وزارة التعليم العالي والبحث العلمي

BADJI MOKHTAR – ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR – ANNABA

جامعة باجي مختار– عنابة

Faculté : Sciences de l'Ingénieur **Année : 2009**
Département: Electronique

MEMOIRE
Présenté en vue de l'obtention du diplôme de MAGISTER

Intitulé

Conception of Multiagent System Applied on Mobile Robots

Option
Commande, détection de défauts et diagnostic des processus industriels

Par
Adel Djellal

DIRECTEUR DE MEMOIRE: Rabah Lakel Docteur-M.C. (A)

DEVANT Le JURY

PRESIDENT :	Pr. N. Debbeche	Professeur
EXAMINATEURS:	Pr. H. A. Abbassi	Professeur
	Dr. N. Guersi	Docteur-M.C. (A)
	Dr. B. Boulebtateche	Docteur-M.C. (B)

Abstract (Arabic)

لعبة السعي و التهرب هي إيجاد هدف غير متنبأ في مكان عمل. هي واحدة من المشاكل الأساسية التي درسها الباحثون في الآلية. المشكلة تنقسم إلى قسمين : العثور على عدد المطاردين اللازم لضمان العثور على المتهربين (إذا كانوا موجودين) ، وإيجاد الطريق الأمثل للمتعقبين لكي لا يتمكن الهارب من الهرب إلى جانب مستكشف، ثم يعثر عليه في نهاية المطاف. لتقليص النفقات، الحد الأدنى لعدد المطاردين سيكون من المهم أيضا التعرف عليه. التطبيقات غير مقتصرة بالضرورة على الخصومة. على سبيل المثال، الذين يقومون بجهود الإنقاذ حيث يمكن إيجاد استراتيجيات البحث آليا ويمكن أن تستخدم فيما بعد من طرف الروبوتات المتحركة أو باحثين بشريين.

الفكرة التي طرحت في هذا العمل هي تبسيط البيئة إلى عدد من النقط الحرجة و هي القمم (حواف الجدران مع فتح أكثر من 180 درجة)، وتوليد رؤية الرسم البياني (الرؤية بين القمم)، والعثور على العدد الأمثل للباحثين واقتراح الاستراتيجيات المثلى.

هذا العمل يحتوي على جزأين:

- **بحث دون اتصال** أين البيئة مدركة تماما من قبل الروبوت الرئيسي الذي يولد الإستراتيجية الأمثل و يتوقع عدد الروبوتات اللازمة لحراسة المبنى. اثنين من التقنيات مقترحة لهذا الجزء: أسوأ الحالات للتفتيش و البحث المحسن.

- **بحث باتصال** حيث البيئة مدركة جزئيا من طرف الروبوت الرئيسي الذي يرى فقط الوضع الحالي وماضي الوضعيات السابقة, ويتعين عليه أن يقرر عند كل وضع إما أن يتقدم أو يطلب روبوت آخر أو حتى العودة للبحث في منطقة أخرى. إثنين من التقنيات مقترحة لحل هذه المشكلة للبحث: البحث المباشر و البحث بالذهاب والعودة.

عمل تجريبي تم القيام به للتحقق من الأساليب المقترحة والمقارنة بينهم لتبيين خصائص كل أسلوب.

Abstract (English)

The pursuit-evasion game is problem of finding an unpredictable Target in a workspace with obstacles. It is one of the fundamental problems studied by robotic researchers to optimise the number of pursuers and the research time. The determination in dynamic manner the number of pursuers in a given moment is an important problem. Applications are not necessarily limited to adversarial targets. For example, people in search/rescue effort; where the search strategies can be automated and then used by either mobile robots or human searchers.

The idea proposed in this work is to simplify the environment into several critical positions which are vertices (edges of walls with opening degrees more than 180°), and to generate the visibility graph (visibility between vertices), and find the optimal number of searchers and their optimal motion strategies.

This work has two parts:

- **Offline processing** where the environment is totally percepted by the main agent that generates the optimal strategy and the optimal number of needed agents to clear the building. Two techniques are proposed for this part: Worst-Case Search and Improved Search.
- **Online processing** where the environment is locally percepted by the main agent that sees only the current position and has the history of previous states. The main agent has to decide for each position the appropriate action: go on, request another agent, or go back to explore other area. Two techniques are also proposed to solve this problem of search: Direct Search and Go-And-Back Search.

An experimental work has been done to verify the proposed techniques and a comparison between the obtained results with comments has been made to show the characteristics of each technique.

Abstract (French)

Le jeu de poursuit-évasion est un problème de recherche d'une cible imprédictible dans un espace de travail avec obstacles. C'est l'un des problèmes fondamentaux étudiés par les chercheurs dans le domaine de la robotique pour optimiser d'une part le nombre de poursuivants et d'autre part le temps de recherche. La détermination d'une manière dynamique le nombre d'agents poursuivants à un moment donné est un problème important. Les applications ne sont pas nécessairement limitées à la recherche de cibles ennemies. Par exemple, les efforts de recherche/secours de personnes en danger, où les stratégies de recherche peuvent être automatiquement déterminées et appliquées par des agents robots ou bien des intervenants humains.

L'idée proposée dans ce travail est de simplifier l'environnement et le représenter par quelques points critiques qui sont les vertex (nœuds de murs avec des degrés d'ouvertures plus de 180°), et de générer le graph de visibilité (visibilité entre vertex), et de trouver le nombre optimal de chercheurs et leurs stratégies optimales de mouvement.

Ce travail est structuré de deux parties :

- **Recherche Hors Ligne** où l'environnent est totalement perçu par l'agent maitre qui génère la stratégie optimale et estimer le nombre optimal d'agents guardiens demandés pour explorer le bâtiment. Deux techniques sont proposées dans cette partie : Recherche dans le pire des cas et Recherche Améliorée.

- **Recherche En Ligne** où l'environnement est localement perçu par l'agent principal qui voit seulement la position actuelle et a en mémoire l'historique des états précédents. L'agent principal doit décider pour chaque position l'action approprié : aller en avant, demander un autre agent, ou bien faire marche arrière pour visiter d'autres zones. Deux techniques sont proposées pour résoudre ce problème de recherche : Recherche Direct et Recherche en Va-Et-Vient.

Un travail expérimental a été fait pour valider les techniques proposées ainsi qu'une comparaison entre les résultats obtenus a été réalisée et commentée pour montrer les caractéristiques de chaque technique

To my Parents and my Love...

Contents

List of Figures

List of Tables

List of Algorithms

Preface

The pursuit-evasion problem is finding an unpredictable Target in a workspace with obstacles. The problem is divided into two parts: find the number of pursuers needed to guarantee that the evader will be found, and find the optimal path of the pursuers such that the evader can not escape to an explored part of the environment and will be eventually located. This problem can be generalised into unknown environment, where the searchers explore the building and try to detect the evader at the same time, the algorithms and techniques used in known environment can not be entirely used in the unknown environment because of the dynamic of visibility graph. Therefore, other techniques must be applied to solve the Pursuit-Evasion problem in unknown environment.

The first rigorous formulation of the pursuit-evasion problem is due to Parsons in 1976 [27], he restricted his study to the study of the environment represented as a discrete graph. Other concepts have been used to solve the Pursuit-Evasion problem, such as *Geometry-based view* [7, 13, 14, 22, 23, 38, 36, 45], where authors used polygons to represent environment. *Randomized view* [19, 1], and *probabilistic view* [15, 16, 41].

Work organisation

The first chapter presents the general context of the problematic of cooperation and competition in society of agents, also the tools needed to understand this topic.

The first part, containing chapters 2, 3, and 4, is reserved to present the techniques to model the environment also the development of techniques of search for the two cases of search: First case, environment is totally known a priory. Offline Search. Second case; the environment is partially known and its model is built dynamically during the progress of search: Online Search.

The second part, containing chapters 5 and 6, is reserved to the presentation of experimental work. It contains, in a part, the presentation of platform *NetLogo* used to construct the model of the environment and to simulate the techniques of search. And in other part, the results of simulation of different studied environments.

The conclusion helps to evaluate the presented work and the perspectives opened by this work.

Introduction Générale

Le jeu de poursuit-évasion est un problème de recherche de cibles inconnues dans un environnement à obstacles. Le problème est divisé en deux parties : trouver le nombre d'agents nécessaires pour garantir que l'évadé soit trouvé (s'il existe), et de trouver le chemin optimal des poursuivants de tel sorte que l'évadé ne fuit pas vers une zone déjà explorée et décontaminée, et qu'il sera finalement localisé. Ce problème peut être généralisé vers des environnements inconnus, où les poursuivants explorent le bâtiment et essayent de détecter l'évadé en même temps. Les algorithmes et techniques utilisés dans un environnement connu ne peuvent pas être utilisés entièrement pour des environnements inconnus à cause de la dynamique du model de l'environnement. Alors, autres techniques doivent être appliquées pour résoudre ce problème de poursuit-évasion.

La première formulation rigoureuse de poursuit-évasion est faite par Parsons en 1976 [27], il a limité son étude pour le cas d'environnement représenté autant que graph discret. Autres concepts ont été utilisés pour résoudre le problème de Poursuit-Evasion, tel que *point de vu géométrique* [7, 13, 14, 22, 23, 38, 36, 45], où les auteurs utilisaient des polygones pour représenter l'environnement. *Point de vu aléatoire* [19, 1], et *point de vu probabiliste* [15, 16, 41].

Organisation de travail

Le premier chapitre permet de présenter le contexte général de la problématique de la coopération et de la compétition dans une société de robots, ainsi que les outils nécessaires pour apprendre le domaine.

La première partie, comprenant chapitre 2, 3 et 4, est réservée à la présentation des techniques de modélisation de l'environnent ainsi que le développement des techniques de recherche dans les deux cas étudiés : Premier cas, l'environnement est totalement connu à priori : Recherche Offline. Deuxième cas, l'environnement est partiellement connu et son modèle est construit dynamiquement au cours de progression de la recherche : Recherche Online.

La deuxième partie, contenant chapitres 5 et 6, est réservée à la présentation du travail expérimental. Elle contient, d'une part, la présentation de la plateforme logicielle *NetLogo* utilisée pour construire le modèle de l'environnement et pour la simulation des techniques de recherche. Et d'autre part, les résultats de la simulation pour les différents environnements étudiés.

La conclusion permet de mettre en valeur les perspectives ouvertes par ce travail.

Chapter 1

State of the art

1.1 Introduction

The first step for each research is to study the proposed methods by other works to be able to justify the proposed work.

This chapter begins with an overview about the Pursuit-Evasion Problem. Afterwards, the several methods used to solve the Pursuit-Evasion problem are presented. The next step is the background needed to easily understand the Pursuit-Evasion research.

1.2 Problematic

The pursuit-evasion game is finding an unpredictable Target in a workspace with obstacles, it is known as one of the fundamental problems studied by robotic researchers. The problem is divided into two parts: find the number of pursuers needed to guarantee that the evader will be found, and find the optimal path of the pursuer(s) such that the evader can not escape to an explored part of the environment and will be eventually be located. This problem can be generalised into unknown environment [26], where the searcher(s) explore the building and try to detect the evader at the same time, the algorithms and techniques used in known environment can not be used all in the unknown environment because of the dynamic of visibility graph[1]. Therefore, other techniques must be applied to solve the Pursuit-Evasion problem in unknown environment.

The Pursuit-Evasion problem can be used to solve several problems of security (clearing civic and industrial buildings) and safety (searching survivors in inflamed buildings and clearing hazardous industrial areas).

1.3 Previous work

The first rigorous formulation of the pursuit-evasion problem is due to Parsons in 1976 [27], he restricted his study to the study of the environment represented as a discrete graph, the evader is assumed to be able to move arbitrarily fast through the graph.

Other concepts have been used to solve the Pursuit-Evasion problem, such as Geometry-based view [7, 13, 14, 22, 23, 38, 36, 45], where authors used polygons to represent environment. Randomized view [19, 1], and probabilistic view [15, 16, 41].

[1] Modelling technique of search space

1.3.1 Visibility-based point of view

Many variations of the polygon search problem (also called visibility-based problem) [7, 13, 14, 23, 45] have been proposed and studied in the literature since its first proposal by Suzuki and Yamashita [38].

Detection of mobile intruders in a simple polygon was first considered in the searchlight scheduling problem [36] in which the rays of stationary searchlights are used to find the intruder. The use of a mobile searcher having various degrees of visibility for detecting mobile intruders was then considered as polygon search problem in [38] where a number of necessary conditions and sufficient conditions for given polygon to be searchable by various searchers are presented.

The visibility-based pursuit-evasion problem used a continuous polygonal environment, and coined the term *k-searcher*. In this formulation, in order to find an evader, a k-searcher do not need touch the evader, but can instead "see" the evader from a distance. The k-searcher is equipped with k infinitely thin "flashlights" with which it can search the environment. These flashlights have unlimited range (but cannot see through walls) and can be freely rotated around the searcher at bounded speed and independently of searcher's motion. Commonly studied are cases when $k = 1$, $k = 2$, and $k = \infty$ [14, 22, 36]. The ∞-searcher can see in all directions at once. [13]

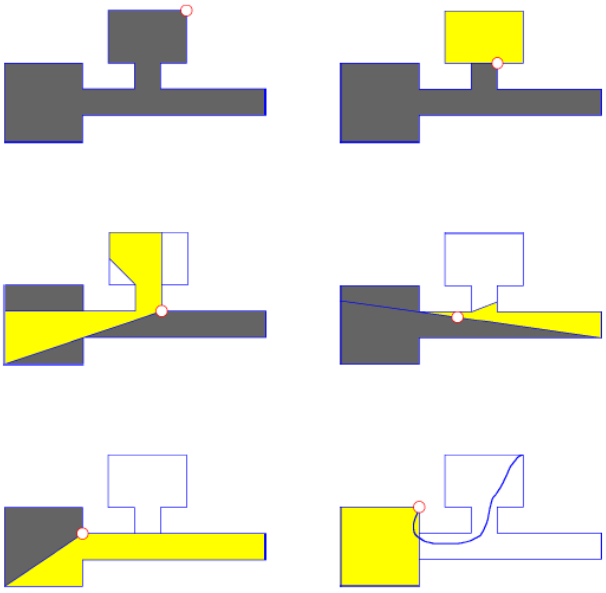


Figure 1.1: A computed clearing trajectory for a π-searcher.[13]

1.3.2 Randomized point of view

Another point of view is based on randomized algorithms to solve the pursuit-evasion problem [18, 1]. The random-based pursuit-evasion problem used environments represented by graphs and utilises random-based algorithms to find the optimal strategy to locate the intruder(s), so works go further by trying to capture the located intruder [18].

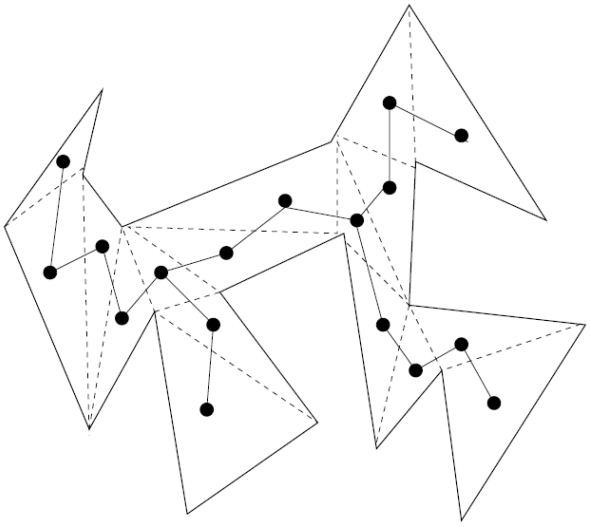


Figure 1.2: Triangulation of a polygon and its dual tree.[18]

1.3.3 Probabilistic point of view

Other works used probabilistic tools to solve pursuit-evasion problem [15, 16, 41]. The classical approach to pursuit-evasion games is to first build a map of the terrain and then play the game in a known environment. For the map building stage, several techniques have been proposed, see e.g. [11] and references therein. Most of them are based on Bayesian estimation and are implemented using Extender Kalman Filter. The main problem with these map building techniques is that they are time consuming and computationally expensive, even in the case of simple two dimensional rectilinear environments [9]. On the other hand, most of the literature in pursuit-evasion games, see e.g. [4, 15, 29, 38, 35, 40], assumes worst case motion for the evaders and an accurate map of environment.

In [15] the pursuit-evasion game and map building problems are combined in a single probabilistic framework. The basic scenario considers multiple pursuers trying to capture a single randomly moving evader. In [40] we extended the scenario to consider multiple evaders and proposed a single vision-based algorithm for evaders and proposed a simple vision-based algorithm for evader detection.

In [41] the implemented probabilistic framework was on a team of Unmanned Aerial Vehicle (UAV) and Unmanned Ground Vehicle (UGV). [15] proposed a "greedy" policy to control a swarm of autonomous agents in the pursuit of one of the several evaders.

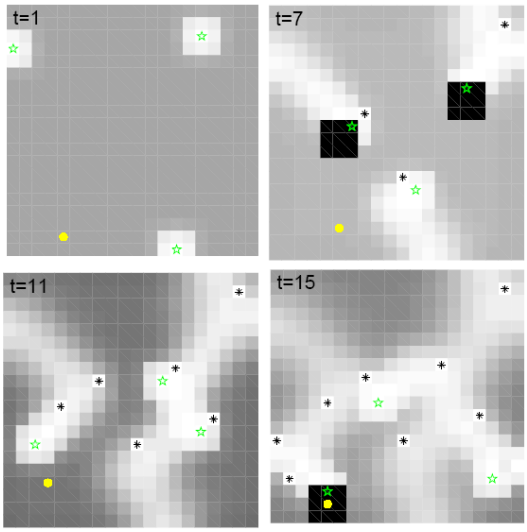


Figure 1.3: Pursuit with the constrained greedy policy.[15]

1.4 Background

To master and understand concepts of the pursuit-evasion problem, basic knowledge about agents, multiagent systems, and mobile robots is needed.

1.4.1 What are agents?

Surely, we must all agree on what an agent is. Surprisingly, there is no such agreement: there is no universally accepted definition of term agent, and indeed there is a good deal of ongoing debate and controversy on this very important subject. Essentially, while there is a general consensus that *autonomy* is central to the notion of *agency*, there is little agreement beyond this. Part of the difficulty is that various attributes associated with agency are of differing importance for different domains. Thus, for some applications, the ability of agents to learn from their experiments is of paramount importance; for other applications, learning is not only unimportant, it is undesirable.

The definition presented here is adapted from [44]:

Definition 1. *an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objective.*

Shoham defines Agent as follows [32]:

Definition 2. *An agent is an entity whose state is viewed as consisting on mental components such as beliefs, capabilities, choices and commitments.*

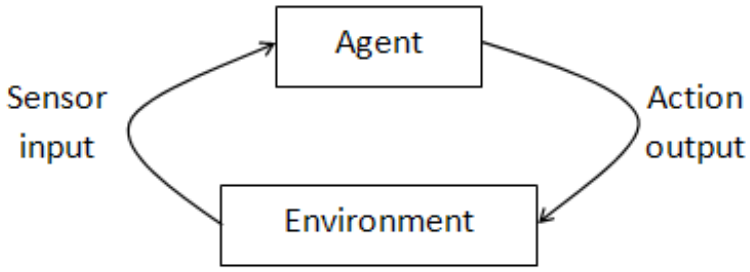


Figure 1.4: An agent in its environment

Figure 1.4 gives an abstract, top-level view of an agent. In this diagram, we can see the action output generated by the agent in order to affect its environment. In most domains of reasonable complexity, an agent will not have *complete* control over its environment. It has at best *partial* control, in that it can *influence* it. From the point of view of the agent, this means that the same action performed twice in apparently identical circumstances might appear to have entirely different effect. And in particular, it may *fail* to have the desired effect. Thus agents in all but the most trivial of environment must be prepared for the possibility of *failure*. We can sum this situation formally by saying that environments are *non-deterministic*[43].

Any agent should contain the following concepts: [28]

- **Persistence:** code is not executed on demand but runs continuously and decides for itself when it should perform some activity.
- **Autonomy:** agents have capability of task selection, prioritization, goal-directed behaviour, decision-making without human intervention.
- **Social Ability:** agents are able to engage other agents through some sort of communication and coordination; they may collaborate on a task.
- **Reactivity and Proactivity:** agents perceive the context in which they operate and react to it appropriately as per input.

Purely reactive agents

Certain types of agents decide what to do without reference to their history. They base their decision making entirely on the present, with no reference at all to the past. This kind of agents can be called *purely reactive*, since they simply respond directly to their environment. Formally, the behaviour of a purely reactive agent can be represented by a function

$$action = S \longrightarrow A \tag{1.1}$$

It should be easy to see that for every purely reactive agent, there is an equivalent standard agent; the reverse, however, is not generally the case.

The thermostat example is a good example of a purely reactive agent. Assume, without loss of generality, that the thermostat's environment can be in one of two states – either too cold, or

temperature OK. Then the thermostat's action function is simply

$$action\,(s)\begin{cases} heater\ off & \text{if s=temperature OK} \\ heater\ on & \text{otherwise} \end{cases} \tag{1.2}$$

Agents with state

Modelling an agent's decision function *action* as from sequence of environment states or *percepts* to actions allows us to represent agents whose decision making is influenced by history. However, this is a somewhat unintuitive representation, and we shall replace it by an equivalent, but somewhat more natural scheme. The idea is that we now consider agents that maintain *state* – see Figure 1.5. [43]

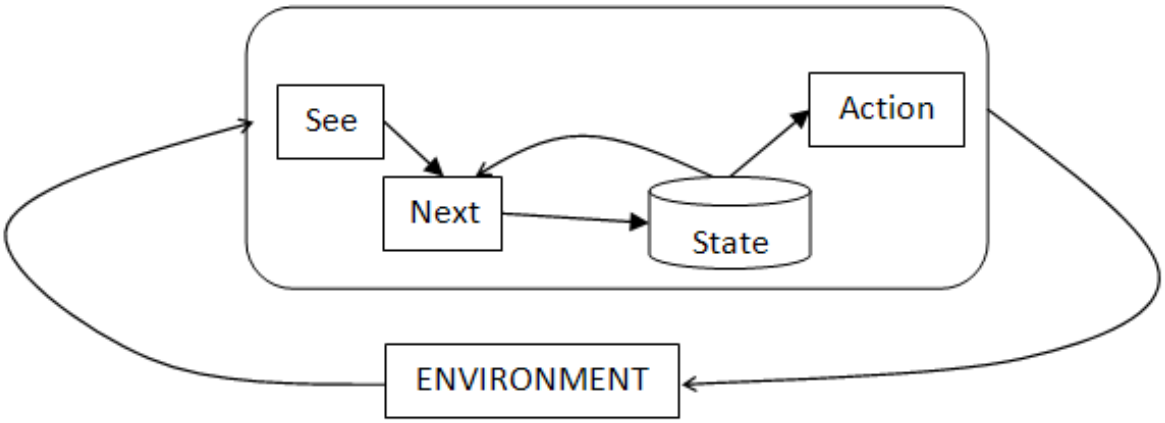


Figure 1.5: Agents that maintain state

1.4.2 Agent-Oriented Programming (AOP)

Yoav Shoham [31, 32] has proposed a "new programming paradigm, based on societal view of computation" which he calls Agent-Oriented Programming. The key idea which informs AOP is that of directly programming agents in terms of *mentalistic* notions (such as *belief*, *desire*, and *intentions*) that agent theorists have developed to represent the properties of agents. The motivation behind the proposal is that humans use such concepts as an *abstraction* mechanism for representing the properties of complex systems. In the same way that we use these mentalistic notions to describe and explain the behaviour of humans, so it might be useful to use them to program machines. [43]

1.4.3 Carachteristics of multiagent environment

1. Multiagent environments provide an infrastructure specifying communication and interaction protocols.

2. Multiagent environments are typically open and have no centralised designer.

3. Multiagent environments contain agents that are autonomous and distributed, and may be self-interested or cooperative.

Property	Definition
Knowable	To what is the environment known to the agent
Predictable	To what extent can it be predicted by the agent
Controllable	To what extent can the agent modify the environment
Historical	Do future states depend on the entire history, or only the current state
Teleological	Are parts of it purposeful, i.e., are there other agents
Real-time	Can the environment change while the agent is deliberating

Table 1.1: Environment-agent characteristics according to [43]

Table 1.1 lists some key properties of an environment with respect to a specific agent that inhabits it. These generalize the presentation in [30]

1.4.4 Societies of agents

Much of traditional Aritificial Intelligence has been concerned with how an agent can be constructed to function intelligently, with a single locus of internal reasoning and control implemented in a Von Neumann architecture. But intelligent systems do not function in isolation – they are at the very least a part of the environment in which they operate, and the environment typically contains other such as intelligent systems. Thus, it makes sense to view such systems in social terms.

A group of agents can form a small society in which they play different roles. The group defines the roles, and the roles define the commitments associated with them. When an agent joins a group, he joins in one or more roles, and acquires the commitments of that role. Agents join a group autonomously, but are then constrained by the commitments for the roles they adopt. The group define the *social context* in which the agents interact. [43]

1.4.5 Environment's Modelling

The environments in this work has been modelled using graphs, A *graph* is a pair $G = (V, E)$ of sets such that $E \subseteq [V]^2$; thus, the elements of E are 2-elements subsets of V. A *path* is a non-empty graph of the form $V = \{x_0, x_1, ..., x_k\}$ and $E = \{x_0x_1, x_1x_2, \ldots, x_{k-1}x_k\}$ Where the x_i are all distinct, the vertices x_0 and x_k are *linked* by P and are called its *ends*. If $P = x_0x_1...x_{k-1}$ is a path and $k \geq 3$, then the graph $C := P + x_{k-1}x_0$ is called a *cycle*.

1.4.6 Instrumentation

Actors

The wheel has been, by far, the most popular locomotion mechanism in mobile robotics and in man-made vehicles in general. It can achieve very good efficiencies and does so with a relatively simple mechanical implementation.

Perception

To percept walls, a method is proposed to detect vertices. This technique is based on laser to detect the distance from agent to walls, and then generate the virtual perception of current position of the master and its perception. This method is not used in the experimental work, the perception of vertices is supposed done before.

1.5 Conclusion

This chapter gave a general idea about the Pursuit-Evasion and Intelligent Agents. An overview of various techniques used to solve the Pursuit-Evasion Problem has been done. And also the background used to facilitate the understanding of Multiagent systems has been proposed.

The next part talks about the different techniques proposed in this work. The techniques are gathered in two points of view: Offline search and Online Search. The first approach assumes that environment[2] is totally known to pursuers and evaders work as Nature. The loss of pursuers is related to whether the evader is caught, so pursuers have to take policies based on worst-base analysis to avoid regret. The second approach is more practical; it assumes that the environment is partially known to pursuers. The main pursuer (Master)[3] must take decision whether to request other agent or not for each position. The next part contains theoretical explanation of techniques applied to solve the Pursuit-Evasion problem for both cases.

[2]Environment is the workspace of agents. it is also called as building and world

[3]Master is the main agent that will explore the building; it generates the clearing strategy and requests other agents, etc. It is also called main agent.

Part I

Our Work

Chapter 2

World Conditionning

2.1 Introduction

This chapter explains the first steps that the environment passes to be used by the Offline and Online search. The first step is to decide how to model environment, and then the other steps consists of the walls and critical points detection. And then, the way how a vertex sees another vertex has been shown.

2.2 Environment

The used environment in the simulation work is a grid of patches; each patch is either black (empty space) or red (obstacle). Figure 2.1 shows examples of worlds (Also known as environments and buildings). The black areas are the reachable zones, and the red parts are the walls.

(a) World 0 (b) World 1

Figure 2.1: Examples of environments used in experiments

The next step is the edge detection, where the vertices and walls are detected and the vertex set is generated.

2.3 Edge detection

In this part, the environment is refined to find the critical points of the environment and the walls, the critical points, or vertices, are the blue numbered points, and the walls are the green lines (see Figure 2.2). this edge detection is done by the following algorithm (verifying the definition 7 in page 56).

Algorithm 1 Edge detection

```
For each patch p of the world do:
  C:= number of neighbour patches with pcolor = black
  If C>3 then p is a critical point (vertex)
    Else If C>0 then p is an edge (wall)
```

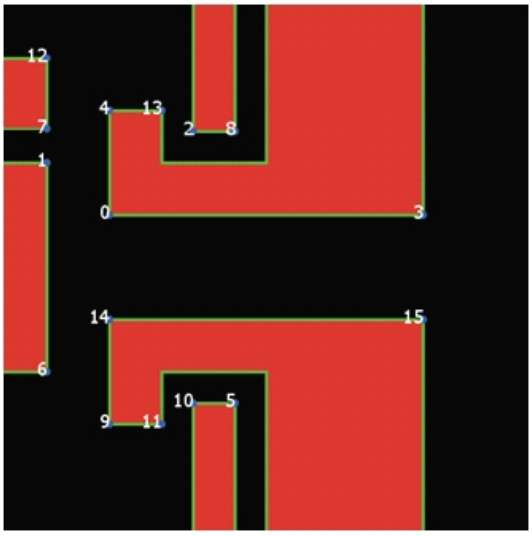


Figure 2.2: Edge detection

The next step helps to generate the visibility graph, this graph is the most important element in this work, and all strategies are based on this graph.

2.4 Visibility graph

Visibility graph is very important in this work, the vertices are connected by links, each link between two vertices defines that these two vertices see each the other. This visibility is verified by the following algorithm (applying definition 8 in page 56).

Algorithm 2 Visibility detection

```
  visibility := true
  Send tester from vertex 1 to vertex 2
  If the tester find in the way a wall (obstacle) then visibility := false
  Return visibility
```

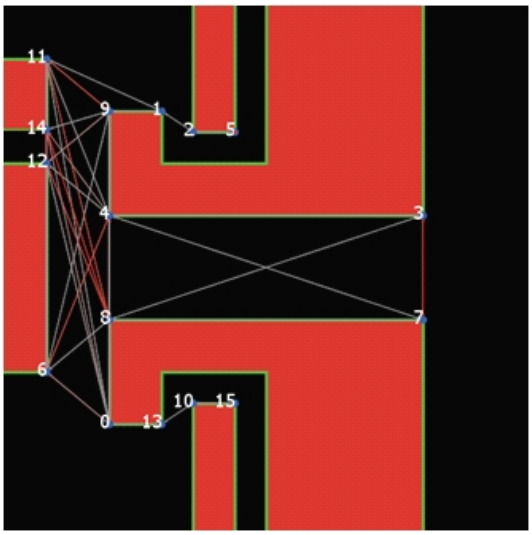


Figure 2.3: Visibility graph

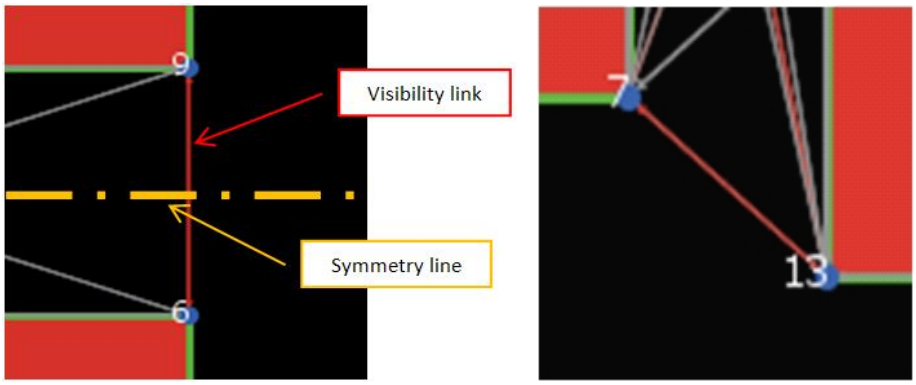
Figure 2.3 shows an example of environment after generating the visibility graph; the gray links are the simple visibility links, the red links are visibility links with direct view (as defined in definition 9 in page 56).

The next step consists of simplifying visibility graph by neutralizing useless vertices. Even though this part is facultative for the search, but it showed usefulness for the computation time.

2.5 Simplification of visibility graph

From Figure 2.3, we can observe that the graph is uselessly crowded by vertices and visibility links. So we decided to find how to simplify the visibility graph so the computation and the study become easier. We found that useless vertices are of two types with respect to their importance.

1. Vertices in the doors (figure 2.4(a)) can be simplified into one of them.
2. Vertices which are directly seen by at least another vertex (Figure 2.4(b)) can be neutralized in one condition, that this vertex is not *survivor* of door neutralization.



(a) Example of *door* (vertices 9 and 6)

(b) Directly visible vertex (vertex 13 directly sees vertex 7)

Figure 2.4: Cases of simplification of the environment

Definition 3. *A door is pair of vertices witch have the same set of visible vertices, it is obvious that the suppression of one of these vertices does not disturb the area graph. (see Figure 2.4(a)).*

$$V_{c6} = V_{c9} \tag{2.1}$$

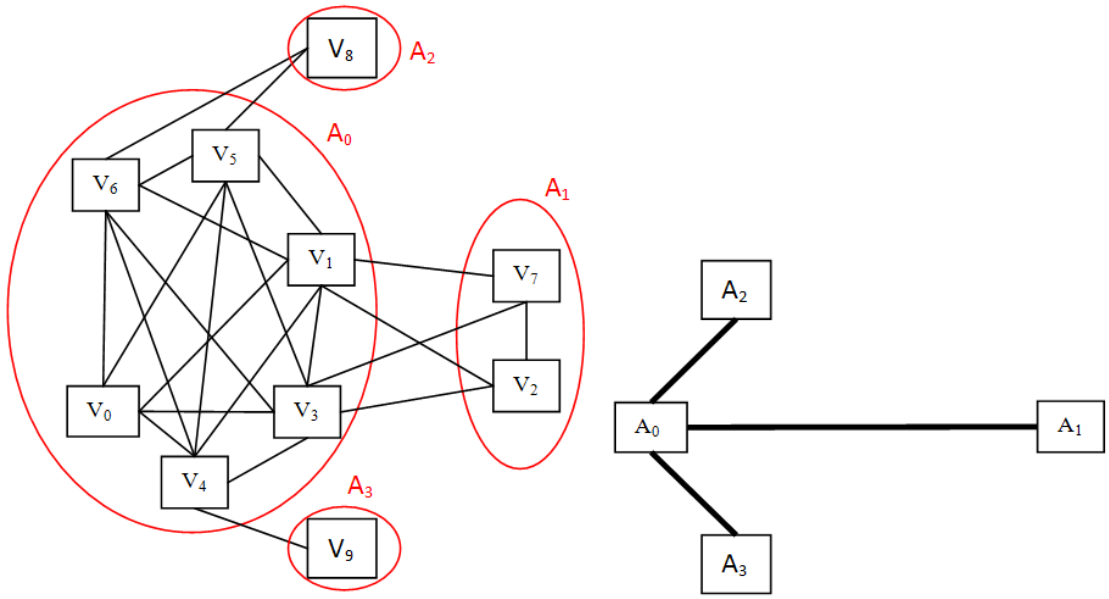
Definition 4. *A vertex sees directly another vertex when the first can see the adjacent edges of the second vertex, and its set of visible vertices is included in the set of the dominant vertex; generally, this vertex can be deleted. (see Figure 2.4(b))*

$$V_{c7} \subset V_{c13} \tag{2.2}$$

The next step is the creation of Area graph, this graph is generated using the simplified (or not) visibility graph. It is very helpful for the exploration strategies in the Offline search.

2.6 Area graph

This graph is generated from the visibility graph, this is by representing each connected sub graph by a node, and this node is named *Area*.



(a) Visibility Graph (b) Area Graph

Figure 2.5: Visibility and Area Graph corresponding to the environment in figure 2.1(a)

While we suppose that the environment is a priori known, the corresponding visibility graph of the environment shown in Figure 2.1(a) is represented in Figure 2.5(a), the set of vertices V_i is N, and the set of edges between these vertices is E, i.e. $N = \{V_0, \ldots, V9\}$ and $E = \{V_0V_1, V_4V_5, \ldots\}$. Obviously, the set of vertices where each vertex can see all the other vertices is in the same zone of the environment. So we can compute the Area graph Figure 2.5(b) of the corresponding visibility graph by replacing each complete sub graph by an area. In this example (Figure 2.5) there exists four different areas, A_0 (including the vertices V_0, V_1, V_3, V_4, V_5, and V_6), A_1 (including V_2 and V_7), A_2 (including vertex V_8), and A_3 (including vertex V_9).

The following algorithm shows how to compute connected sub graphs:

Algorithm 3 Generating connected sub graphs

```
function nodes %generates the complete sub-graphs
  ask links between vertices {
    set node recurse(both ends of links)
  }

function recurse(ls) %recursion to find complete sub-graphs
  let bout empty list
  let q compute-q(ls)
  for each element of the list q {
    let tmp1 ls with the current element of q inserted the end
    let tmp2 recurse(tmp1)
    set bout the largest set of connected sub graph between bout and tmp1
    set bout the largest set of connected sub graph between bout and tmp2
  }
  return bout

function compute-q(p) %computation used by "recurse"
  let tmp list of all vertices
  ask vertices of p {
      for each element of tmp {
        if (not out-visible-neighbor? (current vertex of tmp)) remove the vertex from tmp
      }
    }
  return tmp
```

2.7 Conclusion

This chapter explained a very important stage of environment search. This step prepares the environment for the experimental studies. The proposed refinement steps are applied on the simulation of the environment using *NetLogo*[1]. To simplify the study, the environment is assumed as set of rectangles. The study can be enlarged for examples of buildings with polygonal obstacles and even curved walls.

The next chapter talks about the first part of our study; the Offline Search, where the environment is globally percepted by the agent[2]; and the study of the optimal number of agents and optimal strategy of search is done before starting the search strategy. This study uses the previous chapter to model the environment to apply the techniques of search.

[1] *NetLogo* is a multi-agent programming language, it is detailled in Chapter 5 in page 26

[2] Mobile robot that will clear the building using other agents to guard some places

Chapter 3

Offline Search

3.1 Introduction

This chapter explains experiments and methods implemented for the Offline Search, in this part of study, the environment is completely known and the search strategy is based on the environment's scheme[1]. This study is composed of two methods: Worst-Case search, and Improved search. The first method is based on the number of critical areas in the Area Graph, and computes the number of agents needed to explore the building without any chances for the evader of escaping from an area to a cleared area. And the second strategy is the optimisation of the first one. In this method, the master begins to explore the environment and decides whether it has to request an agent or not.

3.2 Worst-Case Search study

In this part, optimisation of number of agents needed to explore the environment had the following stages:

1. Number of agents needed to explore the environment equals to the number of vertices[2].
2. The number of vertices has been reduced by neutralizing[3] useless vertices.
3. The number of needed agents became the number of areas[4] in the environment.
4. The number has been reduced into the number of critical areas[5] + 1.

[1] The Visibility Graph
[2] See Definition 7 in page 56
[3] See Section 2.5 in page 12
[4] For the definition of areas, see Definition 10 in page 56
[5] For the definition of critical areas, see Definition 12 in page 56

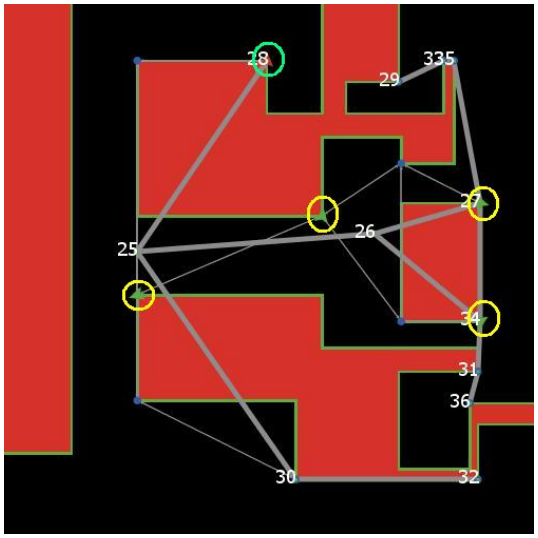


Figure 3.1: used agents for the worst case. The agent in green circle is the master, and agents in yellow circles are walkers (or guardians)

3.2.1 Principle

In this technique; the main idea is to put a guardian[6] in each critical area, this has to prevent evader of escaping from path[7] to another. Each environment is composed of interconnected paths, a path is successive areas, paths are connected by areas. So, the connected area is a critical (because it is connected to more than two areas). The path needs one agent to be decontaminated. So, when the master clears the building, it decontaminates the paths. And guardians prevents from contamination of a path from another contaminated path. This explains why the number of needed agents is the number of critical areas incremented of one.

3.2.2 Used functions

The Worst-Case technique is based on the following different functions:

Node function This function is used to generate areas; it adds for each visibility link vertices in the same area, an area is a connected graph[8]. To do that, it needs a function called "Recuse", this function is defined as follows:

Recurse function It computes the greatest set of connected vertices for the current visibility link.

Get-tree function This function has as duty to check all vertices and give to each area a level; this level is number which used to generate the area graph.

Simplify function This function uses the levelled visibility graph resulted from "get-tree", in which each vertex has a level in function of its area, it generates the area graph and the connections between

[6]Guardian is a supplemental agent that its duty is to guard its current area. Generally it is requested by the main agent

[7]See the definition of paths in Section A.3 in page 54

[8]see definition 10 in page 56

areas in function of visibility between areas[9].

3.3 Improved search study

In this case, a new concept has been introduced, which is the *fitness*. This fitness allows the agent to find the best branch to be explored based on the number of critical areas in this branch. For example (figure 3.1), suppose that we have an agent in area 25, after finishing areas 30 and 32, to select the next area that will be visited in such way that the global area will be cleared, this is a very delicate choice. If it goes into area 26, it will need another agent to guard this area, at the same time that an agent has to stay and guard area 25. But if the agent goes to 28 and returns to 25 and then goes to 26, then the global area 25[10] will be completely cleared.

3.3.1 Principle

This technique is based on the assumption that the search must always start from a leaf[11], or from area with minimal number of connected areas if there are no leaves. The clearing strategy is tested for each leaf, and then the best start is chosen according to the optimal number of used agents. Each time an agent is requested for guarding a critical area and all available agents are used to guard areas, so a new agent must be requested and the number of needed agents will be incremented. And if an area is cleared and all its successors, and the master will retreat, then the number of currently used agents will decrease.

3.3.2 The Fitness

To choose the best next area, extra criterion is proposed, this criterion gives the number of critical areas in each branch of critical area 25, with the depth n.

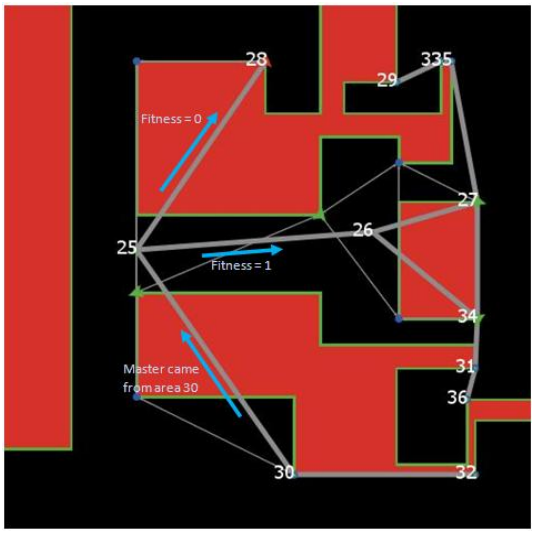


Figure 3.2: Scenario of selecting the best branch with the minimal fitness

The following is the algorithm of computing this fitness:

[9] using definition 11 in page 56
[10] Global area 25 is the branches connected to area 25
[11] A leaf is an area connected to one other area

Algorithm 4 The *fitness*

```
function fitness(ar0,ar,dp)
if (count branches > 2) then return 1
let tmp 0
if (dp > 0) {
  compute the fitness of each branch fitness(ar,current branch,(dp - 1))
  tmp = tmp + maximal fitness of branches
}
return tmp
```

where:

$ar0$ is the previous area

ar is the current area

dp is the depth of the search

This algorithm is used to compute the fitness of each area connected to the current zone. This fitness is used to choos the next vertex to be visited. The algorithm is based on a recursion function. This function computes the number of critical area according to the used depth.

3.4 Conclusion

This chapter studied the case where the agent has a total perception of the environment. The Master generates the motion strategy and the number (and location) of needed supplemental agents. This situation is applicable for the security of important buildings. This case has the problem for the unknown environments; the next study (Online Search) is based on this case.

In the next case, the agent has a partial perception of the environment, and it is dynamically changing with its position. The agent decides whether it must go on or go back, or even request another agent in function of its state[12]. This case is applicable for saving persons in dangerous buildings.

[12]This state is composed of its position, its history (previous states), its internal state, etc.

Chapter 4

Online Search

4.1 Introduction

This chapter explains experiments and methods implemented for the Online Search, in this part of study, the environment is partially known and the search strategy is dynamic and in function of the current situation. This study is composed of two methods: Direct search and Go-and-Back search. In the first method, the master checks the current situation and the visible vertices from its position, and decides whether request an agent or not or even go back to the previous position to check other areas. In the second strategy the master has to check all visible vertices seen from its position to ensure if he needs to request an agent or not.

4.2 Direct strategy

In this case, the master has to decide whether to request other agent or go forward for each position. In the case where the searcher detects more than one vertex at sight, it thinks that it is appropriate to request agent to *guard* this area, and then to go forward and continue clearing the building. This method is inspired from the human behaviour when the search routine must be done as fast as possible.

4.2.1 Principle

The main algorithm of decision used by the searcher is as follows:

Algorithm 5 Decision scenaio in Direct Strategy.

```
If there is one Vertex in CV then:
  Insert current vertex in stack
  Move to this new vertex

If there is more than one vertex in CV then:
  If there is no walker controlling this area (including this vertex) then Clone-walker
  Insert current vertex in stack
  Move to the first vertex in CV

If there is no vertex in CV then:
  Create tmp, the list of vertices in QL that are visible from V
  If tmp is not empty then:
    if tmp contains only one vertex then kill-walker
    Insert current vertex in stack
    Move to this new vertex
  If tmp is empty then:
    Kill-walker
    Go back
```

Where:

CV	is the list of visible vertices through Vertex V
stack	is the list of visited vertices, it is used to go back in case of finishing a path
Clone-walker	is to Create a walker on V to guard the area
QL	is the list of non-visited vertices through the time
V	is the current vertex
Kill-walker	is to kill walker (if it exists) in the same place with *Master*
Go-back	is to go back to the previous vertex in the stack

4.2.2 Scenario

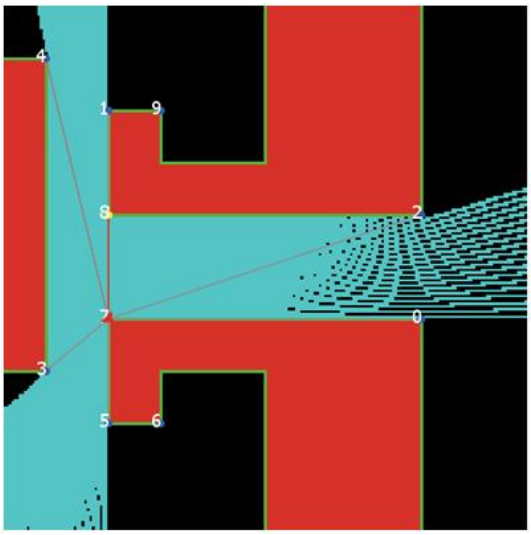


Figure 4.1: Case of visibility checking

For example, suppose the master is on vertex 7 (see Figure 4.1). In this situation, the CV contains the vertices 8, 2, 0, 3, 5, 4, 1. The master has to request a new agent (walker). And then go to another vertex (in this case, it will go to vertex 8). Before that, it puts vertex 7 in the Stack. And also, it puts other vertices of CV in QL. And then, Master does the same scenario for vertex 8.

4.3 Go-and-back strategy

In this case, the master visits all vertices in the current area to check critical vertices[1]. And then decides whether to request other agent or continue to other area in the building. This strategy ensures that if the current area deserves requesting walker or not. This technique is inspired of human behaviour, that, by instinct, checks the entire current area and then chooses the appropriate decision.

4.3.1 Scenario

Suppose the situation in Figure 4.1, the master must visit all vertices in CV (vertices 8, 2, 0, 3, 5, 4, 1). And then decides whether request a walker or continue to other vertex. In this case, there are 2 critical vertices (vertices 1 and 5) which have vertices behind them.

4.3.2 State machine

The used state machine is as follows:

[1] They are vertices connected to other vertices from other areas that are not visible to the master from the current area

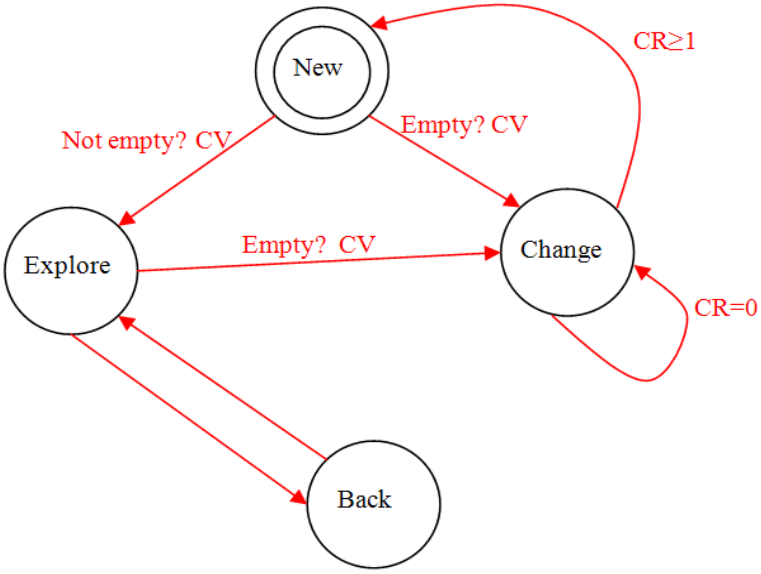


Figure 4.2: State machine of go-and-back strategy

CR is the number of critical vertices. The initial state is "New", Master will check visible vertices, and if there are new vertices (Not empty? CV) then the master will go to "Explore" state, elsewhere; it will "change" the area.

Here is the explanation of each state (from Figure 4.2) the agent can have:

- **New:** In this state, the agent is in the situation to generate the visibility set of the current vertex V_0.
- **Explore:** this state is used to the go-and-back status to check visible vertices V_i from the current vertex V_0.
- **Back:** this situation is alternated with "explore", but in this situation, the agent checks if the visible vertex V_i is critical or not. Critical means that this vertex is in contact with vertex(ices) not visible from vertex V_0.
- **Change:** this state is very important, here the agent decides whether to: request another agent, go to the next critical vertex, or go back to the previous vertex.

This state machine is executed for each time the main agent (the master) changes its position from a vertex to another vertex.

4.3.3 The Levelled Visibility Graph

The visibility graph has been checked by the master, and each time it goes into new zone (using "change"), it increments the level (or depth) of the current zone of vertices. For example, if the master starts from V_0, it gives the level 0 to this vertex, and it checks all visible vertices and gives level 0 to each one. Once the master finishes this zone, it goes to new zone and increments the level to 1 (Figure 4.3).

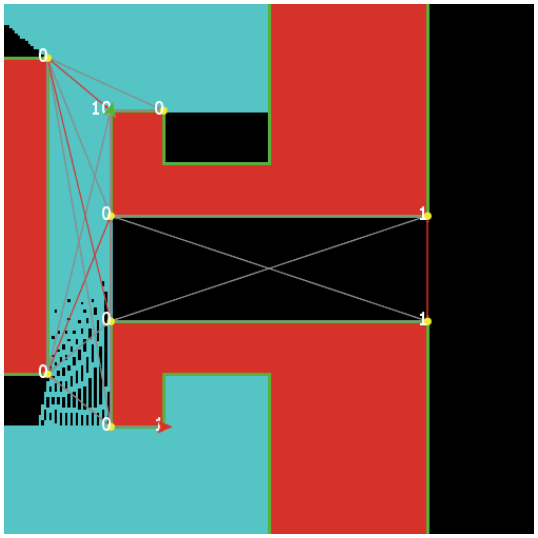


Figure 4.3: Levelled Visibility Graph

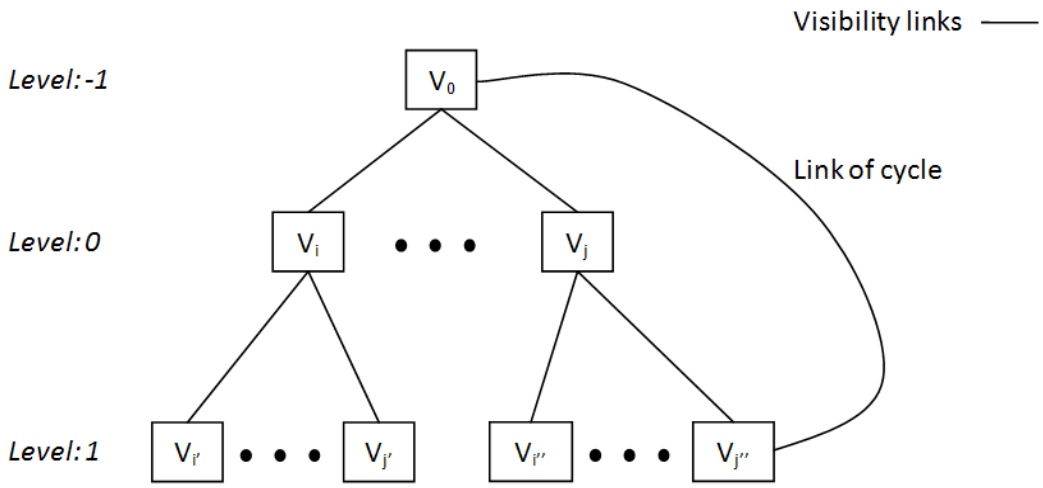


Figure 4.4: Diagram of Levelled Visibility Graph

Figure 4.4 shows the scheme of Levelled Visibility Graph, V_0 is the starting vertex; it has the level -1 (or 10 in experiments). The visible vertices from V_0 has all received the level "0", and for all visible links from a vertex with level "0", they receive the level "1", etc. As we can see, there is a visibility link from a level to an upward level (from $V_{j''}$ to V_0) this link designs a cycle. The detection of cycles by searching links from current level to an upward level helps the master to prevent missing the control of cycles. So once it detects a cycle, it requests an agent and asks it to stay at vertex V_0 to prevent the evader escaping and return to vertex $V_{j''}$ once Master returns to vertex V_j.

4.4 Conclusion

This chapter explains the used techniques to solve the problem of pursuit-evasion in unknown environment. The two methods proposed for this part are based on the instantaneous perception of walls by the Master, so it has to decide the action to be done for each position. The first method (Direct strategy) is based on the current visibility list of the master, and the actions are found according to the

number of visible vertices from the current vertex. But in the second strategy (Go-and-Back strategy), the master has to visit all visible vertices from the current vertex and then decides the appropriated action according to the number of critical vertices.

The next part is the experimental work, where the experiments and tests done to verify the previous techniques for the Offline and Online search. This part begins with a definition of the used tool in these experiments which is *NetLogo*. After that, the experimental results are explained and detailed to verify the usefulness of techniques proposed to solve the Pursuit-Evasion problem.

Part II

Experimental Work

Chapter 5

Used tool

5.1 Introduction

This chapter gives a brief overview about the used tool in the experimental work. This tool exists since 1960's and has been optimised to simulate the behaviour of multiagent systems. It facilitates the simulation of mobile agents and helps to give orders to an agent or a set of agents at once. This framework is *NetLogo*.

This overview starts with the history and ancestry of *NetLogo*. Afterwards, a presentation of the *NetLogo* is done with some of its features. Moreover, an explanation of the components of *NetLogo* and main used breeds (turtles, links, and patches) is presented in this chapter. Finally the used environments in the experimental work that are modelled using *NetLogo* are presented and explained.

5.2 Some *Logo* History

Logo got its start in the 1960's at Bolt, and Newman, Inc., a company in Cambridge, Massachusetts, who worked with people from the Massachusetts Institute of Technology.

A couple of years later, they added small, round robot that was connected by wires to the computer. It looked sort like a turtle. Using a small keyboard, young people gave the turtle commands to make it go forward, back, left, and right, moving over a big piece of paper, drawing pictures as it moved along.

This was fine for awhile. But when personal computers became popular in the late 1970's, the National Science Foundation and Texas Instruments Incorporated both asked the MIT people to make *Logo* work on small computers.

Texas Instruments introduced the first commercial version of *Logo* in April, 1981. In January of 1982, the MIT version of *Logo* that had been developed for *Apple //* family of computers was introduced. Since then, there have been many more versions of *Logo* for just about all personal computers.

[25] has explained *Logo* not as a programming language, but as a family interactive tool about the fun exploring the scope of the imagination.

Several extensions have been done basing on *Logo*, such as, MSWLogo[1], StarLogo[2], StarLogo TNG[3] (The Next Generation), *NetLogo*[4], etc.

[1] `http://www.softronix.com/logo.html`
[2] `http://education.mit.edu/starlogo/`
[3] `http://education.mit.edu/drupal/starlogo-tng`
[4] `http://ccl.northwestern.edu/netlogo/`

5.3 What's *NetLogo*?

NetLogo is a multi-agent programming language and integrated modelling environment. *NetLogo* was designed in the spirit of the *Logo* programming language to be "low threshold and no ceiling", that is to enable easy entry by novices and yet meet the needs of high powered users. The *NetLogo* environment enables exploration of emergent phenomena. It comes with an extensive models library including models in a variety of domains such as economics, biology, physics, chemistry, psychology, and many other natural and social sciences. Beyond exploration, *NetLogo* enables the quick and easy authoring of models.

It is particularly well suited for modelling complex systems developing over time. Modellers can give instructions to hundreds or thousands of independent "agents" all operating concurrently. This makes it possible to explore the connection between the micro-level behaviour of individuals and the macro-level patterns that emerge from the interaction of many individuals.

NetLogo was designed and authored by Uri WILENSKY, director of Northwestern University's Centre for Connected Learning and Computer-Based Modelling. Development has been funded by the National Science Foundation and other foundations.

5.4 Components

5.4.1 framework

The framework is composed of three parts: Interface, Information, and Procedures.

Interface contains the commands and the environment. The Command can be button, slider, switch, etc. The Button executes commands The Slider allows to change the value of global variable. The switch allows to change the value of global variable that has only the value true/false. There is other commands can be Input, Monitor, Plot, Output and Note.

Information is used to explain the program features.

Procedures contain functions and the main program. The programming language is very near to human interaction, but with some Agent Oriented concepts, such as `ask turtles`, `turtles-own`, etc.[5]

5.4.2 Patches

They are basic elements of the environment, we can call them "pixels", they compose the matrix of the environment. Therefore, by pointing each Patch through its coordinates (x and y), we can ask it to do some tasks, such as change colour (`set pcolor`), and change variables declared at the top of the program in the section `patches-own`.

5.4.3 Turtles/breeds

They are agents in this framework, they can move, turn, change colour, etc. Turtles can have different variables declared in the section `turtle-own`.

[5]For more information about programming language of *NetLogo*, please visit *NetLogo* Home Page `http://ccl.northwestern.edu/netlogo/`

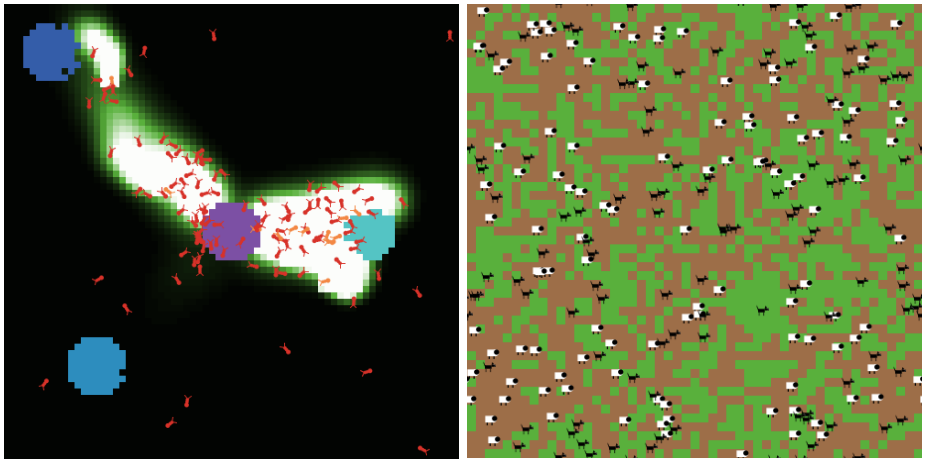
Breeds are other kinds of turtles. Therefore, we can create other types of agents by using the command `breed` so the command is for example: `breed [vertices vertex]` such that `vertices` is the plural name of the breed, and `vertex` is the name of one agent.

5.4.4 Links/breed-links

These breeds are used to connect between agents, these links can be directed or undirected. In this work, they are used in Graphs as edges. To create a new type of link, the used command is `directed-link-breed [visibles visible]` and `undirected-link-breed [lareas larea]`. The first command creates a directed link "visible" that has the plural "visibles" and the second command creates undirected link "larea" that has the plural "lareas".

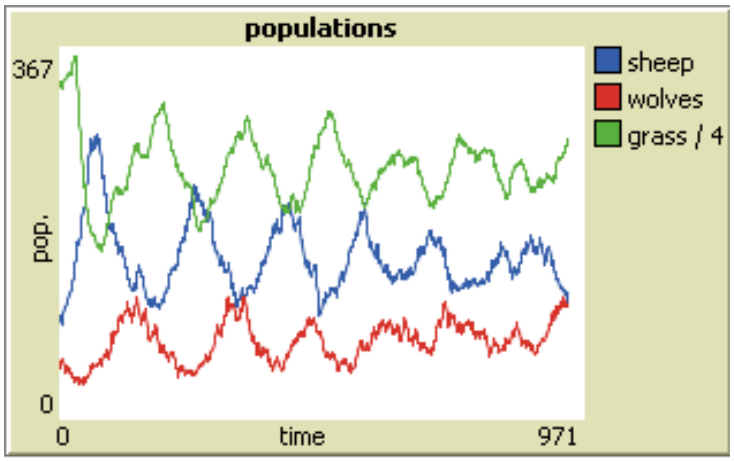
5.5 Some applications

NetLogo is applied in several domains of research - Biology, Artificial Intelligence and computer science, earth science, mathematics, etc. One of the interesting applications is Ants (Figure 5.1(a)), it simulates how a colony of ants foraging for food. Other application is the Wolf Sheep Predation (Figure 5.1(b)), where the natural equilibrium is simulated. Figure 5.1(c) shows the number of sheep and wolves and the amount of Grass in the environment.



(a) Ants foraging for food

(b) Worlf Sheep Predation

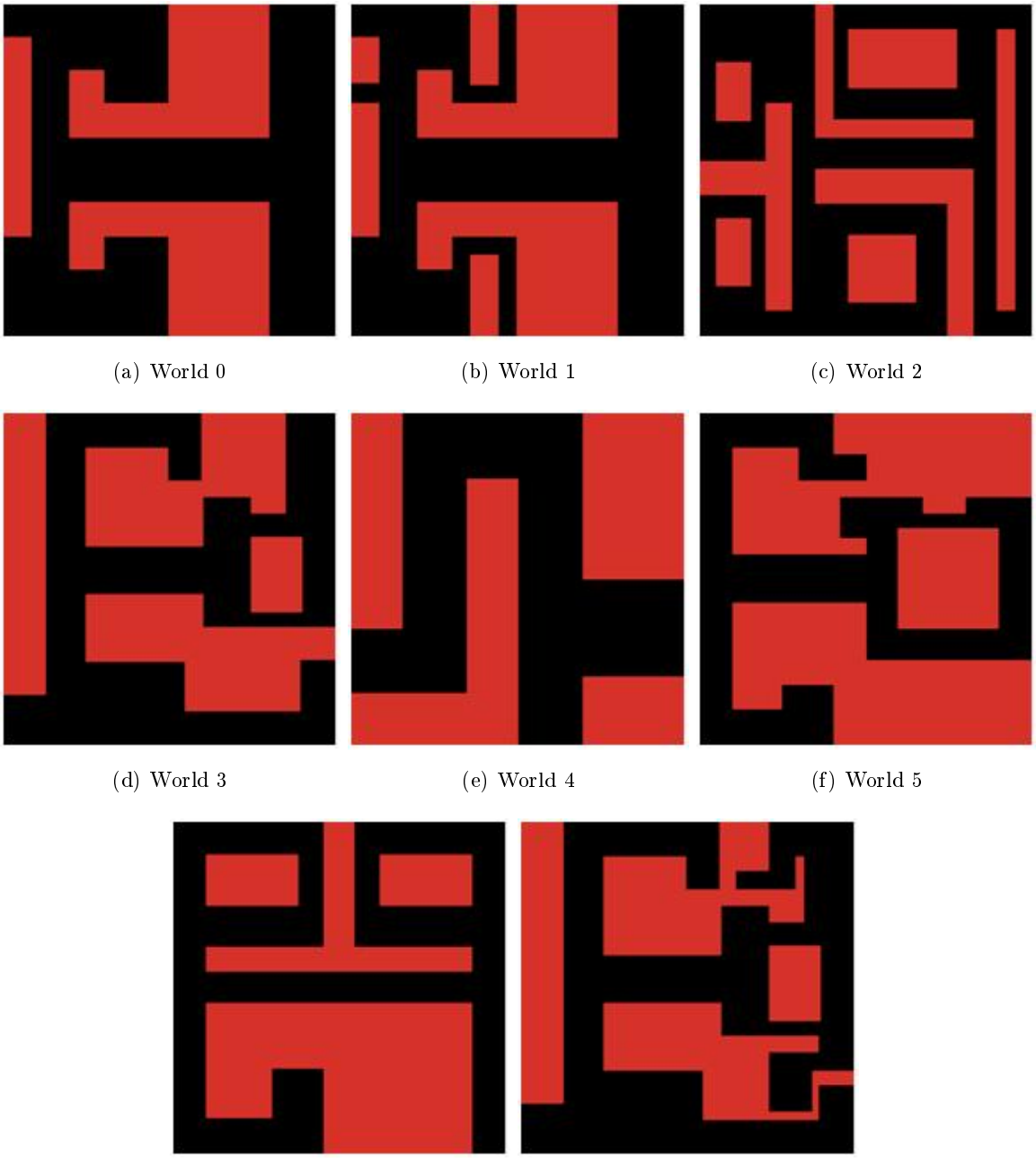


(c) Worlf Sheep Predation plots

Figure 5.1: Some examples of use of *Netlogo*

5.6 Used worlds

The used environments in this work have been modelled using *NetLogo*, the examples of environment used in experiments are as follows:

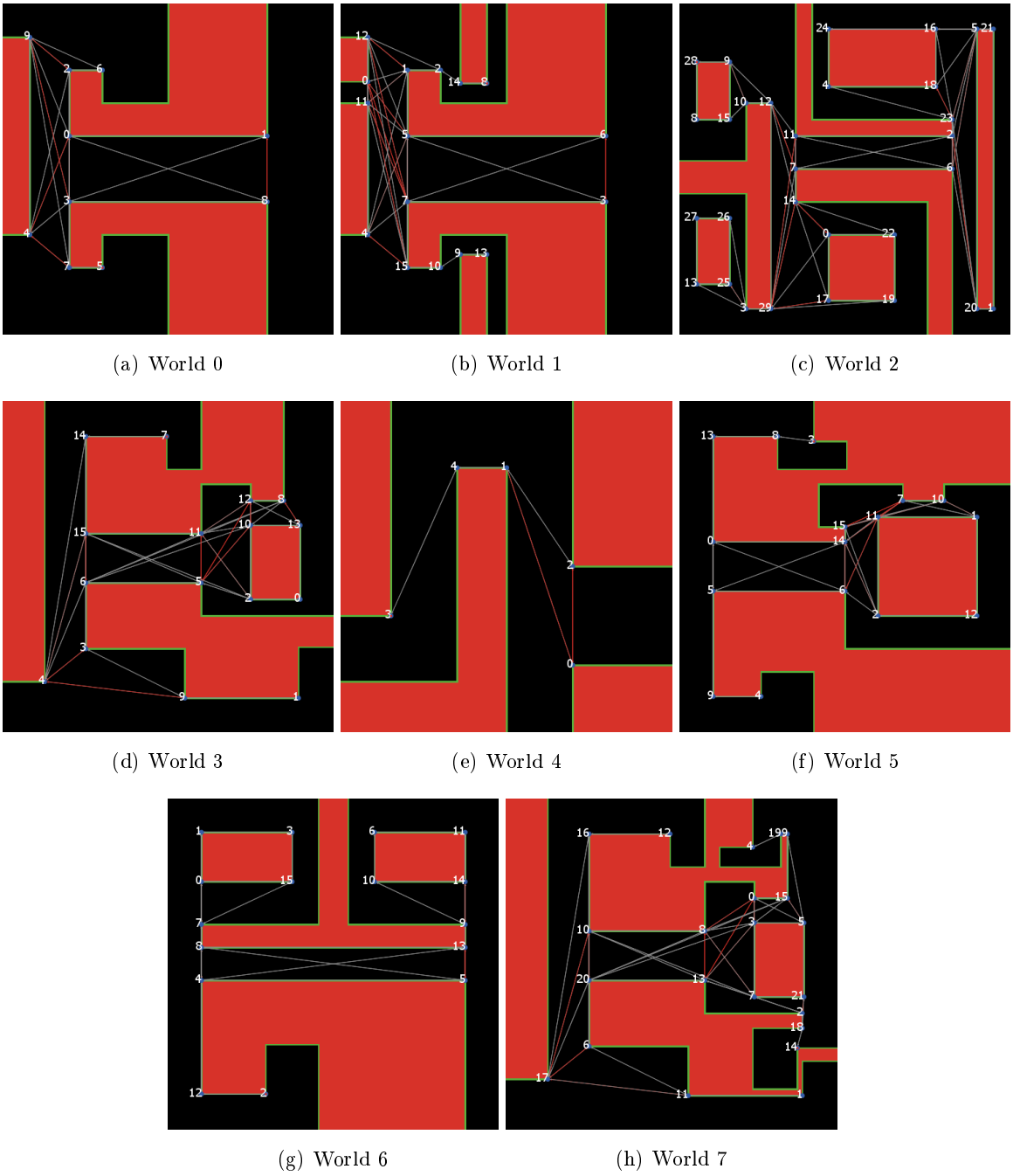


(a) World 0 (b) World 1 (c) World 2

(d) World 3 (e) World 4 (f) World 5

(g) World 6 (h) World 7

Figure 5.2: Used environments in the experiments

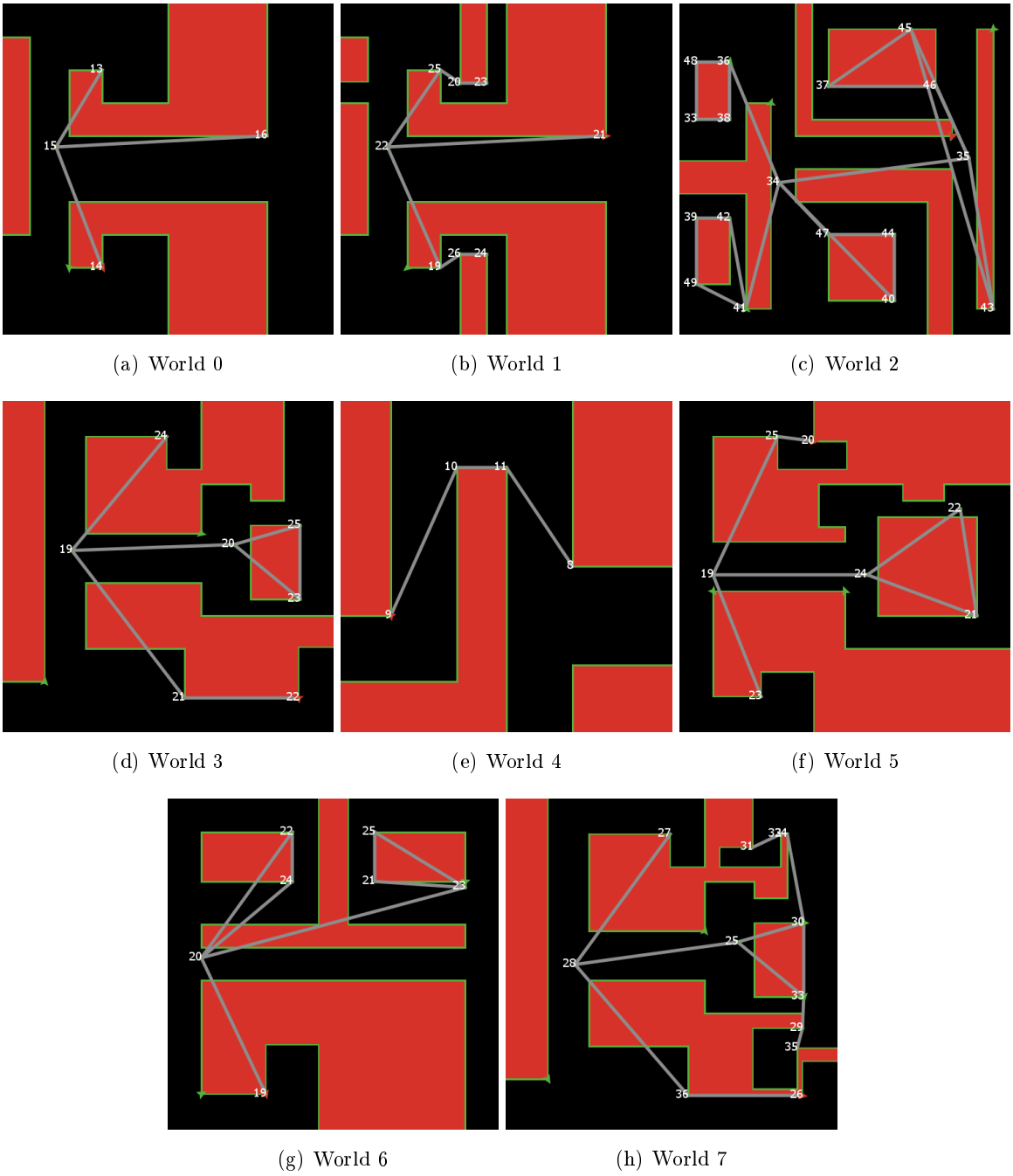


(a) World 0 (b) World 1 (c) World 2

(d) World 3 (e) World 4 (f) World 5

(g) World 6 (h) World 7

Figure 5.3: Visibility graph of the used environment in experiments



(a) World 0 (b) World 1 (c) World 2

(d) World 3 (e) World 4 (f) World 5

(g) World 6 (h) World 7

Figure 5.4: Area graph of the used environment in experiments

These environments are explained as follows:

- **World 0, World 1:** these worlds are simple cases of environment, the generated area graph is a tree with one root and 3 leaves[6].
- **World 2:** this environment is a critical case, in this world; the generated area graph contains interconnected cycles. This case is used to show the weakness of offline search for complicated graphs.
- **World 3, World 5:** these worlds are seen as trees with one root with 3 leaves and one cycle in

[6] A leaf is an end of a path; the leaf is a an area connected to one other area

a leaf. They are used to verify the techniques to solve the problem of cycles, especially in case of Online search.

- **World 4:** this case is a simple building which is simplified into one path.
- **World 6:** this environment is composed of a tree with 3 leaves and 2 cycles. This case is used to verify the usefulness of fitness function for the Improved technique in Online search.
- **World 7:** this case is a complicated tree, there is a cycle connected into two leaves and a root of two other leaves.

5.7 Conculsion

This chapter gave a general presentation of the used tool in this experimental work which is *NetLogo*. It can be seen that this framework makes the agent programming easier for cases of multiagent systems and societies of agents. It can be seen that this programming language is very near of naturally speaking language; this simplifies the commands and helps the user to get familiar with it faster.

The next chapter is where the simulations and experimental results are presented and explained. Several experiments are presented with comparison of results between techniques to understand the usefulness of each method.

Chapter 6

Simulation

6.1 Introduction

This chapter contains the simulations and experimental works. It concentrates on the tests of each technique. And comparison of results for Offline search and Online search to verify the usefulness of each method for the appropriate cases.

6.2 Offline Search

6.2.1 Usefulness of simplification of Visibility Graph

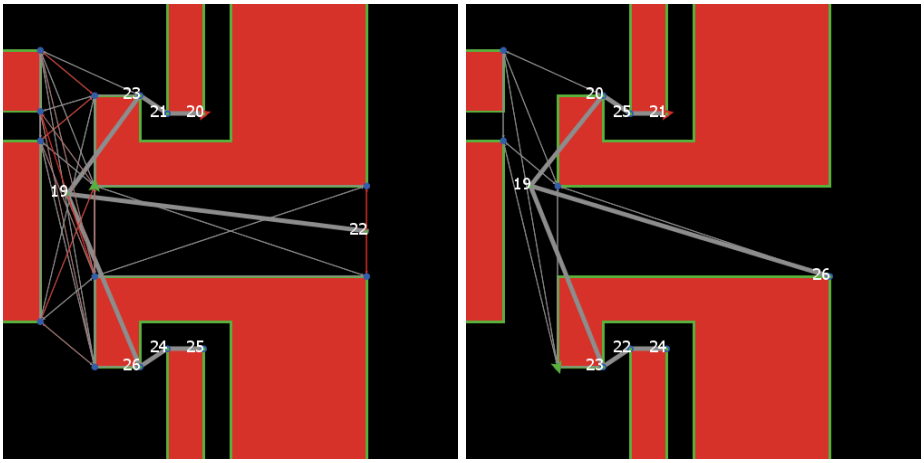
The idea of simplifying the visibility graph showed interesting results concerning the generating time of area graph. Table 6.1 is the experimental results found to compare computation time of several environments for the case of simplification and without simplification.

Example	**World without simplification**		**World with simplification**	
	Number of visibility links	**Time of areas computing in milliseconds**	**Number of visibility links**	**Time of areas computing in milliseconds**
World 0	46	109	12	≈ 0
World 1	80	7109	26	16
World 2	130	172	58	15
World 3	78	93	28	16
World 4	10	16	6	≈ 0
World 5	68	31	30	16
World 6	76	156	34	31
World 7	98	79	38	16

Table 6.1: Computation with and without graph simplification

It is shown that computation time has decreased severely after removing useless vertices, this is due to the simplification of areas, the algorithm used to generate area graph suffers of the problem that it needs to detect each connected graph through each edge contained in the connected sub graph. For example, let us see the "World 1" (Figure 6.1), the of computation time before and after neutralization of useless vertices has severely decreased. Even though, the number of visibility links is not so large. Please take a look at Area 19; it contains 8 vertices, i.e. 56 visibility links. The algorithm of area

graph has to compute the connected sub graph for each visibility link. This explains the very large time needed ($9.531s$) to generate the Area Graph for World 1.



(a) Without simplification (b) With simplification

Figure 6.1: Visibility and Area Graph of "World 1"

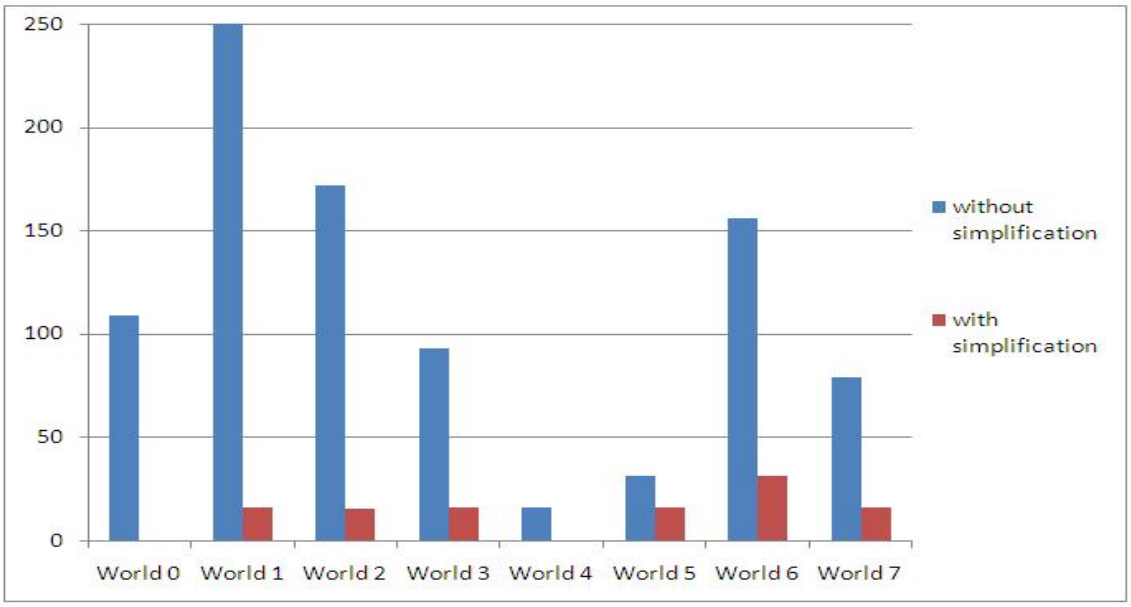


Figure 6.2: computation time (in milliseconds) with and without simplification of visibility graph

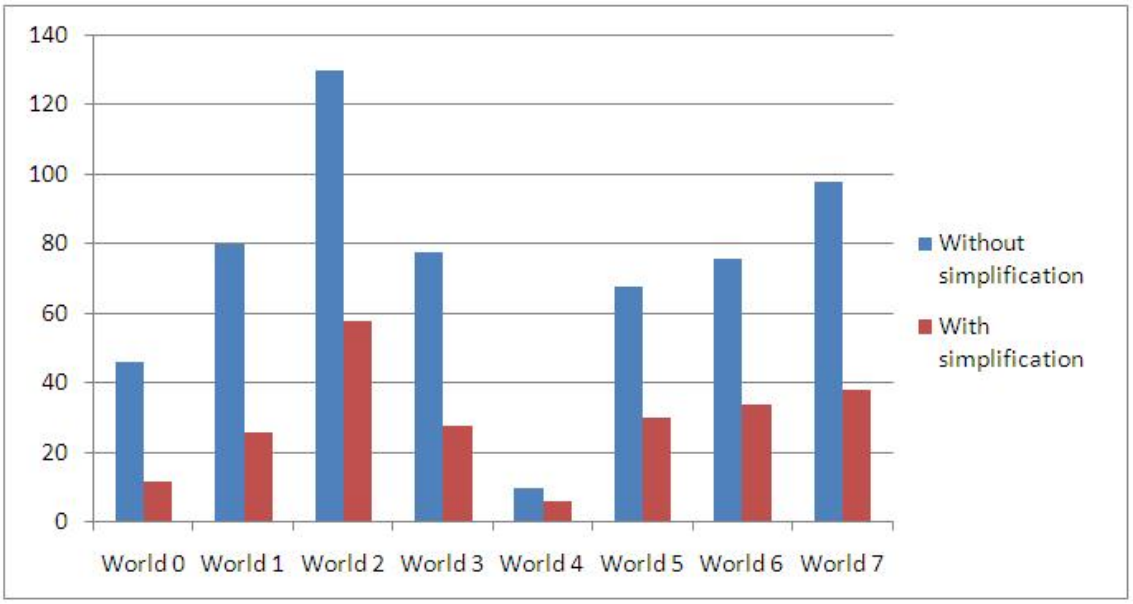
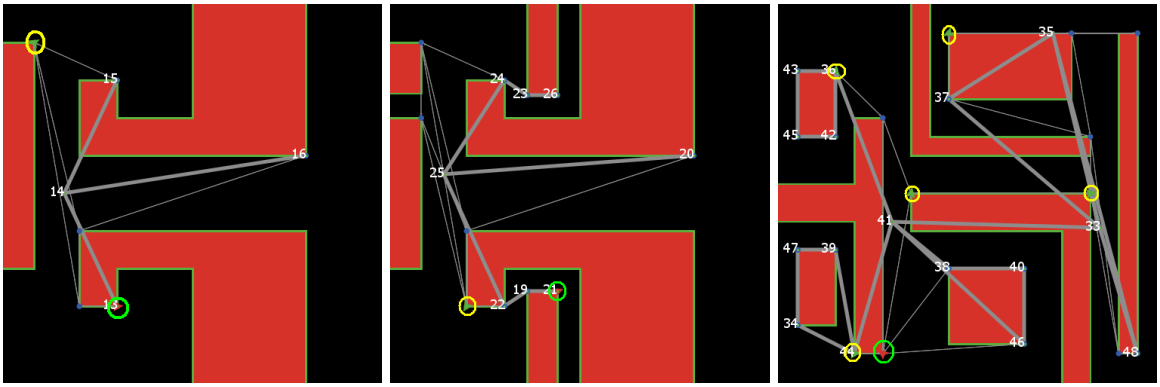


Figure 6.3: Number of visibility links in the environment before and after neutralization of useless vertices

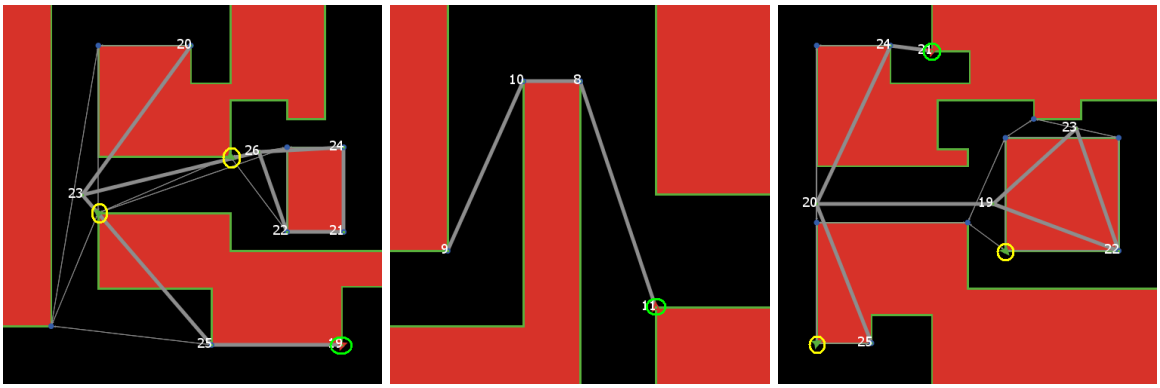
6.2.2 Worst-Case search

This method is based on the idea that each critical zone must be controlled by an agent. Firgure 6.4 shows the agents' deploying for each case of environment. The agents controling critical areas are in yellow circles, and main agent that will control the paths is in a green circle. Each agent must control vertices included in its area.

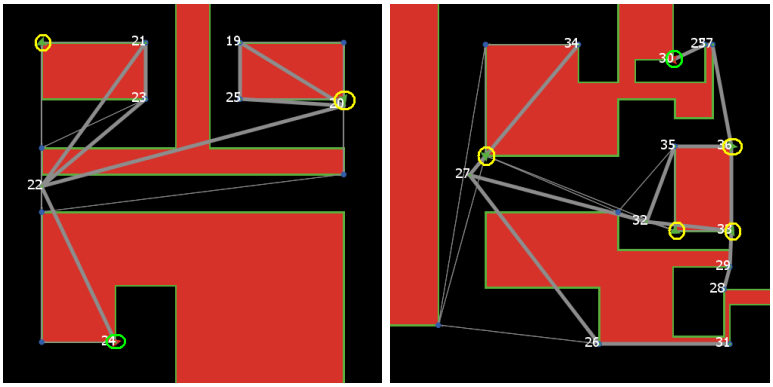
For case of World 2, the application of offline techniques is not possible because of the difficulty of the area graph, this graph contains several cycles interconnected which makes our proposed techniques non applicable for this environment.



(a) World 0 (b) World 1 (c) World 2



(d) World 3 (e) World 4 (f) World 5



(g) World 6 (h) World 7

Figure 6.4: Agents' deploying for each environment

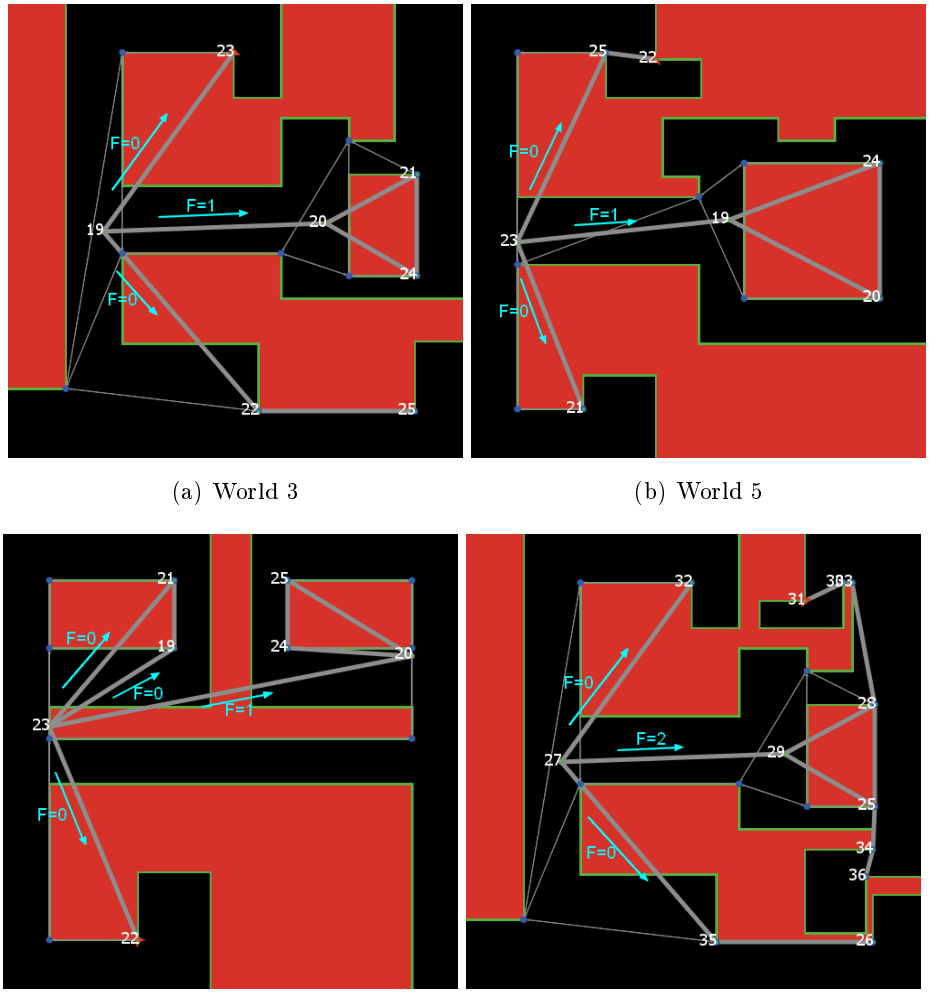
6.2.3 Improved search

This method is the optimisation of the previous method. For this case, the main agent starts exploring the environment from a leaf, and requests an agent for each critical area. Figure 6.5 contains different cases of environments to show the usefulness of the fitness[1]. The fitness for each branch is computed for the depth of 1.

For case of World 7 (Figure 6.5(d)), the fitness depends of the depth of fitness computation. For depth 1 the fitness is 2, and for depth 2 the fitness is 3. This is due to the number of critical areas seen from Area 27, e.g. for depth 1 the critical areas (which are counted in Fitness function) are 29

[1]The fitness function is defined in Section 3.3.2 in page 17

and 28, and for depth 2 the critical areas are 28, 29, and 25.



(a) World 3

(b) World 5

(c) World 6

(d) World 7

Figure 6.5: The Fitness for each branch for different environments

6.2.4 Comparison

Environment	Number of agents at worst case	Number of agents with Improved search
World 0	2	2
World 1	2	2
World 2	6	3
World 3	3	2
World 4	1	1
World 5	3	2
World 6	3	2
World 7	5	3

Table 6.2: Number of agents needed to cover the environment for different cases

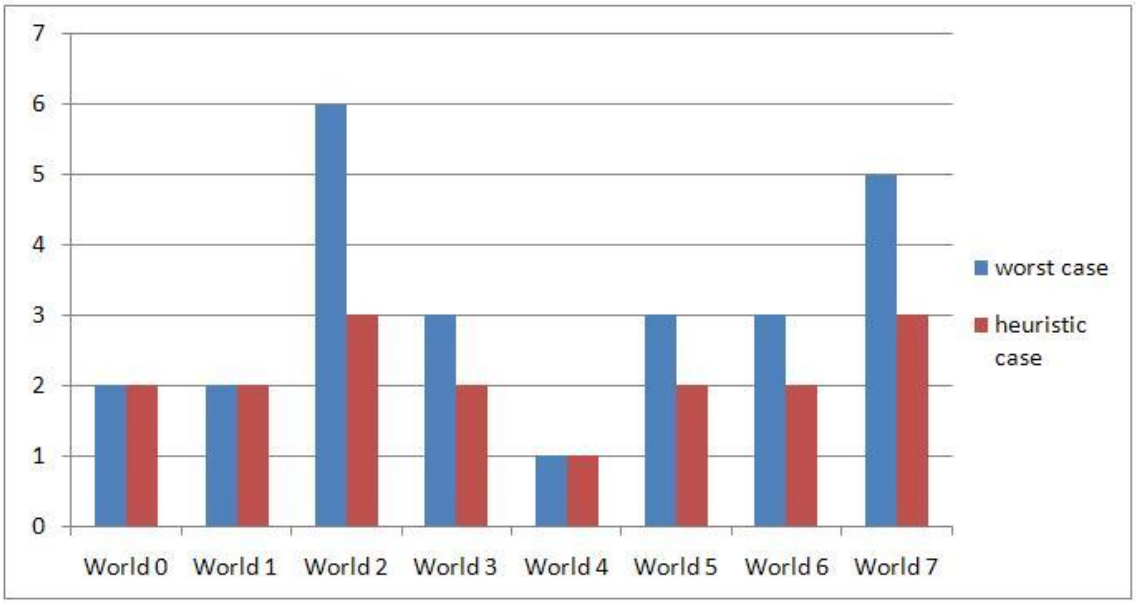


Figure 6.6: Number of agents used to cover the buildings

Figure 6.6 shows the number of agents needed to clear the building for each example. It can be seen that the number of agents has been decreased using the Improved search technique for several examples. But some ones, the number has not been changed, this is due to the nature of the environment; World 0, 1, and 4 are simple graphs with just one (or no) critical area. The usefulness of the second technique can be seen for examples with more than one critical area.

6.3 Online Search

6.3.1 Master's perception of walls and vertices

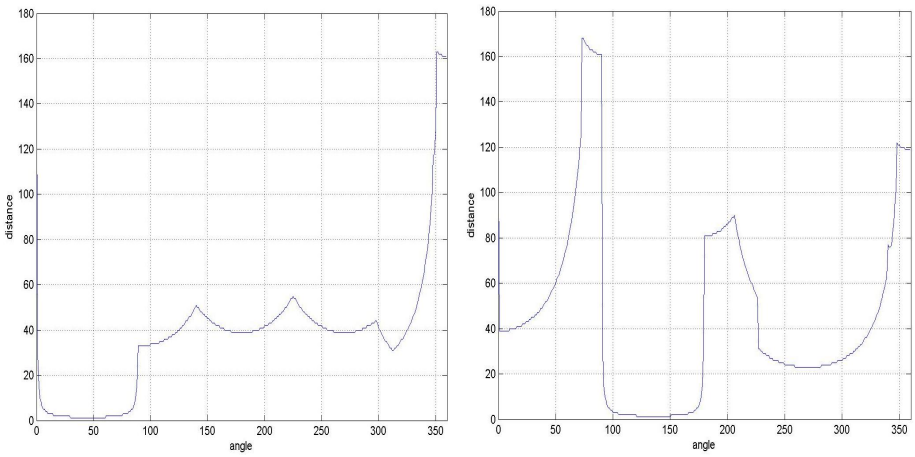
To percept walls, a method is proposed to detect vertices. This technique is based on laser to detect the distance from agent and wall, and then generate the virtual perception of current position of the master. This method is not used in the experimental work, the perception of vertices is supposed done before.

The simulation of the laser is done using *NetLogo*.

(a) Point 1 (b) Point 2

Figure 6.7: Laser-Based wall's simulation

Figure 6.7 shows the simulation of Laser-Based wall's perception using *NetLogo*. The red zones of the view are walls; the green dots are the percepted points of the walls using the Laser, here are two cases of agent's position, point 1 and 2.

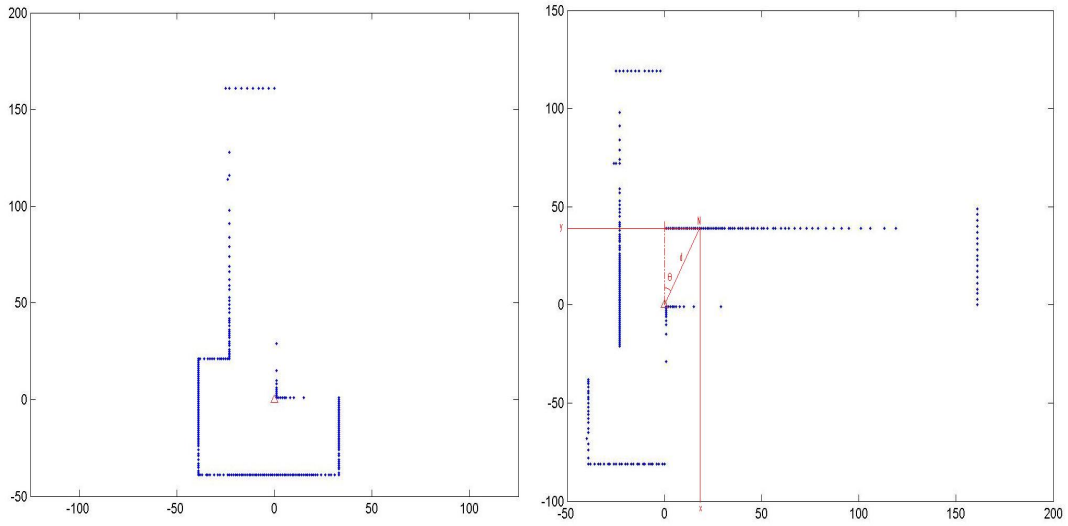


(a) Point 1 (b) Point 2

Figure 6.8: Laser-Based wall's perception

Figure 6.8 is the generated view from the laser sensor; it is a distance in function of angle. This angle can be according to the north, or any other reference frame. This is the signal which can be used to detect critical points[2]. These experiments did not study the Laser's perception, but this part can be a good part of future works.

[2]which can be transformed into vertices



(a) Point 1 (b) Point 2

Figure 6.9: Laser-Based wall's view

Figure 6.9 is the perception of walls according to the agent. These images are plotted on agent's *mind* using the angle and distance from the agent to the percepted wall for each point. As we can see, the view is not very clear, but the critical point of walls can be detected.This perception is built using the following trigonometric laws:

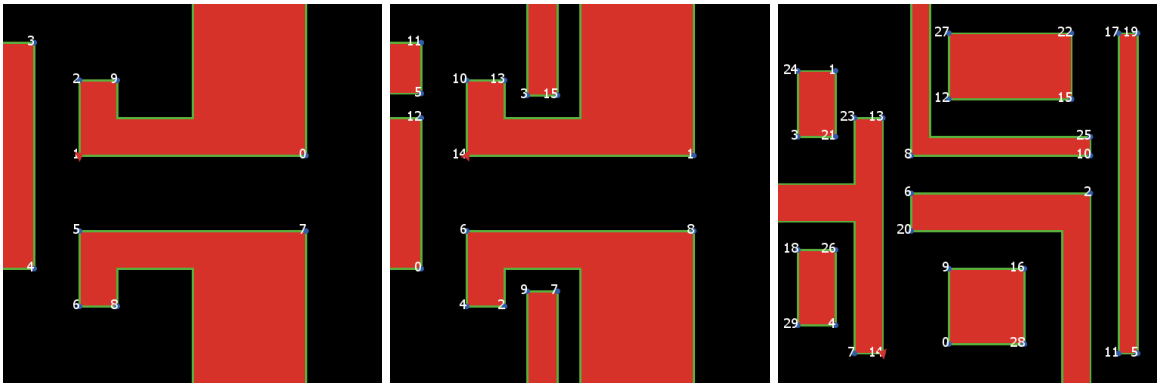
$$N(x,y) = \begin{cases} x = d.sin\theta \\ y = d.cos\theta \end{cases} \tag{6.1}$$

Where d is the distance from the agent to the wall; and θ is the angle from the referential north[3] to the angle of the corresponding point (see Figure 6.9).

6.3.2 Direct strategy study

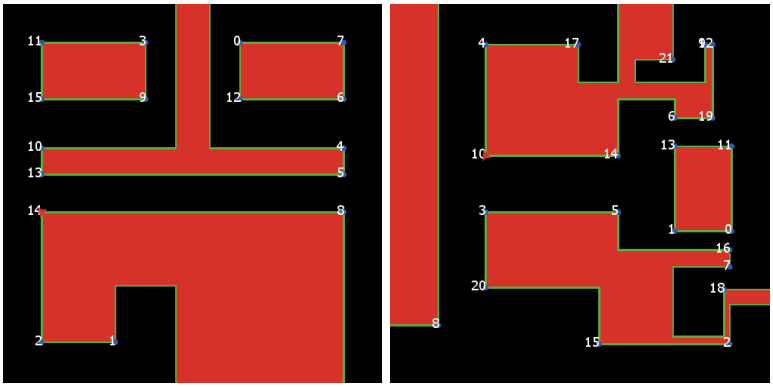
This method is applied for cases where the environment is not totally percepted, so the main agent must decide its actions according to its position and its state.

[3]The referential north for the NetLogo is the angle that the agent has when it is heading up; this angle is referenced by 0.



(a) World 0 (b) World 1 (c) World 2

(d) World 3 (e) World 4 (f) World 5



(g) World 6 (h) World 7

Figure 6.10: Direct search strategies

The visited visibles by the main agent, corresponding to Figure 6.10, are as follows in function of iterations:

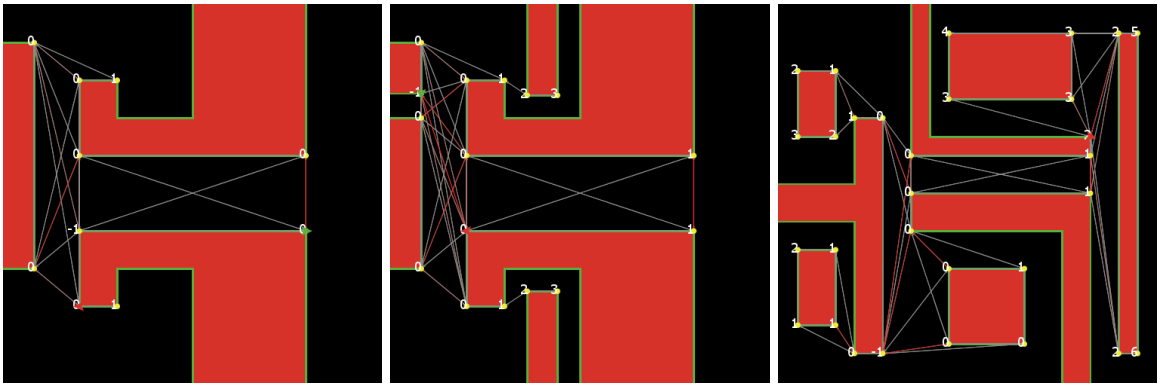
- World 0: [1 2 5 6 8 6 3 4 3 9 3 6 5 7 5]
- World 1: [14 5 6 12 4 2 9 7 9 2 4 11 13 3 15 3 13 11 4 12 6 1 6]
- World 2: [14 20 8 10 6 2 25 12 15 17 19 22 27 22 19 5 11 5 19 17 15 12 25 2 6 13 1 24 3 21 23 21 3 24 1 13 6 10 8 20 16 9 0 28 0 9 16 20 14 7 29 18 26]
- World 3: [3 6 12 8 14 0 5 4 5 9 15 9 5 0 1 10 1 7 13 11 13 2]

- World 4: [4 1 2 3]
- World 5: [10 14 2 1 6 11 8 13 4 9 0 7 15 7 12 5 3]
- World 6: [14 10 13 15 11 3 9 3 11 2 1 2 11 15 13 8 6 12 4 5 7 0]
- World 7: [10 8 15 20 3 19 11 0 16 5 1 6 14 13 14 6 1 5 16 7 18 7 16 0 11 12 9 21 9 12 11 19 3 4 17 4 3 20 15 2]

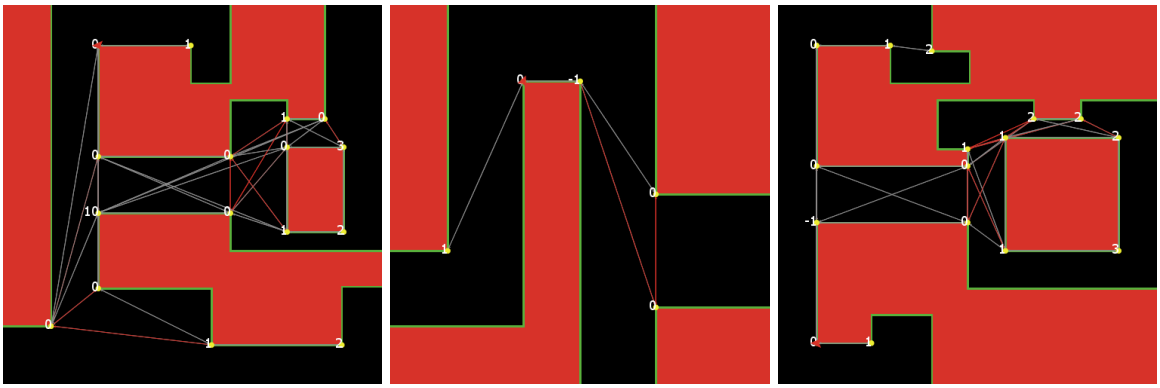
This method has proven a very interesting usefulness for the time of clearing the building, but it suffers of a big problem with the decision of requesting supplemental agents, this is seen for cases where the main agent sees more than two vertices, so it has to request another agent even if this place is in the same area (this case is more explained in section 6.3.4 in page 46).

6.3.3 Go-and-back strategy study

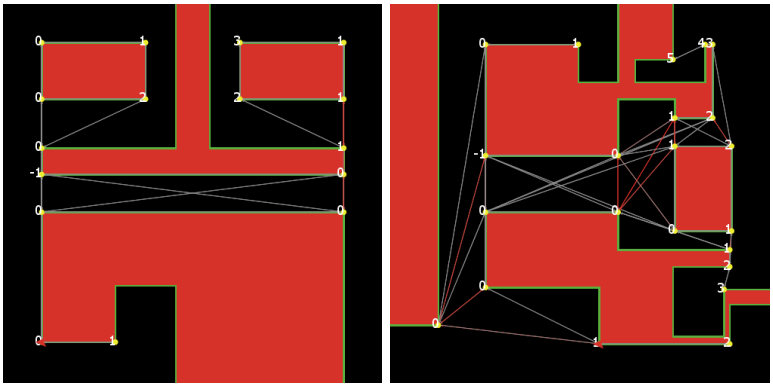
This technique is based on the idea of visiting all vertices in the current area to decide whether it needs a guardian or not. Figure 6.11 shows the levelled visibility graph for each case of study. Each vertex shows its level starting from the starting vertex which has the value of -1. World 0, 1, and 4 are simple examples to apply the technique. Cycles are, as defined in Section 4.3.3 in page 22, can be seen in World 2, 3, 5, 6, and 7.



(a) World 0 (b) World 1 (c) World 2



(d) World 3 (e) World 4 (f) World 5



(g) World 6 (h) World 7

Figure 6.11: Levelled Visibility Graph for each environment

This method ensures the optimization the use of supplemental agents, but it needs more time to visit the vertices in the current area. This method showed a better results for cases where the main agent is in an area which contains more than one vertex, so before deciding to request another agent, it visits all vertices in the current area, and decides that this area does not need a guardian (this critical case is more explained in section 6.3.4 in page 46).

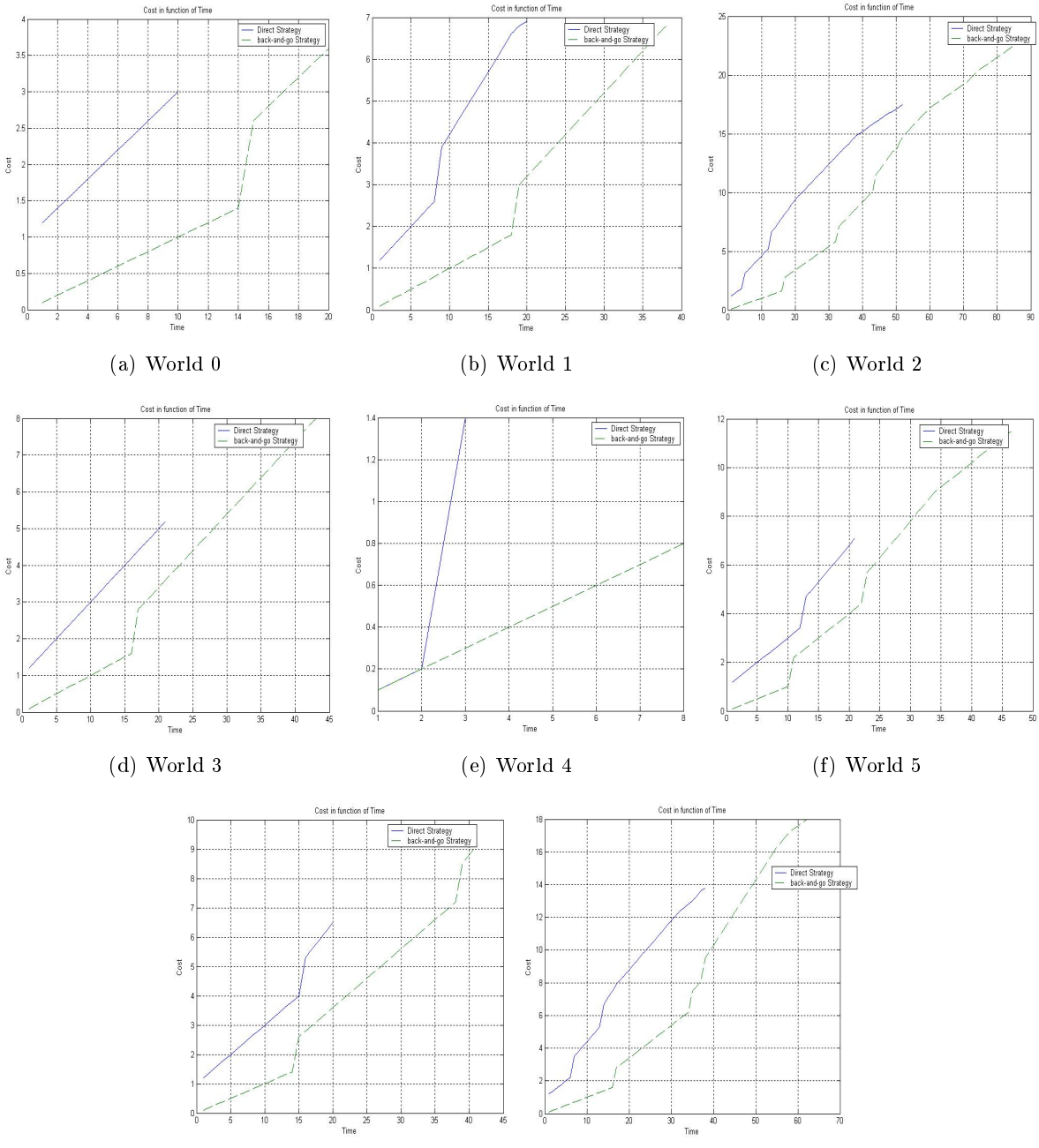
6.3.4 Comparison

A comparison between the two strategies has been done to see the usefulness of these methods in function of time and number of used walkers.

Cost computation

This cost criterion is computed as follows:

- Each agent needs +0.1 as cost for its *use*.
- Each new agent requested needs +1 as cost for its *recruitment*.
- Each agent that is no more needed is put as *spare*, so if there is need to a new agent, the cost of new agent is not added; the cost of spare agents is supposed null.



(a) World 0 (b) World 1 (c) World 2

(d) World 3 (e) World 4 (f) World 5

(g) World 6 (h) World 7

Figure 6.12: Cost in function of time for several examples blue line for direct strategy and dashed green line for go-and-back strategy

Figure 6.12 shows the comparison of cost in function of time. It can be seen that the first technique (Direct search) has given a lower cost and a faster time. But it suffers of the problem of risk that it requests more additional agents for areas that do not need guardians.

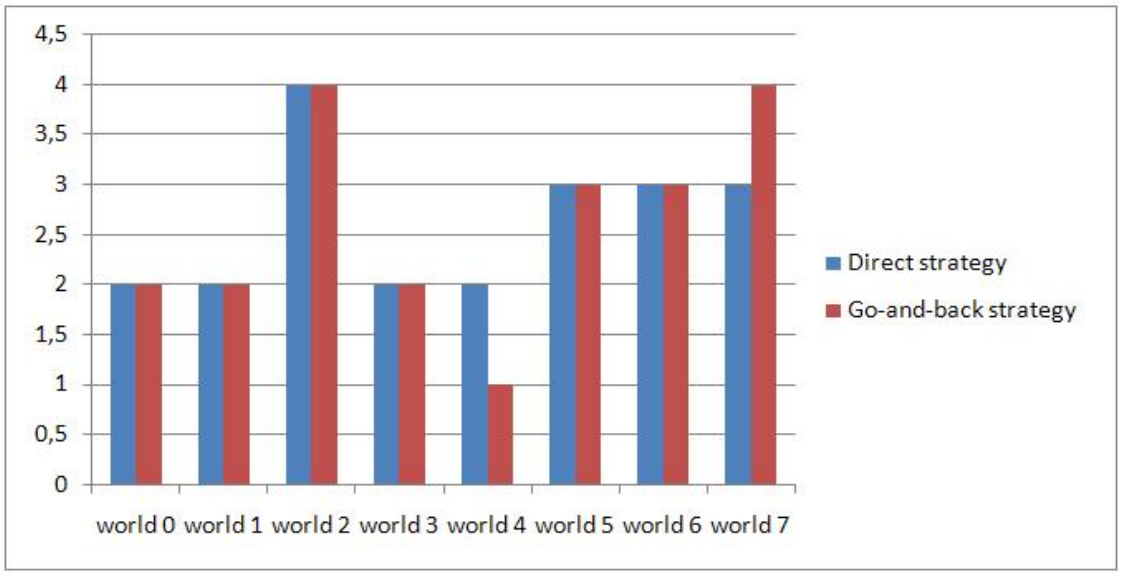
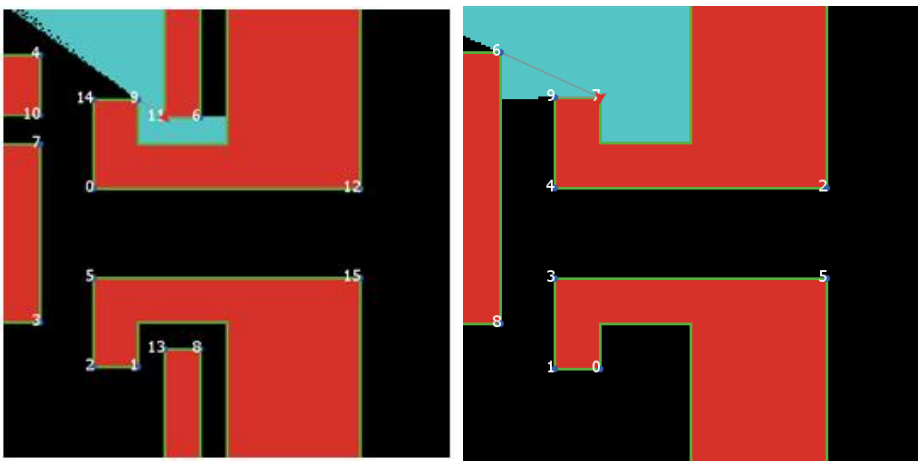


Figure 6.13: Number of used agents for the online search

Figure 6.13 shows a comparison of the used agents to clear environments for the two techniques. This number is for the same starting point; this number differs for environments where there is a critical case.

Critical case

There are cases where go-and-back strategy shows better results, for example (see Figure 6.14(a)). For direct strategy, the master will find 2 new vertices in CV (vertices V_9 and V_6), so it will automatically request a walker. But in go-and-back strategy, the master will check vertices V_9 and V_6, and then find that there is no need to request walker since there is only one critical vertex V_9.



(a) A case for World 1 (b) A case for World 0

Figure 6.14: Critical cases

6.4 Conclusion

This chapter contains the experimental work and results found to implement the techniques proposed to solve the Pursuit-Evasion problem for a totally-percepted and partially-percepted environment.

The proposed techniques gave interesting results for specific cases, but for some critical cases, one method would be more recommended than others. For example, for the offline search, where the environment is totally percepted, the Worst-Case method is better for cases where the search is delicate, and the evader must be found in any circumstances. And for cases where the number of used agents is more important, and must be smaller, the Improved technique is recommended. Moreover, for the Online search, where the agent percepts only the current visible area, the Direct method is recommended for cases of rescue, where the time is more delicate than the number of used agents. And the Go-And-Back strategy is recommended for cases where the number of agents is very important and must be as small as possible, and the time is not so important. Other idea is to mix the two methods for examples where the importance of time is variable.

The next part contains the final conclusion and perspectives for these techniques. It contains conclusions inspired from all this work, and ideas to optimise the proposed methods to solve more cases of environments.

Conclusion and Perspectives

In this part, we will talk about the conclusion taken from this work. And also, the perspectives for future works.

The proposed techniques can be used efficiently to solve the Pursuit-Evasion problem for different cases. Some techniques showed more efficiency for particular kinds of environment, and others showed some defaults for other cases.

For the Offline search, where the agent sees totally the environment and decides the optimal strategy of search, the Worst-Case technique has been useful for all cases, but it needs an important number of agents to clear the current environment, this is due to the need of an agent for each critical area. This increases the number of agents, and prevents the evader from escaping from path to another. the Improved technique has optimised the first technique by checking the entire Area Graph and find the optimal motion sequence and the positions of guardians. Unfortunately, The two methods were useless for cases where the environment is very complicated, especially for the case of environment with mixed cycles.

For the Online search, where the main agent has only the possibility to percept its current position and decides what to do according to what it sees, the Direct strategy gave good results for all kinds of environments, it showed fastness of clearing of building, but its main problem is the risk of using additional agents to guard areas that does not need to be controlled. The second technique, Go-And-Back strategy, which is based on the idea of visiting all vertices in the current area, was very efficient for clearing buildings, especially for environments with cycles, this is due to the idea of Levelled Visibility Graph, but it suffers of the problem of time needed to explore the whole building.

The applications of the pursuit evasion techniques have a promising future, the need of mobile robots to clear buildings is increasing with the increase of the use of technology in human life, the proposed Offline search techniques can be used efficiently for the security routines in important buildings like museum and mall. The idea of guardians in each critical area for the Worst-Case technique can be replaced by security cameras that detect the evader and inform the main agent of its position. The Improved technique can be used in cases where the used mobile robots are expensive and the number of used agents must be as small as possible. The techniques proposed to solve Online search problem can be used for the safety and rescue efforts in buildings on fire or chemically contaminated areas, this is because of the no need of building's plan to generate the search strategy. The Direct search strategy can be used for cases where the time is the most important criteria, and the Go-And-Back strategy can be used for cases where the time is not so important, but the number of needed agents must be as small as possible.

This work opens paths for several research topics; the first idea is the combination of the two parts of work, a global perception of the environment and a local instantaneous perception of current location. Furthermore, the optimization of the two techniques for the offline search to solve the problem for environments with complex area graphs, complicated wall forms (curved, circular, etc.). For the online search, the perception layer must be studied more to apply the laser based technique, and search other techniques to detect walls and vertices. In addition, the insertion of random supplemental information can be studied. For example, the case where the evader makes noise that the agent detects and estimates the position of the evader. Further study on these techniques to improve their results to be used for other cases of environments is recommended to enlarge the applications of Pursuit-Evasion search techniques on human life.

Bibliography

[1] Michah Adler, Harald Racke, Haveen Sivadasan, Christian Sohler, and Berthold Vocking. Randomized pursuit-evasion in graphs. *Cambridge University Press*, 12:225–244, 2003.

[2] Dana H. Ballard and Christopher M. Brown. Computer vision. *Prentice Hall*, II:523, 1982.

[3] Ramon Castano Barber, M. Mata, M.J.L. Boada, J.M. Armingol, and Miguel A. Salichs. A perception system based on laser information for mobile robot topologic navigation. *Proceedings of the 28th Conference on Industrial Electronics, Control and Instrumentation*, pages 2779–2784, 2002.

[4] Tamer Basar and Geert Jan Olsder. *Dynamic non-cooperative game theory second edition.* Academic Press, San Diego, CA, 1995.

[5] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Trans. Systems, Man, and Cybernetics*, (19):1179–1187, 1989.

[6] Peng Cheng. A short survey on pursuit-evasion games. *Department of Computer Science, University of Illinois at Urbana-Champaign*, 2003.

[7] David Crass, Ichiro Suzuki, and Masafumi Yamashita. Searching for a mobile intruder in a corridor–the open edge variant of the polygon search problem. *Internation Journal of Computational Geometry and Applications*, 1994.

[8] James L. Crowley. World modeling and position estimation for a mobile robot using ultra-sonic ranging. *IEEE Conference on Robotics and Automation*, 3:1574–1579, 1989.

[9] Xiaotie Deng, Tiko Kameda, and Christos H. Rapadimitriou. How to learn an unknown environment i: The rectilinear case. *Journal of the ACM*, 45:215–245, 1998.

[10] Reinhard Diestel. *Graph Theory.* Springer-Verlag Heidelberg, third edition, 2005.

[11] Fox Dieter, Henry Hexmoor, and Maja Mataric. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning and Autonomous Robots*, pages 29–53, 1998.

[12] Michael Drumheller. Mobile robot localization using sonar. *IEEE Transactions on Patten Analysis and Machine Intelligence*, 9(2):325–332, 1987.

[13] Brian P. Gerkey, Sebastian Thrun, and Geoff Gordon. Visibility-based pursuit-evasion with limited field of view. *The National Conference on Artificial Intelligence (AAAI 2004)*, 00:20–27, 2004.

[14] Leonidas J. Guibas, Jean-Claude Latombe, Steven M. La Valle, David Lin, and Rajeev Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry and Applications*, 09:471–494, 1999.

[15] Joao P. Hespanha, Hyoun Jin Kim, and Shankar Sastry. Multiple-agent probabilistic pursuit-evasion games. *In Procedure of 38th IEEE Conference on Decision and Control*, pages 2432–2437, 1999.

[16] Joao P. Hespanha, Maria Prandini, and Shankar Sastry. Probabilistic pursuit-evasion games: a one-step nash approach. *In Procedure of 39th IEEE Conference on Decision and Control*, pages 2272–2277, 2000.

[17] Geoffrey Hollinger, Athanasios Kehagias, and Sanjiv Singh. Probabilistic strategies for pursuit in cluttered environments with multiple robots. *In Proc. International Conf. on Robotics and Automation*, pages 3870–3876, 2007.

[18] Volkan Isler, Kannan Sampath, and Khanna Sanjeev. Randomized pursuit-evasion in a polygon environment. *IEEE Transaction on Robotics*, 2004.

[19] Volkan Isler, Dengfeng Sun, and Shankar Sastry. Roadmap based pursuit-evasion and collision avoidance. *Robotics: Science and Systems*, I:257–264, 2005.

[20] Johan Larsson and Mathias Broxvall. Fast laser based feature recognition. 2005.

[21] Johan Larsson, Mathias Broxvall, and Alessandro Saffiotti. Laser based intersection detection for reactive navigation in an underground mine. *In Proc. of the IEEE/RSJ Int Conf on Intelligent Robots and Systems (IROS)*, pages 2222–2227, 2008.

[22] Steven M. LaValle, David Lin, Leonidas J. Guibas, Jean claude Latombe, and Rajeev Motwani. Finding an unpredictable target in a workspace with obstacles. *In Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 737–742, 1997.

[23] Jae-Ha Lee, Sang-Min Park, and Kyung-Yong Chwa. Simple algorithms for searching a polygon with flashlights. *Information Processing Letters*, 81(5):265–270, 2002.

[24] John J. Leonard and Hugh F. Durrant-Whyte. *Directed Sonar Sensing fo Mobile Robot Navigation.* Kluwer Academic Publishers, 1992.

[25] Jim Muller. *The Great Logo Adventure: Discovering Logo on and Off the Computer.* Doone Publications, 1998.

[26] Damien Pallier and Humbert Fiorino. Coordinated exploration of unknown labyrinthine environments applied to the pursuitevasion problem. *AAMAS'05*, 2005.

[27] Torrence Douglas Parsons. Pursuit-evasion in a graph. *Theory and Applications of Graphs*, pages 426–441, 1976.

[28] Chavarkar Pradnya. Agent oriented programming. Technical report, Indian Institute of Technology, Bombay.

[29] Isaacs Rufus. *Differential Games.* John Wiley & Sons, 1965.

[30] Stuart J. Russel and Peter Norvig. Artificial intelligence: A modern approach. *Prentice Hall*, 1995.

[31] Yoav Shoham. Agent-oriented programming. Technical report, Stanford University, 1990.

[32] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 1:51–92, 1993.

[33] Sam Ge Shuzhi and Frank L. Lewis. *Autonomous Mobile Robots: Sensing, Control, Decision Making and Applications*. Taylor & Francis, 2006.

[34] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2004.

[35] Larry M. Stephens and Matthias B. Merx. The effect of agent control strategyon the performance of a dai pursuit problem. *Proceedings of the 1990 Distributed AI Workshop*, 1990.

[36] Kazuo Sugihara, Ichiro Suzuki, and Masafumi Yamashita. The searchlight scheduling problem. *SIAM Journal on Computing*, 19(6):1024–1040, December 1990.

[37] Sreenivas R. Sukumar, David L. Page, Andrei V. Gribok, Andreas F. Koschan, Mongi A. Abidi, David J. Gorsich, and Grant R. Gerhart2. Surface shape description of 3d data from under vehicle inspection robot. *Proc. SPIE Unmanned Ground Vehicle Technology VII*, 5804:621–629, 2005.

[38] Ichiro Suzuki and Masafumi S. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM J. Computing*, 21:863–888, 1992.

[39] Robert J. Urick. Principles of underwater sound. *McGraw-Hill*, 1989.

[40] Rene Vidal, Shahid Rashid, Cory Sharp, Shkernia Jin, and Kim Shankar Sastry. Pursuit-evasion games with unmanned ground and aerial vehicles. *in Proc. of IEEE ICRA*, pages 2948–2955, 2001.

[41] Rene Vidal, Omid Shakernia, Jin H. Kim, David Hyunchul Shim, and Shankar Sastry. Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation. *IEEE Transactions on Robotics and Automation*, XX:100–107, 2002.

[42] Talbot H. Waterman. Animal navigation. *Scientific American Library*, 1989.

[43] Gerhard Weiss, editor. *Multiagent Systems, A modern approach to Distributed Artificial Intelligence*. The MIT Press, 2000.

[44] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 12:115–152, 1995.

[45] Zhong Zhang. *Applications of visibility space in polygon search problems*. PhD thesis, Simon Fraser University, 2005.

Appendix A

Graph Theory

A.1 Graphs

A *graph* is a pair $G = (V, E)$ of sets such that $E \subseteq [V]^2$; thus, the elements of E are 2-elements subsets of V. To avoid notational ambiguity, we shall always assume that $V \cap E = \Phi$. The elements of V are the *vertices* (or *nodes*, or *points*) of the graph G, the elements of E are its *edges* (or *lines*). The usual way to picture a graph is by drawing a dot for each vertex and joining two of these dots by a line if the corresponding two vertices form an edge. Just how these dots and lines are drawn is considered irrelevant: all that matters is the information of which pairs of vertices form an edge and which not. [10]

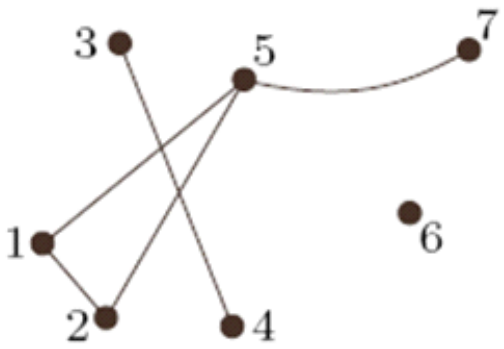


Figure A.1: Example of Graph $V = \{1, \ldots, 7\}$ with edges set $E = \{\{1,2\}, \{1,5\}, \{2,5\}, \{3,4\}, \{5,7\}\}$

A.2 The degree of a vertex

Let $G = (V, E)$ be a (non-empty) graph. The set of neighbours of a vertex v in G is denoted by $N_G(v)$, or briefly by $N(v)$. More generally for $U \subseteq V$, the neighbours in $V \backslash U$ of vertices in U are called neighbours of U; their set is denoted by $N(U)$.

The *degree* (or valency) $d_G(v)$ of a vertex v is the number $|E(V)|$ of edges at v; by our definition of a graph, this is equal to the number of neighbours of v. A vertex of degree 0 is isolated. The number $\delta(G) := \min\{d(v) | v \in V\}$ is the minimum degree of G, the number $\delta(G) := \max\{d(v) | v \in V\}$ its maximum degree. If all the vertices of G have the same degree k, then G is k-regular, or simply regular. A 3-regular graph is called cubic.

A.3 Paths and cycles

A path is a non-empty graph of the form

$$V = \{x_0, x_1, ..., x_k\} \qquad E = \{x_0x_1, x_1x_2, \ldots, x_{k-1}x_k\}$$

Where the x_i are all distinct, the vertices x_0 and x_k are linked by P and are called its ends; the vertices x_1,..., x_{k-1} are the inner vertices of P. The number of edges of a path is its length, and the path of length k is denoted by P^k. Note that k is allowed to be zero; thus, $P^0 = K^1$.

We often refer to a path by the natural sequence of its vertices, writing, say,$P = x_0x_1...x_k$ and calling a path from x_0 to x_k (as well as between x_0 and x_k).

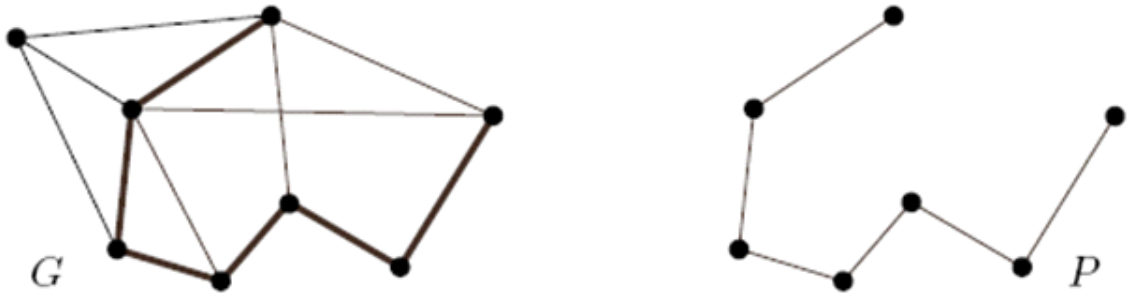


Figure A.2: A path P = P^6 in G

If $P = x_0x_1...x_{k-1}$ is a path and $k \geq 3$, then the graph $C := P + x_{k-1}x_0$ is called a *cycle*. As with paths, we often denote a cycle by its (cyclic) sequence of vertices; the above cycle C might be written as $x_0x_1...x_{k-1}x_0$. The length of a cycle is its number of edges (or vertices); the cycles of length k is called k-cycle and denoted by C^k. And edge which joins two vertices of a cycle but is not itself an edge of the cycle is a chord of that cycle. Thus, and induced cycle in G, a cycle in G forming an induced subgraph, is one that has no chords – Figure A.3. [10]

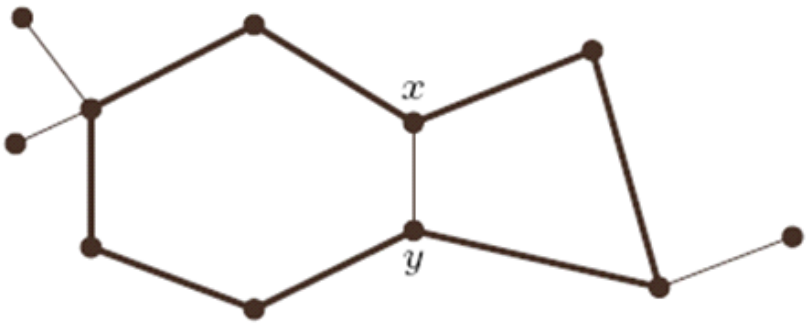


Figure A.3: A cycle C^8 with chord xy, and induced cycles C^6, C^4

A.4 Connectivity

A non-empty graph G is called connected if any two of its vertices are linked by a path in G. If $U \subseteq V(G)$ and $G[U]$ is connected, we also call U itself connected (in G). Instead of "not connected"

we usually say "disconnected".

the following definition is adapted from [10]

Definition 5. *The vertices of a connected graph G can always be enumerated. Say as $v_1, \ldots, v_n$, so that $G_i := G[v_1, \ldots, v_i]$ is connected for every i.*

Let $G = (V, E)$ be a graph. A maximal connected subgraph of G is called component of G. Note that component, being connected, is always non-empty; the empty graph, therefore, has no components. [10]

Figure A.4: A graph with three components and a minimal spanning connected subgraph in each component

A.5 Cut vertices

A vertex which separates two other vertices of the same component is a cutvertex, and an edge separating its ends is a bridge. Thus, the bridges in a graph are precisely those edges that do not lie on any cycle. [10]

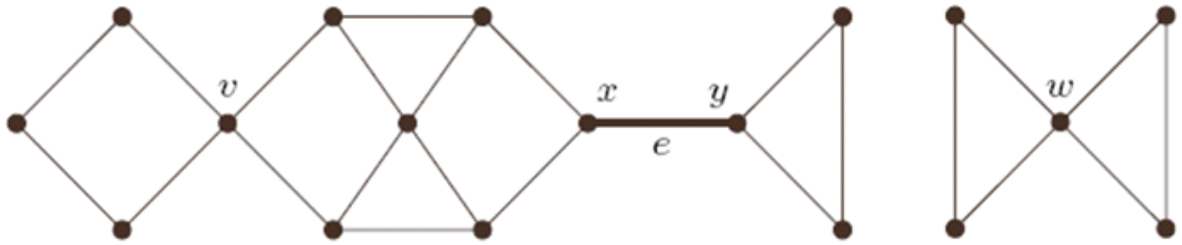


Figure A.5: A graph with *cutvertices* v, x, y, w and bridges $e = xy$

A.6 Trees and forests

An acyclic graph, one not containing any cycles, is called a forest. A connected forest is called tree. (Thus, a forest is a graph whose components are trees.) The vertices of degree 1 in a tree are its leaves[1]. Every nontrivial tree has a leaf – consider, for example, the ends of a longest path. This little

[1] Except that the root of a tree is never called a leaf, even if it has degree 1

fact often comes in handy, especially in induction proofs about trees: if we remove a leaf from a tree, what remains is still a tree. [10]

Figure A.6: A tree

This definition is adapted from [10]

Definition 6. *The following assertions are equivalent for a graph T:*

1. *T is a tree;*
2. *any two vertices of T are linked by a unique path in T;*
3. *T is minimally connected, i.e. T is connected but $T - e$ is disconnected for every edge ;*
4. *T is maximally acyclic, i.e. T contains no cycle but $T + xy$ does, for any two non adjacent vertices $x, y \in T$*

A.7 Definitions

Definition 7. *A vertex of F is a point from a wall that the angle formed by its adjacent edges is greater than π. [26]*

Definition 8. *A vertex sees another vertex when there is a link between these two vertices which is not cut by a wall.*

Definition 9. *A vertex sees directly another vertex when this one can see completely the second vertex, generally, this vertex can be deleted.*

Definition 10. *An area is a set of vertices which each one can see the other vertices.*

Definition 11. *An area is connected to another area when at least one vertex from an area sees a vertex from the other area.*

Definition 12. *A critical area is an area which is connected to more than another area.*