obtained by applying clustering k times, starting from the symbolic layer. The clusters and bridges of a k-clustered layer are called *k-clusters* and *k-bridges*, respectively (landmarks and routes may be seen as 0-clusters and 0-bridges, respectively). We call *cardinality* of a k-cluster the number of $(k-1)$-clusters it contains; the w-cluster which includes a k-cluster or a k-bridge ($0 \leq k < w \leq n$) is said to be its *ancestor* at level w. We will assume that the n-clustered layer contains exactly one n-cluster, which represents the whole environment.

Figure 3 shows an example of how a four-level hierarchical clustering (maximum clustering level $n = 3$) can be applied to an office environment.

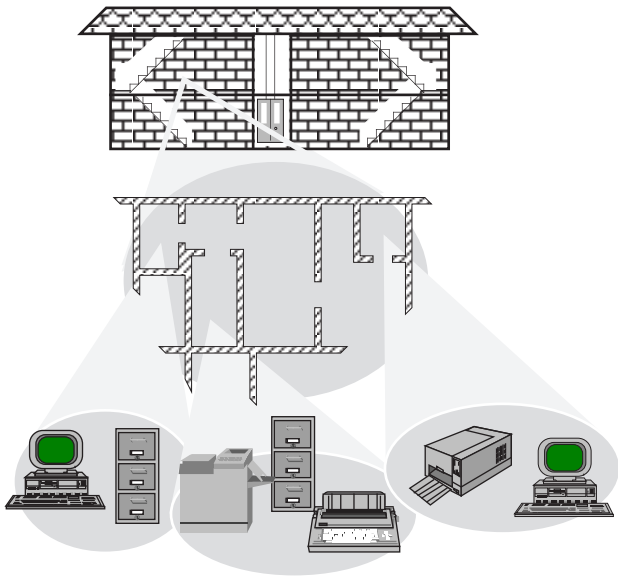


Figure 3: Clustering of an office environment. Landmarks in the symbolic layer correspond to useful objects, and are grouped in 1-clusters corresponding to rooms. Rooms in the 1-clustered layer, in turn, are grouped into floors which form the 2-clustered layer. The 3-clustered layer describes the whole building as a set of floors.

2.1. Applications in the field of information systems

In this section we discuss some applications, outside the robotics field, which could benefit from adopting the layered knowledge architecture outlined above. Basically, these applications share four characteristics:

(i) Knowledge consists of "places" and associations between them, so that it can be modelled by a connected graph. Each place may be a physical object or location, as well as a concept or a piece of information.

(ii) A cost function can be defined on the set of inter-place associations.

(iii) Physical or conceptual navigation between places is possible; the cost of navigating between two places is equal to the sum of the costs of the associations included in the path followed.

(iv) A hierarchical clustering may be defined on the graph of places. Each cluster must be a connected graph; the semantics of a cluster may be that of a place, at a higher abstraction level, which aggregates or includes a set of adjacent places at the level below; or that of an abstract concept which summarizes a set of associated concepts at the level below.

All the applications which require modelling of a structured environment meet these requisites. In some cases, typically for autonomous agents, places correspond

to physical objects or locations (landmarks), and navigation means physical motion between them. In other cases, navigation takes place at a logical level; here we review three examples:

- *Personalized tour planners.* In [18] we described an assistant for planning visits to a museum which enables the user to become acquainted with the artistic contents of the museum, choose some aspects to be further investigated and plan an itinerary taking a number of constraints into consideration. Places correspond to works of art and facilities, and are classified according to different criteria (author, historical period, etc.); clusters at different levels correspond to rooms, apartments and sections. An example of a simple path-planning problem that may be formulated is: "two hours available time; see all Van Gogh and Monet and at least one sculpture by Michelangelo, visit the Duke's Apartment, a brief stop at Leonardo's Mona Lisa, overlook English painters". Similar considerations could be made for information systems for planning cultural itineraries within an artistic city or day-trips to an amusement park.

- *Vehicle navigation systems.* They assist drivers in planning trips and selecting routes, by guiding them through geographic space. In the commercial systems emerging, cars are equipped with an on-board computer, a dash-mounted graphic screen displaying maps and conveying driving instructions, and sensors that return the car position. Spatial objects to be modelled include, for instance, roads, intersections, monuments, shops, and on a different scale, cities, highway exits, clover-leaf junctions; all the objects modelled are grouped into categories (for instance, gas stations, highway exits, restaurants, etc.). The cost might be computed in terms of distance, time, fuel consumption, toll, traffic. Some examples of navigation queries which could be formulated are: "which is the next highway exit?", "which is the closest gas station?", "how long to an Italian restaurant?", "which is the most convenient route to Venice?".

- *Personal planners.* We call personal planner an information system supporting constrained path planning on the city map for scheduling the errands and the appointments of the day. Real-time communication with the outside world is necessary in order to acquire on-line information about the traffic conditions and the social events, the shop and office hours, pictures of places, etc. An example of a natural language sentence expressing some errands for the day is: "take the dog to Hyde Park; collect spectacles at the optician; have lunch at a Chinese restaurant; check out some apartments in residential quarters".

Environment knowledge is not the only type of knowledge which can be effectively modelled. Consider for instance a semantic net: places correspond to concepts, and inter-place associations to associations between concepts; clustering allows for highly-interrelated areas of the net to be outlined. Navigation within the net is ruled by a cost function related to the strength of the associations between concepts.

An interesting example of a non-environmental information space which could be modelled by our architecture is the World Wide Web (WWW). Here, places

and associations correspond, respectively, to uniform resource locators (URLs) and to hyperlinks between them. A cluster identifies the set of URLs referring to the same server; most hyperlinks in each page fall within the server boundary, hence, connectivity within each cluster tends to be higher than inter-cluster connectivity. The applicability of our exploration technique to the WWW will be discussed in section 6..

3. Exploration profiles

The problem of learning the description of an unknown environment by exploring it and interpreting sensor data has been largely addressed in the robotic literature. Some approaches are oriented towards building 2- or 3-dimensional scene maps in the absence of landmarks [2, 14]. The approach to landmark exploration proposed in [24] mainly addresses the problem of recognizing places in spite of positional errors due to sensors. Also in [13] the accent is placed on landmark recognition; an agenda is used to remember the unexplored directions, but the next move is chosen through heuristic criteria such as that of the least rotation. In [22], the problem of acquiring a model of an unknown terrain is approached by implementing a graph search on an incrementally constructed geometric structure called the navigational course, whose vertices correspond to obstacle vertices. [8] describes a vertex-oriented deterministic exploration of an undirected graph; markers are dropped and picked up at places, so that the robot can recognize a place it has visited before.

In our approach, exploration is carried out at a symbolic level; this means that the agents' goal is to acquire knowledge of the hierarchy of graphs representing the environment. The link between symbolic exploration and the sensor level is established by assuming that:

- When an agent reaches a landmark, it can determine whether it has already visited that landmark or not.
- When an agent is located in a landmark, it can determine the directions of the routes departing from that landmark. If a restricted number of paths are physically possible in the environment (for instance, the streets in a city), these can be directly recognized by sensors (for instance, a sonar array); otherwise, assuming that landmarks can be sensed only within a given distance range, the agent will consider as routes all the paths leading to the visible landmarks.
- When an agent is located in a landmark, it can choose to explore one of the routes sensed.
- The agents know the number of levels of clustering at which the environment is to be represented; they can recognize the routes belonging to a k-bridge for any k (for instance, a route crossing a threshold).

Given these assumptions, we are mainly interested in investigating how an agent, each time it reaches a landmark, should decide which route to follow next in order to accomplish globally a given exploration strategy.

Exploration may be carried out according to a multiplicity of criteria. In order to characterize formally the different exploration strategies, we introduce in this

section the concept of *exploration profile*. The exploration profile describing a given strategy σ is a vector

$$\mathbf{p}_\sigma = [p_\sigma^{(1)}, ...p_\sigma^{(n-1)}]$$

whose k-th component is the ratio between the number $v_\sigma^{(k)}$ of k-clusters known at the time when v landmarks have been experienced if strategy σ is being followed, and v:

$$p_\sigma^{(k)} = p_\sigma^{(k)}(v) = \frac{v_\sigma^{(k)}}{v}$$

In general, each component is a function of v.

The exploration profile describes the evolution of the global knowledge of the environment during exploration in terms of the relative amounts of knowledge at the different abstraction levels. It can be used to calculate the adherence of an ongoing exploration process to a given strategy by comparing the actual distribution of knowledge to the ideal one: exploration fully adheres to strategy σ if, at any time, the ratio between the number of k-clusters and the number v of landmarks in the knowledge-base is equal to $p_\sigma^{(k)}(v)$, for $k = 1, ...n-1$.

It is necessary to point out that the exploration strategies are defined with reference to the global knowledge in one or more agents' possession. If only one agent is exploring the environment, then the bridges and clusters experienced are those that the agent has visited "personally". In the multi-agent case, we may say that a place has been experienced if at least one of the agents has visited that place; depending on the policies adopted for inter-agent communication, the knowledge that one agent has visited a given place may or not be shared by all the agents or some of them. Thus, in general, the environmental knowledge we will refer to in the following is the union of those in the single agents' possession.

3.1. The w-optimal strategy

Layering the representation of the environment enables the definition of different specifically-oriented exploration strategies. We call *w-optimal* an exploration strategy whose primary goal is to acquire knowledge of the graph representing the w-clustered layer, $\mathcal{L}^{(w)}$. A w-optimal strategy aims at exhaustively exploring the whole w-clustered layer, without considering the other abstraction levels. The w-clustered layer is completely explored when all the w-bridges have been experienced in both directions, i.e., at least two opposite routes belonging to each w-bridge have been covered. We assume that, when a landmark is experienced, the ancestors of that landmark on all levels are also experienced (if an agent sees a computer, it necessarily sees the room containing the computer and the floor where the room is); the same can be said for routes and bridges. Thus, it is obvious that exhaustive knowledge of any clustered layer implies exhaustive knowledge of all the layers above it.

In estimating the exploration profile of the w-optimal strategy we consider the case of a single agent for the sake of clarity; all the results obtained are equally valid for the multi-agent case, if the global knowledge in all the agents' possession is considered, and assuming that all the agents cooperate in order to pursue the given strategy.

Consider an agent which is following a 0-optimal strategy. The agent aims at exhaustively exploring the symbolic layer, that is, at experiencing all the routes; we may assume that it explores all the routes within each 1-cluster, then it crosses a 1-bridge and explores all the routes within the adjacent 1-cluster, and so on. If c_1 is the average cardinality of 1-clusters, the agent discovers a new 1-cluster for every c_1 landmarks visited; similarly, if c_2 is the average cardinality of 2-clusters, it discovers a new 2-cluster for every c_2 1-clusters visited. Thus, at the time when v landmarks are known, an agent following a 0-optimal strategy should know a number of k-clusters equal to

$$v_0^{(k)} = \frac{v}{\prod_{i=1}^{k} c_i} \qquad (k = 1, ...n-1)$$

where c_i is the average cardinality of i-clusters.

Consider now an agent which is following a 1-optimal strategy. The agent aims at exhaustively exploring the 1-clustered layer, that is, at experiencing all the 1-bridges. Seen on the symbolic layer, it moves in straight-line paths which cross the 1-clusters without necessarily visiting all their landmarks; seen on the 1-clustered layer it follows a path which, 2-cluster by 2-cluster, exhaustively visits all the 1-clusters (exactly as an agent following a 0-optimal strategy would visit all the landmarks). The agent discovers a new 1-cluster approximately for every $\sqrt{c_1}$ landmarks visited (estimate of the cluster diameter), whereas it discovers a new 2-cluster for every c_2 1-clusters visited. After visiting v landmarks, an agent following a 1-optimal strategy should know a number of k-clusters equal to

$$v_1^{(k)} = \begin{cases} \dfrac{v}{\sqrt{c_1}} & (k = 1) \\ \dfrac{v}{\sqrt{c_1}\prod_{i=2}^{k} c_i} & (k = 2, ...n-1) \end{cases}$$

Similarly, an agent following a 2-optimal strategy quickly crosses the 2-clusters and all the 1-clusters inside them, discovering a new 2-cluster for every $\sqrt{c_2}$ 1-clusters visited:

$$v_2^{(k)} = \begin{cases} \dfrac{v}{\sqrt{c_1}} & (k = 1) \\ \dfrac{v}{\sqrt{c_1 c_2}} & (k = 2) \\ \dfrac{v}{\sqrt{c_1 c_2}\prod_{i=3}^{k} c_i} & (k = 3, ...n-1) \end{cases}$$

It is not surprising that the number $v_2^{(1)}$ of 1-clusters experienced is the same as in the 1-optimal case: in fact, in both cases 1-clusters are crossed in the same way. The difference lies in *which* 1-clusters are crossed: in the 2-optimal case, those lying on the diameter of the 2-clusters. Figure 4 reports an example on a simple regular map.

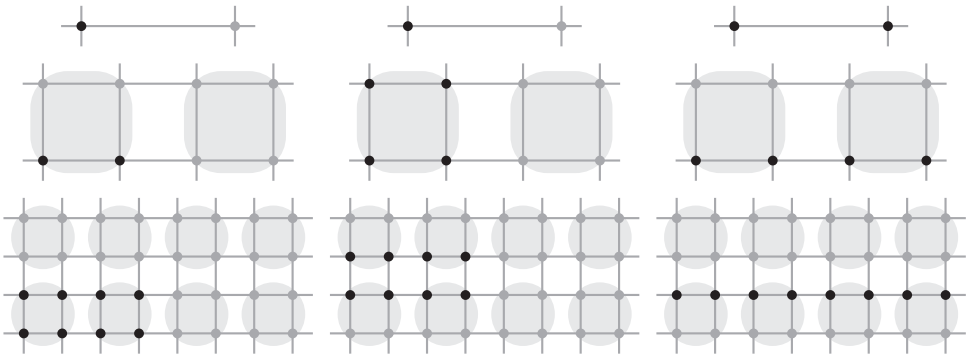


Figure 4: Different exploration strategies in a simple regular map where $c1 = c2 = 4$; known clusters are in black. When $v = 8$ landmarks have been visited, two 1-clusters and one 2-cluster have been visited if a 0-optimal strategy is being followed (a); four 1-clusters and one 2-cluster have been visited if a 1-optimal strategy is being followed (b); four 1-clusters and two 2-clusters have been visited if a 2-optimal strategy is being followed (c).

In general, for a w-optimal strategy ($w = 0, ...n-1$) it should be:

$$v_w^{(k)} = \begin{cases} \dfrac{v}{\prod_{i=1}^{k} \sqrt{c_i}} & (k = 1, ...w) \\ \dfrac{v}{\prod_{i=1}^{w} \sqrt{c_i} \prod_{i=w+1}^{k} c_i} & (k = w+1, ...n-1) \end{cases}$$

If landmarks and routes are uniformly distributed within the map and clusters are regularly-shaped, so that the values estimated above for $v_w^{(k)}$ can be used, the exploration profile describing the w-optimal strategy turns out to be:

$$\mathbf{p}_w = [p_w^{(1)}, ...p_w^{(n-1)}]$$

where

$$p_w^{(k)} = \begin{cases} \dfrac{1}{\prod_{i=1}^{k} \sqrt{c_i}} & (k = 1, ...w) \\ \dfrac{1}{\prod_{i=1}^{w} \sqrt{c_i} \prod_{i=w+1}^{k} c_i} & (k = w+1, ...n-1) \end{cases}$$

The profiles for the map in Figure 4 are as follows ($n = 3$, $c_1 = c_2 = 4$):

$$\mathbf{p}_0 = \left[\frac{1}{4}, \frac{1}{16}\right], \mathbf{p}_1 = \left[\frac{1}{2}, \frac{1}{8}\right], \mathbf{p}_2 = \left[\frac{1}{2}, \frac{1}{4}\right]$$

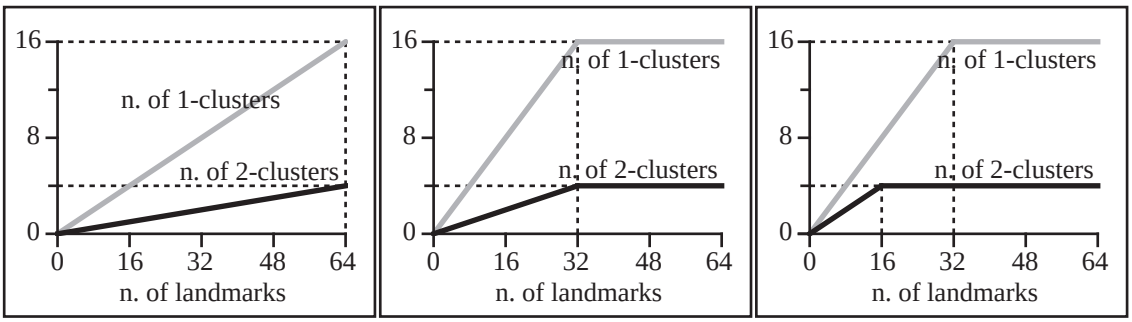


Figure 5: Expected number of 1-clusters and 2-clusters in function of the number of landmarks when a 0-optimal (a), 1-optimal (b) and 2-optimal (c) strategy is followed. We assumed $n = 3$ and $c_1 = c_2 = c_3 = 4$; hence, the map contains 64 landmarks, 16 1-clusters, 4 2-clusters, 1 3-cluster.

Actually, the w-optimal profile we have calculated above does not hold during the entire exploration. In fact, as v increases, the expected number of k-clusters calculated by means of the profile, $v_w^{(k)} = p_w^{(k)} \cdot v$, exceeds the total number of k-clusters in the k-clustered layer,

$$v_{tot}^{(k)} = \prod_{i=k+1}^{n} c_i$$

Thus, the expected number of k-clusters should be calculated as (see Figure 5):

$$v_w^{(k)} = min\{p_w^{(k)} \cdot v, v_{tot}^{(k)}\}$$

4. The agents

In our approach agents are homogeneous, are not coordinated by a central supervisor, and do not share any physical memory. Each agent has a private *knowledge-base* where the landmarks, routes, clusters and bridges known are stored. Every time an agent reaches an unknown landmark, it puts that landmark and the route it has just followed in its knowledge-base.

The agents communicate with each other by broadcasting messages; a message sent from an agent is received only by the agents who are currently placed within a circular area with radius ρ (*communication range*) and centred in the sender. We assume that message reception is error-free. The agents' communication protocol is described in sections 4.4. (as to team formation), 4.5. (management of the acquaintances) and 4.6. (knowledge sharing).

The goal of all the agents is to acquire knowledge of the whole environment. In principle, any exploration profile could be pursued by the agents, depending on the requirements of the specific application. Our main concern is that the agents be ready to carry out navigation tasks as soon as possible; since navigation tasks may

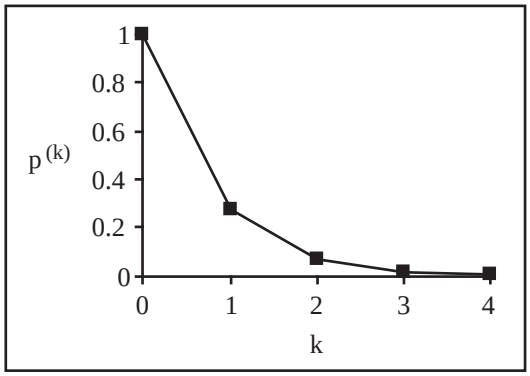


Figure 6: Optimal exploration profile as a function of k ($n = 5$, $c_1 = c_2 = ...c_5 = 10$).

be formulated in terms of places defined at any abstraction level, it is impossible to direct exploration a priori on a specific level.

Based on these considerations, in our approach we require agents to have at any time a general picture of all clustered layers; more precisely, we define the *optimal exploration profile* as the average of n profiles, each aimed at a different abstraction level (see Figure 6):

$$\mathbf{p} = [p^{(1)}, ...p^{(n-1)}]$$

where

$$p^{(k)} = \frac{1}{n}\sum_{w=0}^{n-1} p_w^{(k)} = \frac{1}{n\prod_{i=1}^{k}\sqrt{c_i}}\left(\sum_{j=1}^{k}\frac{1}{\prod_{i=1}^{j}\sqrt{c_i}} + n - k\right) \qquad (k = 1, ...n-1)$$

Our algorithm is aimed at having the union of the agents' private knowledge-bases grow, during exploration, according to the optimal profile $\mathbf{p}$; thus, the number $v^{(k)}$ of k-clusters which should be known when v landmarks have been experienced is:

$$v^{(k)} = min\{p^{(k)} \cdot v, v_{tot}^{(k)}\}$$

where $v_{tot}^{(k)}$ is the total number of k-clusters. Thus, for instance, if $n = 5$ and $c_1 = ... = c_5 = 10$, the optimal number of 2-clusters when v landmarks have been experienced is

$$v^{(2)} = min\{0.068 \cdot v, 1000\}$$

In section 5. we will evaluate to what extent the actual evolution of exploration adheres to the optimal profile.

4.1. Commitments, activities, modes of an agent

As argued in [12], *commitment* is a key concept in the theory of multi-agent systems. In general, a commitment may be viewed as a pledge to undertake a specified course of action. [3] distinguishes between internal, social and collective commitment; in the following we will refer to internal commitment, which corresponds to the

commitment defined in [6]. Internal commitment expresses a relation between an agent and an action, and is linked to the existence of a persistent goal. In our approach, at each time, every agent is either committed to exploring a given cluster (*scope*) or (temporarily) uncommitted. Each agent chooses its scope according to its current position and to the knowledge it has of the environment, be it directly experienced or transmitted from other agents. An agent is temporarily uncommitted when its current knowledge of the environment does not enable it to find itself a scope.

In [12] it is argued that commitments can be re-assessed not only when they have been satisfied, but also for other reasons (for instance, the agent has discovered some new information, or has interacted with another agent); the policies which rule reconsideration of the commitments are called *conventions*. In our approach, commitment to a scope may be dropped for four reasons:

(i) the scope has been completely explored;

(ii) the agent is not able to continue exploration of the scope;

(iii) a more convenient scope is found (agents reconsider the convenience of their scopes during the formation of a team; convenience is not evaluated from the point of view of the single agent, but from that of the collectivity of agents);

(iv) the agent, while moving to reach a particular route within its scope, detects a collision with another agent directed to the same route.

Basically, an agent may be involved in three *activities*. When it is outside the scope it is committed to, its actions are directed to reach the scope. When it is inside its scope, it carries out exploration of the scope by means of a graph-exploration algorithm, supported by an exploration agenda. An agent may also stand still, when it is temporarily uncommitted or is communicating with the surrounding agents in order to select a more convenient scope.

From an internal point of view, an agent may be seen as an automaton. Seven *operating modes* are defined: *explore*, *move_to*, *go_towards*, *go_to*, *wait_ack*, *wait_confirm*, *wait*. When it is in one of the first four modes, the agent moves in the environment in order to reach or explore the scope; hence, at each landmark, it must decide which route to follow next based on its current knowledge of the environment. The agent's mode determines the algorithm used to select a route, but is not sufficient to determine *which* route will be selected. Thus, the first four modes are characterized by an additional piece of information (*extension*) which, together with the operating mode, determines the agent's decision (for instance, when in mode *move_to*, the agent follows a path to a given destination; thus, mode *move_to* is extended with its destination).

Table 1 summarizes the agent's modes and extensions, while Figure 7 gives a global picture of how the transitions between the agents' modes occur. More detailed explanations will be given in sections 4.2. through 4.4. The complete algorithm is outlined in the Appendix.

Mode	Extension	Activity	Decision	Scope
explore	exploration level	explore the scope	graph-expl. algorithm	current cluster
move_to	destination (agenda)	reach/explore the scope	next in path	ancestor of destin. route
go_towards	target (call for team)	reach the scope	compass	target
go_to	target (call for team)		next in path	target
wait_ack	-	stand still	-	-evaluating-
wait_confirm				-evaluating-
wait				-uncommitted-

Table 1: Operating modes of an agent. The fourth column indicates the criterion used to select the next route; the fifth column contains the scope the agent is committed to.

4.2. Exploration

The main activity of the agents consists in carrying out exploration at a given abstraction level within their scope. We call k-*agent* ($k = 0, ..n-1$) one following a k-optimal strategy to explore a scope corresponding to a $(k+1)$-cluster. Since following a k-optimal strategy means being interested in acquiring knowledge of the graph which represents the k-clustered layer, we may say that a k-agent explores the environment "at level k".

The graph-exploration algorithm that agents adopt to explore their scope is a variant of Tremaux algorithm [23]. The classical Tremaux algorithm carries out exhaustive exploration of a directed graph by considering local knowledge only; it requires all arcs to have an opposite and it is optimal, meaning that each arc is visited exactly once. In our approach, several agents may have the same scope and thus interfere in each other's exploration schedule; moreover, we assume that some routes having no opposite may exist in the environment (corresponding to one-way streets, doors which can be opened one way only, etc.). Hence, it may occur that Tremaux algorithm suggests an agent to take a route which does not exist or has already been visited by some other agent; in our variant, when this happens we say that the agent has got "lost".

The graph-exploration algorithm can be sketched as follows:

```
ExploreGraph (vex,arc_set,from_arc)
/* the agent has reached vertex vex through arc from_arc; arc_set is
the set of the arcs departing from vex */

{  if not (vex visited for the first time)
   {  if ∃ to_arc∈arc_set:(to_arc=opposite(from_arc))∧(to_arc∉KB)
         return to_arc;      /* a cycle has been closed:turn back */
```

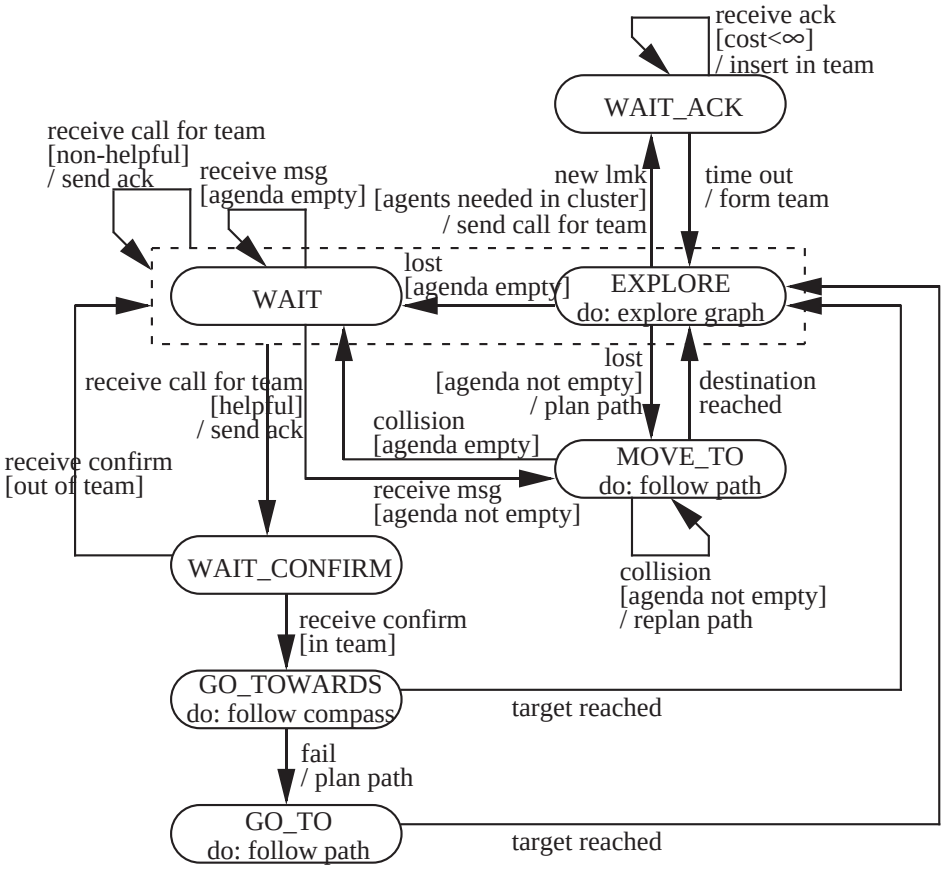


Figure 7: Mode chart for an agent. Following the OMT formalism, boxes represent modes, and arrows transition between modes. Each transition is labelled according to the syntax *event* [*condition*] /*action*, meaning that the transition is caused by *event*, occurs only if *condition* is true, and causes *action* to be executed. The activity executed while being in a state is expressed by a *do:* notation. The dashed rectangle defines a macro-mode.

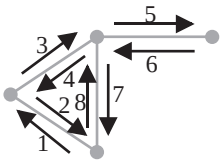


Figure 8: A small sample graph. The routes are numbered from 1 to 8.

route	lm. reached	choice	route	lm. reached	choice
1	U	go ahead	1	U	go ahead
3	U	go ahead	3	U	go ahead
7	K (cycle closed)	turn back	7	K (lost)	-
8	K	go ahead			
5	U (cul-de-sac)	turn back			
6	K	go back			
4	K	go back			
2	K (exp. over)	-			

Table 2: Exploration sequence. Each row reports the route just followed, the landmark reached (K stands for known, U for unknown) and the choice made by the graph-exploration algorithm. The sequence on the left describes a successful exploration; in the sequence on the right, the agent gets lost since another agent has already visited route 8.

```
      else
         if ∃ to_arc∈arc_set:(to_arc∉KB)∧(opposite(to_arc)∉KB)
            return to_arc;    /* go ahead */
         else
            if ∃ to_arc∈arc_set:to_arc∉KB
               return to_arc; /* go back */
            else
               return null;   /* exploration is over, or lost */
   }
   else
      if ∃ to_arc∈arc_set:(to_arc∉KB)∧(opposite(to_arc)∉KB)
         return to_arc;       /* go ahead */
      else
         if ∃ to_arc∈arc_set:to_arc∉KB
            return to_arc;    /* cul-de-sac:  turn back */
         else
            return null;      /* lost */
}
```

Consider as an example the small graph in Figure 8. Its exploration takes place as shown in Table 2, where two different cases are reported: on the left, the agent succeds in completing exploration of the graph; on the right, the agent cannot complete exploration and gets lost, since another agent has explored one of the routes.

From a conceptual point of view, the algorithm adopted by all agents to explore the graphs at the levels they are assigned to is the same. Nevertheless, while the 0-agents apply the algorithm to landmarks and routes, which are physical entities in the environment, the other agents apply it to clusters and bridges, which are only useful abstractions. In particular, while visibility of the routes departing from

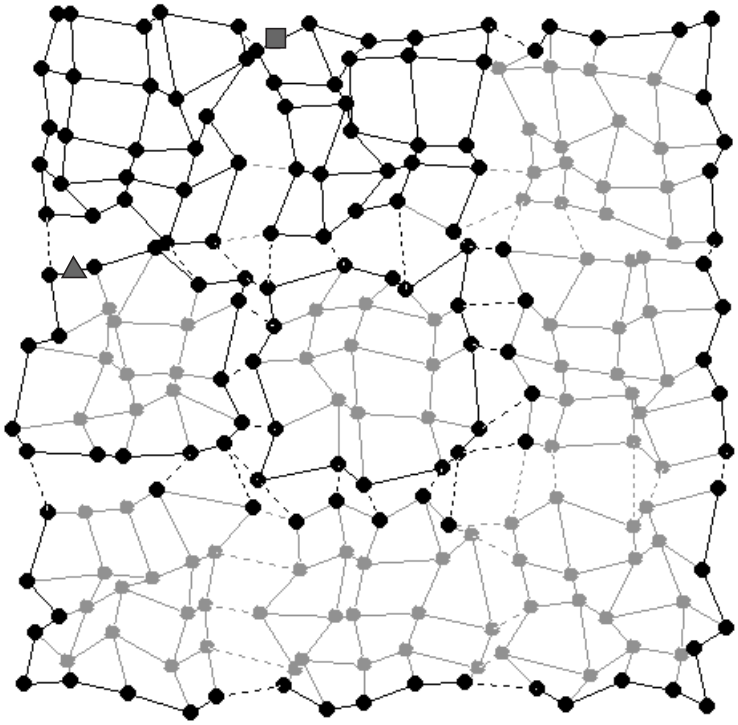


Figure 9: Different exploration paths followed by a 0-agent and a 1-agent in a map containing nine 1-clusters. Dashed routes are those belonging to 1-bridges; black routes are those already covered. The square and the triangle show, respectively, the current positions of the 0- and the 1-agent. The 0-agent has exhaustively explored the two clusters in the upper-left corner; the 1-agent is following the borders of the 1-clusters aimed at exploring the 1-clustered layer.

a landmark is guaranteed by the sensor level, the same is not true for k-clusters ($k > 0$): for instance, knowing which 1-bridges depart from a 1-cluster entails following the whole edge of the 1-cluster. Figure 9 shows the different exploration paths followed by a 0-agent and a 1-agent on the same map: the 0-agent carries out exhaustive exploration inside 1-clusters; the 1-agent, on the other hand, follows the edges of the 1-clusters and takes the routes contained in the 1-bridges. From a behavioural point of view we might say that, though all agents are equally "curious" (due to their standard exploration strategy), those working on low layers are "meticulous", while those working on high layers are more "superficial".

The operating mode corresponding to the activity of exploring a cluster is *explore*, and its extension is the exploration level k. When in this mode, every time the agent reaches a landmark it determines the new route to follow by applying the graph-exploration algorithm at level k.

In mode *explore*, the scope the agent is committed to is its current $(k+1)$-cluster. The commitment is dropped when the scope is completely explored; it may

also be dropped if the agent gets lost, unless it can resume exploration of the scope by consulting its agenda (see section 4.3.).

4.3. The agenda

Each agent has an *agenda* which it uses to continue exploration when, applying the graph-exploration algorithm in mode *explore*, it gets lost. The agenda is structured in n layers: the k-th one reports, for each known $(k+1)$-cluster, all the routes belonging to k-bridges that have not yet been explored. The agenda is updated every time the agent reaches an unknown landmark by adding the departing routes; routes are removed from the agenda as they are explored.

When a k-agent gets lost, it first consults its agenda *locally*, that is, it looks within the k-th layer of the agenda for a route included in the scope (that is, a route belonging to an unexplored bridge contained in the scope). If no such routes are found, the agenda is consulted *vertically*, that is, a route belonging to an unexplored bridge contained in one of the ancestors of the scope is searched. Finally, if the vertical search gives no results, the agenda is consulted *globally*, that is, on the whole map and on all levels.

The procedure for consulting the agenda is sketched below:

```
ConsultAgenda ()
/* Looks for a possible destination and returns the route to be
followed next, if any.  Position, Scope, ExploLevel, Destination are
part of the private memory of each agent; they store, respectively,
the landmark where the agent is or the last one where it has been,
the agent's current scope, its exploration level, its destination */

{  /* local consultation */
   Destination←r∈Agenda[ExploLevel]:(r internal to Scope)∧
         (r is nearest to Position);
   if (Destination is null)
   {  /* vertical consultation */
      newLevel←min{lev:(ExploLevel<lev<n)∧
            (∃r∈Agenda[lev]:r internal to Ancestor(lev+1,Scope))};
      if (newLevel is not null)
      {  ExploLevel←newLevel;
         Scope←Ancestor(ExploLevel+1,Scope);
         Destination←r∈Agenda[ExploLevel]:(r internal to Scope)∧
               (r is nearest to Position);
      }
      else
         /* global consultation */
         if (Agenda≠∅)
         {  Destination←r∈Agenda:(r is nearest to Position);
            ExploLevel←lev:Destination∈Agenda[lev];
            Scope←Ancestor(ExploLevel+1,Destination);
         }
```

```
        else
           Destination←null;
   if (Destination is not null)
   {  Mode←"MOVE_TO";
      RemoveAgenda(ExploLevel,Destination);
      Path←ShortestPath(Position,Destination);
      return NextIn(Path);
      /* returns the first route in the planned path */
   }
   else
      return null;
}
```

If, by consulting its agenda, the agent finds a route to follow (*Destination*), it plans a path to that route and enters mode *move_to*. In this mode, it simply follows the path planned until it reaches the destination; after visiting it, the agent enters mode *explore* again. The extension of mode *move_to* is the destination. If the agenda is empty, so that no destination can be found, the agent enters mode *wait*; it stops, and waits for a "call for team" or for any other message which enables it to put some routes in its agenda.

If the agent succeeds in finding a route by means of a local consultation, its commitment does not change, since the scope remains the same. In this case, since reaching the destination route may be considered to be a phase of the exploration, the activity of an agent in mode *move_to* is still that of exploring the scope. On the other hand, if local consultation fails, the agent has either completed exploration of the scope or is unable to continue it; hence, the commitment is dropped. If vertical or global consultation succeeds, the agent commits to a new scope: the cluster including the destination; in this case, the activity of an agent in mode *move_to* is that of reaching the scope. If the agenda is empty, the agent remains uncommitted; its activity is to stand still.

4.4. Team formation

The team formation mechanism is aimed at distributing agents effectively within the environment. Every time a k-agent ($k \geq 1$), while exploring its scope, discovers a k-cluster, it evaluates whether it is worth forming a team of $(k-1)$-agents to explore that k-cluster. The evaluation consists in comparing the number of $(k-1)$-agents currently exploring the k-cluster with the optimal number of $(k-1)$-agents per k-cluster. The agent estimates the current number of $(k-1)$-agents by counting its acquaintances (see section 4.5.). The optimal number of $(k-1)$-agents per k-cluster, $opt(k)$, is estimated as a function of the communication range ρ and of the average radius of k-clusters $rad(k)$ as follows.

In the optimal situation, agents are uniformly distributed within the cluster, for instance positioned in the vertices of a regular triangular-meshed grid. The optimal inter-agent distance is equal to ρ: in fact, if the agents were closer they would interfere with each other in applying the graph-exploration algorithm; if they were

further away, they would not be able to communicate. Thus:

$$opt(k) = \frac{3}{6}\frac{area\ of\ circular\ k{-}cluster}{area\ of\ a\ triangular\ mesh} = \frac{1}{2}\frac{\pi rad(k)^2}{\frac{\sqrt{3}}{4}\rho^2} = \gamma \cdot \left(\frac{rad(k)}{\rho}\right)^2, \gamma = \frac{2\pi}{\sqrt{3}}$$

(every triangular mesh contributes to $opt(k)$ with three vertices, and each vertex is shared by six meshes).

If the current number of $(k-1)$-agents in the k-cluster is less than the optimal number, the agent decides to issue a "call for team": it broadcasts a message *teamMSG*, whose argument is the k-cluster which the team is supposed to explore (which we will call *target*), and enters mode *wait_ack*. In this mode the agent waits for a given time, collecting the acknowledgements issued by the listening agents; its activity is to stand still. The agent may still be considered to be committed to its previous scope; actually, since it may itself become part of the team, it is evaluating whether it is worth committing to a more convenient scope.

Each agent receiving a "call for team" replies with the message *ackMSG*, whose argument is the cost to be paid for joining the team, which we call *membership cost*. A receiving agent may be willing to be part of the team, in which case it is said to be *helpful*, or not. An agent is always helpful if its mode is *wait*; if its mode is *explore*, it is helpful only if it is not the only agent exploring its cluster. In all the other cases, the agent is non-helpful.

A non-helpful agent includes in its acknowledge message an infinite membership cost, so that the caller does not include it in the team, and keeps operating in its previous mode.

For a helpful agent, however, the membership cost must be estimated. The membership cost is the sum of two contributions: the first, $cost'$, is the cost paid to reach the target, and is calculated as the cost of the cheapest path from the current agent's position to the target; the second contribution, *cost*", expresses the cost which the agent would pay to give up its job in its current cluster. Suppose the agent is exploring a k-cluster $\mathcal{C}$ as a $(k-1)$-agent; let $opt(k)$ be the optimal number of $(k-1)$-agents per k-cluster, and $curr(k-1, \mathcal{C})$ the current number of $(k-1)$-agents within $\mathcal{C}$ (excluding the agent itself). Contribution *cost*" is calculated considering that:

(i) Case $(opt(k) - curr(\mathrm{k}{-}1, \mathcal{C}) > 0$: should the agent leave $\mathcal{C}$, the number of agents remaining within $\mathcal{C}$ would become lower than it should be; the agent must be discouraged to leave, hence, *cost*" must be positive.

(ii) Case $(opt(k) - curr(\mathrm{k}{-}1, \mathcal{C}) < 0$: should the agent remain in $\mathcal{C}$, the agents within $\mathcal{C}$ would be more than they should be; the agent must be encouraged to leave, hence, *cost*" must be negative.

(iii) The higher the difference between the current and the optimal number of agents, the higher the cost should be (both in positive and in negative).

(iv) When $(opt(k) - curr(\mathrm{k}{-}1, \mathcal{C}) = 1$, meaning that one agent would be missing in order to reach the optimal situation, it should be *cost*" $= \rho$ (since ρ is the inter-agent distance in the optimal case).

Based on these considerations, we adopted for *cost*" the following expression:

$$cost" = \rho \cdot (opt(k) - curr(k-1, \mathcal{C}))$$

After transmitting its acknowledgement message, a helpful agent enters mode *wait_confirm*, and stands until it receives a confirm message from the caller.

The caller stands in mode *wait_ack* for a fixed interval of time. When this time expires, it evaluates the proposals it has received from the listening agents. It should be noted that the caller is itself a candidate for joining the team; its membership cost is calculated exactly as described above (since the caller is already on the target, it is $cost' = 0$). The number of agents which should form the team is equal to the number of $(k-1)$-agents missing in the target C, say

$$m = opt(k) - curr(k-1, \mathcal{C})$$

Let h be the number of helpful agents, candidate to form the team. If $h > m$, the $h - m$ agents having the highest membership costs are dropped from the set of candidates.

Consider for instance an agent ag_1 who has issued a call for team to explore a cluster where $m = 3$ agents are missing; let $\rho = 25$ be the communication radius. Four agents have received the call: ag_2, with membership cost $cost_2 = 20$; ag_3, with $cost_3 = 50$; ag_4, with $cost_4 = 30$; ag_5, with $cost_5 = \infty$. The membership cost of ag_1 is $cost_1 = 10$. The fifth agent is non-helpful, hence, it is $h = 4$. Since $h > m$, the agent having the highest membership cost, ag_3, is dropped from the set of candidates which thus becomes $\{ag_1, ag_2, ag_4\}$.

Initially, the team is formed by all the candidate agents. The *team cost* is defined as:

$$tc = \sum_{ag_i \in Team} cost_i + \rho\,(m - Cardinality(Team))$$

where $cost_i$ is the membership cost of agent ag_i. The first term expresses the cost paid by the members; the second expresses the cost paid for having less than the optimal number of agents in the team, and is estimated exactly as for contribution *cost*" in the membership cost (see above). In our example, the team cost is $tc = 60$ ($Cardinality(Team) = m$).

Then, a tentative team $Team'$ is formed by excluding the most costly agent, ag_j. Its cost is:

$$tc' = \sum_{ag_i \in Team'} cost_i + \rho\,(m - Cardinality(Team')) = tc - cost_j + \rho$$

where $cost_j$ is the membership cost of ag_j. If $tc' < tc$, it is convenient to drop ag_j from the team, since the cost paid by ag_j to join the team is higher than the cost for having one less agent. All candidate agents are considered, sorted by their membership costs in descending order, and are progressively dropped from the team until for one agent it is $tc' \geq tc$, meaning that a (sub)optimal team has been formed. In our example, the most costly agent is ag_4; the cost of the tentative team $\{ag_1, ag_2\}$ is 55; thus, ag_4 is dropped from the team. The next agent considered is ag_2; the cost of the tentative team $\{ag_1\}$ is 60; thus, $\{ag_1, ag_2\}$ is the optimal team.

The team-formation algorithm is sketched below:

```
FormTeam (cands)
/* cands is the set of candidate agents; each element is a pair
<ag_i,cost_i> */

{  while (Cardinality(cands)>m)
      cands←cands-GetMostCostly(cands);
   team←cands;
   while (Cardinality(team)>0)
   {  ag_j←GetMostCostly(team);
      if (cost_j>ρ)
         team←team-ag_j;
      else
         break;
   }
   return team;
}
```

After the team has been formed, the caller sends a message *confirmMSG* to all the agents from which it has received an acknowledge message; the message addressed to the agents chosen for forming the team includes the target and the level at which exploration will take place, $k-1$. The caller enters mode *explore*; if it is itself part of the team, it drops its previous commitment and commits to the target, beginning its exploration as a $(k-1)$-agent; otherwise, the agent maintains its commitment and resumes exploration of the scope as a k-agent.

Among the helpful agents, those which were not chosen resume their previous jobs. The agents in the team, on the other hand, drop their previous commitments and commit to the target; they enter mode *go_towards* and start moving towards the target. Mode *go_towards* is aimed at discovering new places and may be thought of as a compass navigation. At each landmark, the agent chooses the route whose direction differs the least from that of the target. If a convenient route cannot be found, the agent plans a path to the target and enters mode *go_to*, in which it follows the planned path similarly to mode *move_to*. For both modes *go_towards* and *go_to* the extension is the target; the activity is to reach the scope. When the scope is reached, the agent starts its teamwork and enters mode *explore* as a $(k-1)$-agent.

In Figure 10, the protocol for team formation is sketched.

It should be noted that team formation may be a recursive process. Consider a 3-agent which has discovered a 3-cluster and has issued a "call for team". Among the 2-agents in the team, the one who will first reach a 2-cluster within the target (the caller itself, if it is member of the team) may issue a new "call for team" to explore that 2-cluster. Similarly, one of the 1-agents in the new team may issue a call to form a team of 0-agents to explore the first 1-cluster discovered.

4.5. Acquaintances

The concept of *acquaintance* has a key role in supporting coherent social behaviour in multi-agent systems. In the concurrent object-oriented language ACTOR, the

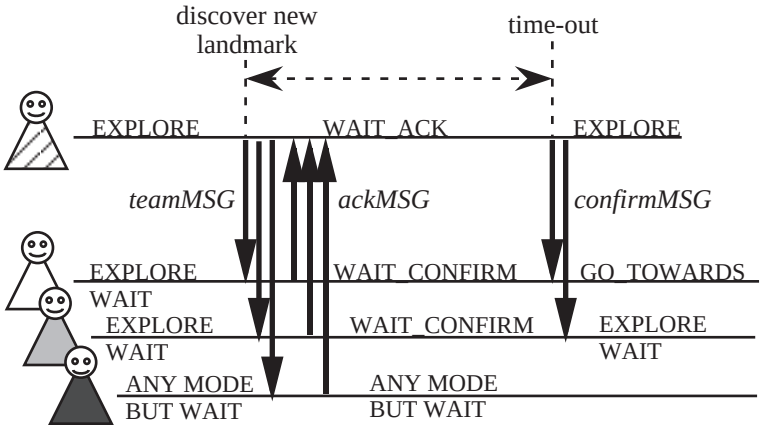


Figure 10: Communication protocol for team formation. The dashed agent is the caller. Among the three listening agents, the white one is helpful and is chosen for the team; the grey one is helpful but is excluded from the team; the black one is non-helpful.

acquaintance model is limited to representing the names and locations of the other agents known in the environment [1]. In MACE, an acquaintance is an explicit model of another agent and describes its roles, goals and skills [9]. In the case of exploration, an accurate acquaintance model enables each agent to cooperate with the other agents situated in the same area, in order to avoid repeated exploration of clusters and bridges.

In our approach, an acquaintance is determined by who the agent is, where it is, what it is doing, and how it chooses what to do next. More precisely, each acquaintance is organized as follows:

```
Acquaintance:  AgentId
{ Position ...
  Direction ...
  ExtdMode:   { Mode ...
                [Extension ...]
              }
}
```

where *Position* is the landmark where the agent is or, if the agent is following a route, the landmark it comes from; *Direction* is the route the agent is following or is about to follow; *ExtdMode* is the combination of the mode and the extension.

The acquaintances of an agent are stored in a private *message box*, so called because it is kept up-to-date through inter-agent communication. Just before leaving each landmark, agents broadcast a message *landmkMSG* which includes their position, their direction, their extended mode, the route they followed to reach the landmark and the routes departing from it (only those still included in their agenda). Every time an agent receives a *landmkMSG* from another agent, it stores in its message box an acquaintance containing the position, direction and extended

mode of that agent (the other information transmitted is used for knowledge sharing; see section 4.6.). If no *landmkMSG* from an agent is received for some time, meaning that it has exited the communication range, its acquaintance is deleted.

An agent uses its message box to calculate the total number of agents exploring its current cluster. By comparing this number with the optimal number of agents in that cluster, it decides if it is worth issuing a "call for team" to explore a cluster, to acknowledge a "call for team" it has received, and to change exploration level when consulting its agenda.

When in mode *move_to*, agents also use their message boxes to look for possible path collisions with other agents. Consider an agent A moving to a destination route r. Agent A detects a collision in two cases: when one of its acquaintances is in mode *explore* and is about to take route r, and when one of its acquaintances is in mode *move_to* with destination r and is nearer to r than A. Every time an agent detects a collision, it consults its agenda in order to find a new destination.

4.6. Knowledge sharing

In order to increase the agents' operativeness and fault tolerance, during exploration and especially when exploration is over, agents should know as much as possible about the whole environment; thus, sharing of the knowledge-base between the agents is strongly encouraged. Through message *landmkMSG*, each agent can inform the agents within the communication range as to the route it has just visited and the landmark it has met.

Agenda sharing is fundamental in avoiding routes being explored more than once by different agents. Message *landmkMSG* includes the set of the routes departing from the current landmark, as well as the one the agent is about to explore (the agent's direction); thus, the other agents can keep their agendas up-to-date by inserting in them all the routes received, except the one being explored by the sender.

Message *landmkMSG* carries local knowledge only; alone, it cannot guarantee a satisfactory level of knowledge sharing, since the communication range is limited. Thus, also a message carrying global knowledge has been provided.

Every time an agent receives any message from another agent which is not in its message box (meaning that the two agents have not communicated recently), it broadcasts a message *transmitMSG* including its whole knowledge-base and its agenda. Thus, two agents who have not met for some time are enabled to share their knowledge.

5. Performance evaluation

In this section we discuss the performance of our algorithm from three points of view: adherence to the optimal exploration profile, efficiency, fault tolerance. We will describe the evolution of exploration in function of the cost c paid by each agent from the beginning of exploration for travelling along the routes; if agents move at a constant speed, c may be assumed to be proportional to the time that has elapsed since the beginning of exploration.

The adherence of the exploration profile to the optimal one can be evaluated at cost c, when v landmarks have been experienced, as

$$profileAdherence(c) = 1 - \frac{1}{n-1} \sum_{k=1}^{n-1} \frac{|v'^{(k)}(c) - v^{(k)}|}{v^{(k)}}$$

where $v'^{(k)}(c)$ is the number of k-clusters actually experienced at cost c and $v^{(k)}$ is the optimal number of k-clusters calculated in function of v as shown in section 4. Figure 11.a shows, for a sample map, how the average profile adherence evolves during an exploration; adherence turns out to be higher than 80% for more than half the exploration global time. Figure 11.b shows how the adherence, averaged on the whole exploration, depends on the number of agents; as we could reasonably expect, adherence increases with the number of agents, since several agents can more easily be distributed on the different levels. Simulations show that, given the number of agents, the average adherence is not greatly affected by the communication range ρ.

It may seem that adherence to the profile is relatively low, especially during the first phases of exploration. This is mainly due to two distinct factors. Firstly, the optimal profile is calculated with reference to an ideal map, with a regular structure and with constant cluster cardinality on each level, which is not true in general and in particular for the sample maps used for simulations. Secondly, efficiency and profile adherence are often contrasting requirements which cannot be achieved together: our approach to exploration pursues a trade-off between the two, so that profile adherence is partially sacrificed in favour of efficiency.

Efficiency is evaluated by comparing, at the end of exploration, the cost paid by each agent with the ideal cost which would have been paid if no route had been taken more than once:

$$costPerAgent = \frac{1}{idealCost} \cdot \frac{totalCost}{g}$$

where g is the number of agents, $totalCost$ is the cost globally paid by all agents for exploration and $idealCost$ is the sum of the costs of all the routes in the environment ($totalCost \geq idealCost$). In an ideal situation, it is

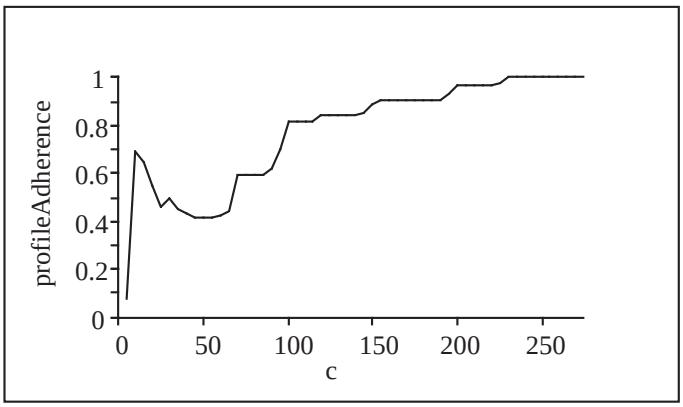
$$idealCostPerAgent = \frac{1}{g}$$

Figure 12 compares the cost per agent with the ideal cost per agent. The cost per agent is proportionally higher when several agents are employed, since they tend to interfere with each other.

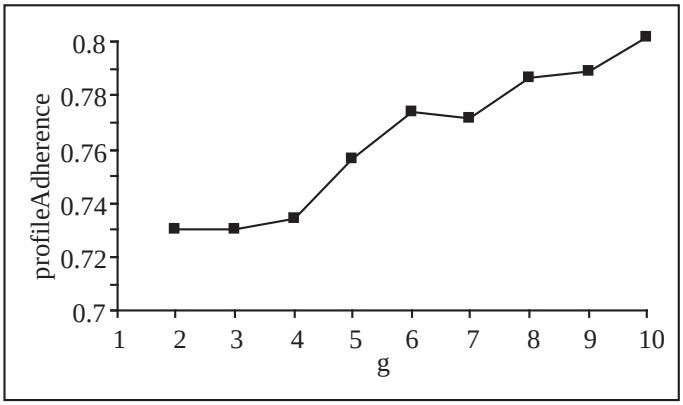
Fault tolerance is calculated at cost c as the average percentage degree of knowledge sharing:

$$faultTolerance(c) = \frac{1}{g} \cdot \frac{1}{r} \sum_{i=1}^{g} i \cdot h(i)) \leq 1$$

where r is the total number of known routes and $h(i)$ is the number of routes whose knowledge is shared by a number i of agents. When all g agents share all the



(a)



(b)

Figure 11: (a) Profile adherence during a 6-agents exploration, in function of the cost c paid by each agent. (b) Profile adherence averaged on the whole exploration, in function of the number of agents g. The sample map employed has more than 500 landmarks and 1400 routes, and has 4 clustering levels. The communication range is 50% of the total map diameter.

knowledge it is $h(g) = r$, hence $faultTolerance(c) = 1$. Figure 13.a shows how the average fault tolerance evolves during exploration; it is found to be higher than 97% during the whole exploration. Figure 13.b shows how the average fault tolerance depends on the number of agents; it appears that fault tolerance is always higher than 90%.

We close this section by discussing some issues concerning communication between agents. Figure 14 reports the average cost paid by an agent in the interval between two subsequent transmissions of the same message. The frequencies of the *landmkMSG* and *teamMSG* messages broadcasted by each agent appear to be substantially independent of the number of agents, as we could expect since these messages are connected to the map topology and not to interactions between agents. Instead, the frequency of the *transmitMSG* message grows linearly with the num-

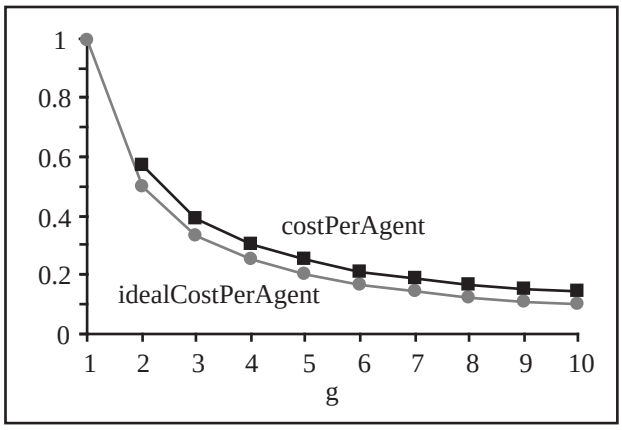


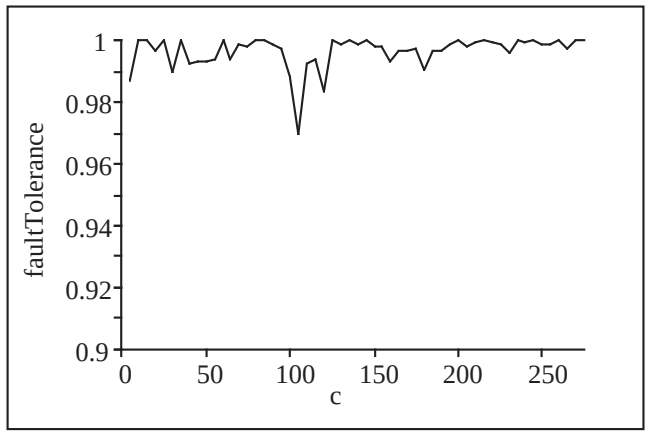
Figure 12: Cost per agent compared to the ideal cost per agent in function of the number of agents g.

ber of agents. It is interesting to observe that, while for the *landmkMSG* and the *transmitMSG* messages the frequency does not change significantly with the communication range, the frequency of the *teamMSG* message decreases when the communication range increases.

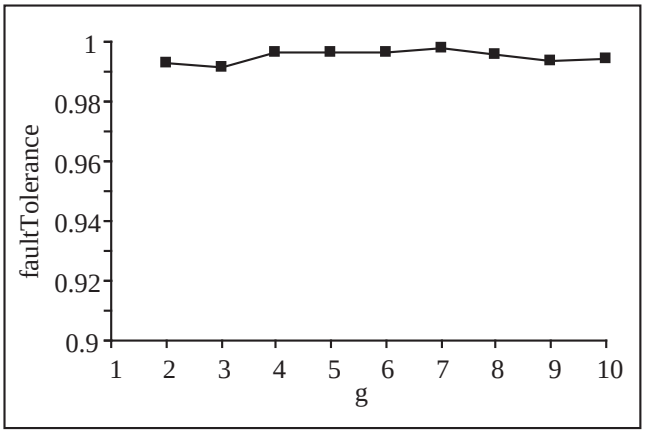
6. Conclusion

In this paper we have presented an algorithm for unsupervised multi-agent exploration of structured environments. According to the current necessity, each agent dynamically selects a specific abstraction level for exploring the environment; coordination with the other agents and knowledge sharing are accomplished by message broadcasting. A protocol for team formation is provided, aimed at conducting exploration of clusters more effectively.

We claim that the basic principles underlying our algorithm could be effectively applied to carry out exploration of large, unknown information spaces. Consider the WWW example, outlined in section 2.1.; in that context, exploration is relevant since it allows for indexes used by search engines to be kept constantly updated. Indeed, some aspects of our approach should be dropped. In the first place, whereas robot navigation in a physical environment requires knowledge of routes, navigation of the web requires knowledge of URLs: thus, the Tremaux-based graph-exploration algorithm, which is oriented to acquiring knowledge of routes, should be replaced with a landmark-oriented exploration algorithm. Secondly, while an unknown URL may have been reached for the first time by following a path of hyperlinks, once an URL is known it may be accessed directly, at a lower cost. Lastly, unless the exploring agent is capable of moving physically from one site to another, the cost for reaching an URL does not depend on the location of the previously visited URL: thus, reaching an URL from a neighbouring one is not necessarily less costly than reaching an URL from an URL located on a different server !(actually, some web explorers called *worms* do move from one site to another; for such explorers, moving



(a)



(b)

Figure 13: (a) Fault tolerance during a 6-agents exploration, in function of the cost c paid by each agent. (b) Fault tolerance averaged on the whole exploration, in function of the number of agents g.

between URLs on the same server is less costly than moving from one server to another). On the other hand, the most relevant aspects of our approach are worth to be preserved. In particular, the team formation protocol allows for agents to distribute effectively within the web, by crowding at those sites where several URLs are located. The acquaintance model discourages redundant exploration of URLs and sites, by making each agent aware of the allocation of the other agents on the web. The use of an agenda is still necessary to support the agent whenever it gets lost, due to the presence of other agents at the same site. Knowledge sharing, and in particular agenda sharing, lead the agents to a more accurate evaluation of the commitment scope.

Currently, we are working to extend our multi-agent approach to the case of heterogeneous agents which must carry out a set of navigational tasks in a structured

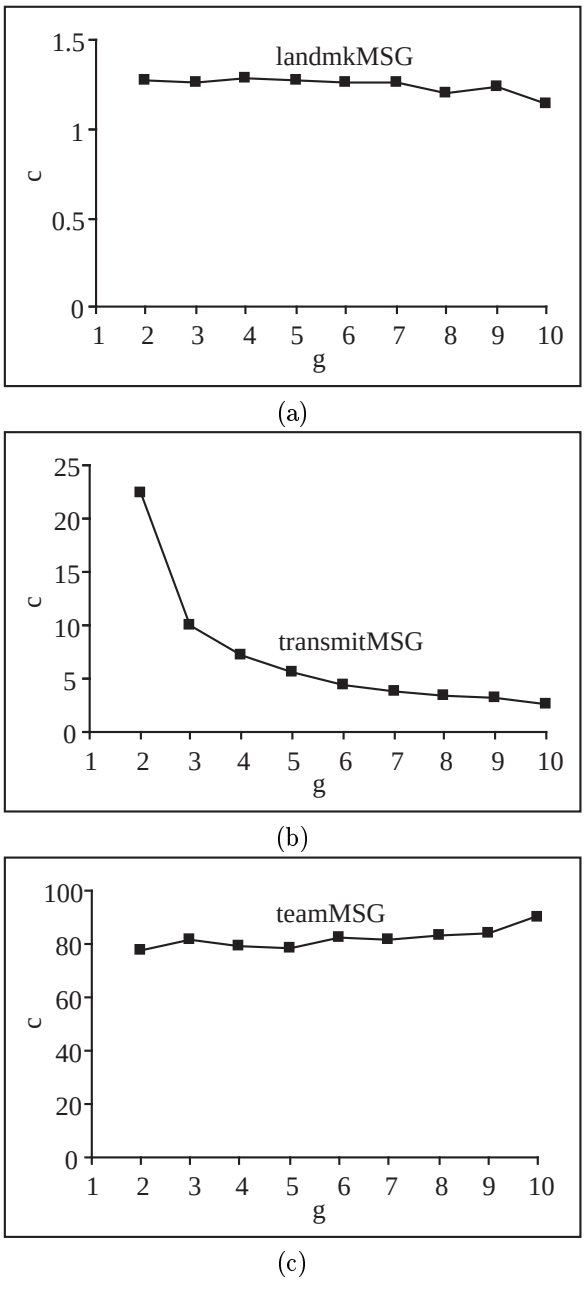


Figure 14: Average cost interval between two *landmkMSG* (a), *transmitMSG* (b) and *teamMSG* (c) messages broadcasted by one agent, in function of the number of agents g. Since the average cost of the routes of the map employed is 1, the diagram may also be read as reporting the average number of landmarks an agent visits in the interval between two subsequent transmissions of the message.

environment. An agent entrusted with a task that, due to the agent's capabilities or position, turns out to be very costly may decide to issue a "call for team" in order to entrust some sub-tasks to other agents; each agent replies by proposing an exchange with its most costly task. The evaluation of proposals and counterproposals gives rise to a negotiation protocol which, if successful, will lead to reassigning the tasks in a globally cheaper manner, and consequently to a significant advantage for all the partecipating agents.

Acknowledgement

Dr. Valentina Moriani and Dr. Matteo Golfarelli significantly contributed to the implementation of the exploration algorithm and to the testing of its performance. Their work is gratefully acknowledged.

References

[1] G. Agha. *Actors: a model of concurrent computation in distributed systems.* MIT Press, Cambridge, MA, 1986.

[2] M. Asada. Map building for a mobile robot from sensory data. *IEEE Transactions on Systems, Man and Cybernetics*, 37(6), 1990.

[3] C. Castelfranchi. Commitments: from individual intentions to groups and organizations. In *Proc. First Int. Conf. on Multi-Agent Systems*, pages 41–48, San Francisco, CA, 1995.

[4] H.I. Christensen, N.O. Kirkeby, S. Kristensen, L. Knudsen, and E. Granum. Model-driven vision for in-door navigation. *Robotics and Autonomous Systems*, 12:199–207, 1994.

[5] P. Ciaccia, D. Maio, and S. Rizzi. Integrating knowledge-based systems and neural networks for navigational tasks. In *Proc. IEEE COMPEURO*, pages 652–656, Bologna, Italy, 1991.

[6] P.R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[7] A.A. Covrigaru and R.K. Lindsay. Deterministic autonomous systems. *AI Magazine*, 12(3):110–117, 1991.

[8] G. Dudek, M. Jenkin, M. Evangelios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865, 1991.

[9] L. Gasser, C. Braganza, and N. Herman. Mace: a flexible testbed for distributed ai research. In M.N. Huhns, editor, *Distributed Artificial Intelligence*, pages 119–152. Morgan Kaufmann Publishers, San Mateo, CA, 1987.

[10] W.E.L. Grimson. *Object recognition by computer: the role of geometric constraints.* MIT Press, 1990.

[11] S.C. Hirtle and J. Jonides. Evidence of hierarchies in cognitive maps. *Memory and Cognition*, 13(3):208–217, 1985.

[12] N.R. Jennings. Commitments and conventions: the foundations of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[13] B.J. Kuipers and Y.T. Byun. A robust, qualitative method for robot spatial learning. In *Proc. AAAI 88*, volume 2, pages 774–779, Saint Paul, Minnesota, 1988.

[14] J. Leonard, H. Durrant Whyte, and I.J. Cox. Dynamic map building for an autonomous mobile robot. In *Proc. IEEE Int. Workshop on Intelligent Robots and Systems*, pages 89–95, 1990.

[15] D. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31(3):355–395, 1987.

[16] D. Maio and S. Rizzi. Knowledge architecture for environment representation in autonomous agents. In *Proc. Eighth Int. Symposium on Computer and Information Sciences*, pages 4–11, Istanbul, Turkey, 1993.

[17] D. Maio and S. Rizzi. Supervised multi-agent exploration of unknown environments. In *Proc. 3rd Workshop D-AI*IA*, pages 90–99, Rome, Italy, 1993.

[18] D. Maio and S. Rizzi. Cicero: an assistant for planning visits to a museum. In *Proc. 6th Int. Conf. on Database and Expert Systems Applications*, pages 564–573, London, England, 1995.

[19] D. Maio and S. Rizzi. Unsupervised Multi-Agent Exploration Of Structured Environments. In *Proc. First Int. Conf. on Multi-Agent Systems*, pages 269–275, San Francisco, CA, 1995.

[20] D. Nitzan. Development of intelligent robots: achievements and issues. *IEEE Journal of Robotics and Automation*, 1(1):3–13, 1985.

[21] W.A. Phillips, P.J.B. Hancock, N.J. Willson, and L.S. Smith. On the acquisition of object concepts from sensory data. In R. Eckmiller and Ch.v.d. Malsburg, editors, *Neural Computers*, NATO ASI Series, pages 159–168. Springer-Verlag, 1988.

[22] N.S.V. Rao and S.S. Iyengar. Autonomous robot navigation in unknown terrains: incidental learning and environmental exploration. *IEEE Transactions on Systems, Man and Cybernetics*, 20(6), 1990.

[23] P. Rosentiehl. Labyrinthologie mathématique. *Mathématiques et Sciences humaines*, 33:5–32, 1971.

[24] C.J. Taylor and D.J. Kriegman. Exploration strategies for mobile robots. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Atlanta, Giorgia, 1993.

Appendix

In this section the complete exploration script is outlined. We denote the variables belonging to the private memory of each agent with names beginning with a capital letter: *Position* (the landmark where the agent is or the last one where it has been), *Direction* (the route the agent is following), *Mode*, *ExploLevel* (the current exploration level), *Destination*, *Target*, *KB* (the knowledge-base), *Agenda*, *MBox* (the message box), *Path* (the path being followed to reach the destination or the target); it is convenient to denote with *Acq* a structure including the position, the direction and the mode together with its extension. Names beginning with small letters denote local variables and parameters.

```
Explore(lm)            /* lm is the landmark where I am initially */
{  Position←lm;
   Mode←"EXPLORE";
   ExploLevel←0;
   Destination←null;
   Target←null;
   KB←∅;
   Agenda←∅;
   MBox←∅;
   Path←empty;
   rs←RoutesFrom(lm); /* routes exiting lm, as determined by sensors */
   PutKB(lm);         /* PutKB adds a new landmark or route
                      to knowledge-base */
   PutAgenda(rs);     /* PutAgenda adds a set of routes to agenda */
   Direction←Random({r:r∈Agenda[0]});
   if (Direction is not null)
   {  Mode←"EXPLORE";
      ExploLevel←0;
      Scope←Ancestor(1,Position);
      RemoveAgenda(0,Direction);
      /* RemoveAgenda(k,r) removes route r from level k of agenda */
      Broadcast(all,landmkMSG(Acq,null,rs∩Agenda));
      /* the second parameter is the route just followed (if any),
      the third is the set of the routes departing and still included
      in agenda */
   }
   else
   {  Mode←"WAIT";
      /* all routes exiting lm have already been explored;
      wait for messages */
      Broadcast(all,landmkMSG(Acq,null,∅));
   }
   do
      if (Direction is not null)
         MoveIn(Direction);  /* else stand still */
```

```
   loop
}
```

Event handling:

```
when Landmark(lm, from_r) do
/* reached landmark lm through route from_r */
{  Position←lm;
   rs←RoutesFrom(lm);
   if (from_r∉KB)
      PutKB(from_r);
   if (lm∉KB)
   {  PutKB(lm);
      PutAgenda(rs);
   }
   case Mode of
   {  "EXPLORE" do
         Direction←ExploreDirection(rs,from_r);
      "MOVE_TO" do
         if (Position=SecondEndOf(Destination))
         /* reached my destination */
            Direction←ExploreDirection(rs,from_r);
         else
            /* look for collisions with other agents */
            if Collision()
            {  Direction←ConsultAgenda();
               if (Direction is null)
                  Mode←"WAIT";
            }
            else
               Direction←NextIn(Path);
      "GO_TO" do
         if (Position∈Descendants(0,Target))
         /* reached the target:teamwork starts */
            Direction←ExploreDirection(rs,from_r);
         else
            Direction←NextIn(Path);
      "GO_TOWARDS" do
         if (Position∈Descendants(0,Target)
         /* reached the target:teamwork starts */
            Direction←ExploreDirection(rs,from_r);
         else
            Direction←TowardsDirection(rs);
   }
   Broadcast(all,landmkMSG(Acq,from_r,rs∩Agenda));
}
```

```
when TimeOut do
/* the time for team formation has ended */
{  Team←FormTeam(Cands);
   /* send confirm messages to the agents in the team */
   for each agent∈Team do
      Broadcast(agent,confirmMSG(Ancestor(ExploLevel,Position),
           ExploLevel-1));
   if (myself∈Team)          /* I will be part of the team */
   {  Scope←Ancestor(ExploLevel,Position);
      ExploLevel←ExploLevel-1;
   }
   /* send messages to the agents not in team */
   for each agent∈(Cands-Team) do
      Broadcast(agent,confirmMSG(null,null));
   /* resume exploration */
   Direction←ExploreDirection(rs,from_r);
   Broadcast(all,landmkMSG(Acq,null,rs∩Agenda));
}

when Received(message,sender) do
/* received a message from sender */
{  if (sender∉MBox)  /* I have not met sender for a while */
      Broadcast(sender,transmitMSG(KB,Agenda);
   case message of
   {  landmkMSG(acq,from_r,rs) do
      {  if (from_r∉KB)
            PutKB(from_r);
         if (landmark∉KB)
         {  PutKB(landmark);
            PutAgenda(rs);
         }
         else
            RemoveAgenda(level,direction);
         if (acq.Mode="MOVE_TO")
            RemoveAgenda(level,destination);
         PutMBox(sender,<acq>);
      }
      transmitMSG(kb,agenda) do
      {  MergeKB(kb);
         MergeAgenda(agenda);
      }
      teamMSG(target) do
      {  if (Mode="EXPLORE"∧
               NumberOfAgentsWithin(Scope,ExploLevel)≥1)∨
               (Mode="WAIT")
         {  /* helpful */
            Broadcast(sender,ackMSG(MembershipCost(Position,target)));
```

```
            OldMode←Mode;         /* save my previous mode */
            OldDirection←Direction;
            Mode←"WAIT_CONFIRM";
            Direction←null;
            Broadcast(all,landmkMSG(Acq,null,null));
         }
         else
            /* non helpful */
            Broadcast(sender,ackMSG(∞));
      }
      ackMSG(cost) do
         if (cost<∞)
            Cands←Cands∪{<sender,cost>};
      confirmMSG(target,level) do
      {  if (target is not null)  /* I will be part of the team */
         {  Mode←"GO_TOWARDS";
            Target←target;
            ExploLevel←level;
            Scope←target;
         }
         else        /* I will not be part of the team */
            Mode←OldMode;
         Direction←OldDirection;  /* resume movement */
         Broadcast(all,landmkMSG(Acq,null,null));
      }
   }
   if (Mode="WAIT")
   /* see if agenda is still empty */
   {  Direction←ConsultAgenda();
      if (Direction is not null)
         Broadcast(all,landmkMSG(Acq,null,rs∩Agenda));
   }
}
```

Functions:

```
ExploreDirection(rs,from_r)
{  if (Position visited for the first time)∧
        (ExploLevel≥1)∧
        (AgentsNeeded(ExploLevel-1,Position)>0)
   /* send a call for team */
   {  Broadcast(all,teamMSG(Ancestor(ExploLevel,Position)));
      Mode←"WAIT_ACK";
      direction←null;
      Cands←{<myself,MembershipCost(Position,
           Ancestor(ExploLevel,Position))>};
      /* set of candidates to team membership */
   }
```

```
   else
   {  direction←ExploreGraph(Position,rs,from_r);
      if (direction is not null)
      {  Mode←"EXPLORE";
         RemoveAgenda(ExploLevel,direction);
      }
      else
         direction←ConsultAgenda();
      if (direction is null)
         Mode←"WAIT";
   }
   return direction;
}

TowardsDirection(rs)
{  direction←RouteInDirection(rs,Target);
   /* choose, among the routes in rs, the one closer to
   the direction of target */
   if (direction is not null)
   {  RemoveAgenda(ExploLevel,direction);
      return direction;
   }
   else
   {  Mode←"GO_TO";
      Path←ShortestPath(Position,Target);
      return NextIn(Path);
   }
}

AgentsNeeded(k,landmark)
   return OptimalNumberOfKAgentsPerCluster(k+1)-
         NumberOfAgentsWithin(Ancestor(k+1,landmark),k);

NumberOfAgentsWithin(cluster,k)
{  n←0;
   for each agent in MBox do
      if (MBox[agent].Position∈Descendants(0,cluster))∧
            (MBox[agent].Mode="EXPLORE")∧
            (MBox[agent].ExploLevel=k)
         n←n+1;
   return n;
}
```