Computing and Informatics, Vol. 34, 2015, 233–253

AN EMPIRICAL STUDY ON COLLECTIVE INTELLIGENCE ALGORITHMS FOR VIDEO GAMES PROBLEM-SOLVING

Antonio Gonzalez-Pardo

Basque Center for Applied Mathematics (BCAM)
Bilbao, Spain
&
TECNALIA, OPTIMA Unit
E-48160,Derio, Spain
e-mail: antonio.gonzalez@uam.es

Fernando Palero, David Camacho

Escuela Politecnica Superior
Universidad Autonoma de Madrid
Madrid, Spain
e-mail: {fernando.palero, david.camacho}@uam.es

Abstract. Computational intelligence (CI), such as evolutionary computation or swarm intelligence methods, is a set of bio-inspired algorithms that have been widely used to solve problems in areas like planning, scheduling or constraint satisfaction problems. Constrained satisfaction problems (CSP) have taken an important attention from the research community due to their applicability to real problems. Any CSP problem is usually modelled as a constrained graph where the edges represent a set of restrictions that must be verified by the variables (represented as nodes in the graph) which will define the solution of the problem. This paper studies the performance of two particular CI algorithms, ant colony optimization (ACO) and genetic algorithms (GA), when dealing with graph-constrained models in video games problems. As an application domain, the "Lemmings" video game has been selected, where a set of *lemmings* must reach the exit point of each level. In order to do that, each level is represented as a graph where the edges store the allowed movements inside the world. The goal of the algorithms is to assign the

best skills in each position on a particular level, to guide the *lemmings* to reach the exit. The paper describes how the ACO and GA algorithms have been modelled and applied to the selected video game. Finally, a complete experimental comparison between both algorithms, based on the number of solutions found and the levels solved, is analysed to study the behaviour of those algorithms in the proposed domain.

Keywords: Collective intelligence, ant colony optimization, genetic algorithms, video games solving algorithms, Lemmings video game

Mathematics Subject Classification 2010: 68T20

1 INTRODUCTION

Bio-inspired computation has been widely used in different areas from combinatorial optimization problems to stochastic search in a huge number of application domains. From industrial or engineering applications [26] to theoretical developments [32], they have been applied to study new bio-inspired approaches able to deal with *NP-complete* or *NP-hard* problems [11]. In this kind of problems usually the problem solving process needs a huge amount of resources (such as computational effort or time) to find a solution. Some interesting examples of these applications are those based on *scheduling problems* [64], *constrained satisfaction problems* [19], or *routing problems* [55]; among others.

From the set of different methods and techniques that can be considered as bio-inspired (artificial neural networks, fuzzy logic, evolutionary computation and swarm-based intelligence), this paper will focus on the utilization of two of these techniques and their application for video games-based problems. Genetic algorithms (GA) have been selected as a representative example of an evolutionary computation (EC) [30, 34] method, whereas the ant colony optimization (ACO) algorithm has been employed as an example of a swarm intelligence (SI) [15, 31] approach.

The selection of these algorithms (GA and ACO) has been made taking into account two main characteristics. On the one hand, both types of algorithms work with a *population* of possible solutions that navigates through the solution space of the modelled problem. On the other hand, both algorithms need some kind of a *fitness function* to guide the algorithm to the optimum solution. These characteristics facilitate the comparison of both approaches through the utilization of the same fitness function for the selected application domain.

In the case of GA, each individual is evaluated by a fitness function that allows an individual evaluation. Then, those individuals with better fitness values have a higher probability for being selected for the reproduction phase [14, 52]. The operator that allows the generation of new individuals taking into account the

characteristics of the best ones is called the *crossover*. With this operator, new individuals' characteristics (i.e. *genotype*) are built by recombining their parents' genotypes. Finally, the *mutation* operator changes randomly some genes of the new individuals. Crossover and mutation operations provide the mechanisms for the exploitation and exploration of the problem space [17, 35].

Swarm intelligence algorithms, such as ACO, focus on the collective behaviour of self-organizing systems [33] where the iterations among individuals generate collective knowledge based on social colonies [42]. Some examples of this type of algorithms are *particle swarm optimization* [45], *bee colony optimization* [43], *bird flocking* [57] or *bacterial foraging* [25]. In all of these algorithms the initial population does not change, this means that there is not any generation of new individuals. Instead of this, the population travels through the solution space in order to obtain the best solution to the problem. In these approaches, the fitness function is not used to measure the quality of the population, but to measure the quality of the solutions found by the swarm. The fitness values are used to guide the whole population, or swarm, to the solution. This fitness function is usually designed as a part of the meta-heuristics used by this kind of algorithms.

The *Lemmings* game is a popular proven NP-hard puzzle game [23] that can be used as a benchmark for CI algorithms. In spite of the popularity that this game obtained in the 1990s, little research has been applied to it. Computational intelligence has just been applied to video games such as Mastermind [13], the Art of Zen [20], Ms Pac-Man [50], Tetris [18] or Mario Bros [60, 61].

The main contributions of this paper are related to the application of a classical ant colony optimization [22, 29] and a genetic algorithm to the Lemmings video game. To do that, both algorithms have been adapted to the selected video game. An adequate problem modelling has been designed and several meta-heuristics have been tested to determine whether a set of levels can be solved. Finally, this work tries to provide an empirical study on the influence that the environment (in this case, the Lemmings level) exerts over these different algorithms. In order to do that the performance of a genetic algorithm (GA), an ant colony optimization (ACO) and a heuristic for the ACO will be analysed based on the number of solutions found and complexity of the levels solved [51].

2 COMPUTATIONAL INTELLIGENCE IN VIDEO GAMES

Traditionally, the utilization of techniques from the artificial intelligence (AI) area and other sub-areas such as computational intelligence has been employed in classical board games like chess, checkers, kalaha, go, othello, Tic-Tac Toe, etc. [49, 51]. However, the high impact of video game industry has generated an increasing interest in the practical utilization of AI and CI techniques in commercial video games. Currently, there exist a wide number of international conferences that propose different competitions, or challenges, whose main goal is based on the utilization of AI and CI techniques to solve problems like: automatic solving levels, definition of

autonomous non-player characters (i.e. autonomous bots), automatic generation of levels, etc.

Maybe, the most popular competitions are those from the IEEE International Conference on Computational Intelligence and Games (`www.ieee-cig.org`). This conference has promoted different competitions in the last ten years based on popular video games like Mario Bros, Ms-Pacman or StarCraft. The basic idea behind these competitions is based on the utilization of a video game platform that can be used to integrate AI/CI algorithms to test their behaviour in a particular challenge.

2.1 Current Video Games Platforms & Competitions

This section will briefly describe the main and most popular video games competitions offered by CIG conference. In the last edition (CIG'2013) the set of competitions proposed included: Student Video Competition; Physical Travelling Salesman Problem (PTSP); Multiobjective – Physical Travelling Salesman Problem (MO-PTSP); Geometry Friends; Platformer AI Competition (formerly Mario AI); StarCraft RTS AI Competition. From previous competitions, only some of them directly related to our video game proposal (Lemmings) will be analysed.

2.1.1 Ms Pac-Man

Ms Pac-Man is a predator-prey arcade game, where the species, pac-man and ghost, compete, evolve and disperse simply for the purpose of seeking resources to sustain their struggle for their very existence. The game consists of a maze with paths and corridors that the Pac-Man moves through collecting food pills that fill some of these paths. The aim of the game is to control the Pac-Man in order to clear all pills in the current maze and then advance to the next one. To do that, methodologies based on AI/CI techniques that have been used for that game are mainly based on genetic programing and coevolution [50].

During the game, the Pac-Man is chased by four ghosts any of whom will kill the Pac-Man if they are able to catch him. The ghosts behave in a non-deterministic way, which makes it hard to predict their next move although their general behaviour varies from random to very aggressive. The goal of a **Ms Pac-Man controller** is to maximise the score of the game. In the competition, it is the average score against multiple ghost teams that counts and the winning controller is the one which obtains the highest total average score. Therefore, the goal of a *ghost-team controller* is to minimise the average score obtained against it by different Ms Pac-Man controllers. The winning ghost team will be the team with the lowest average score against it.

2.1.2 Platformer AI

The Platformer AI Competition is the successor for the Mario AI Championship (`www.marioai.org`), and it is focused on two main AI topics: procedural content

generation (i.e. level generations) and imitating human behaviour. In the past IEEE-CIG 2012 competition, the Turing Test Track from the Mario AI Championship was focused on developing *human-like controllers*.

The test bed game used for the competition is a modified version of *Markus Persson Infinite Mario Bros* (mojang.com/notch/mario/) which is a public domain clone of Nintendo classical two-dimensional platform game Super Mario Bros. The game-play in *Super Infinite Mario Bros* takes place on two-dimensional levels in which the player (Mario) has to move from left to right avoiding obstacles and interacting with game objects. Mario can move left, right and jump, additionally two keys can be used to allow Mario to run, jump, or fire (depending on the state he could be in the game).

One of the main goals of this competition is to be able to compare different controllers development methodologies against each other. These controllers can be based on different AI/CI techniques such as artificial evolution, evolutionary neural networks, genetic programming, fuzzy logic, temporal difference learning, human ingenuity, hybrids of the above, etc. [61].

2.1.3 Physical Traveller Salesman Problem

The Physical Travelling Salesman Problem (PTSP) is a real-time game played on a two-dimensional map. The map has walls and obstacles, and several waypoints. In the MO-PTSP competition each map has 10 waypoints. The map itself is represented as a bitmap, where each pixel is either a wall or an empty space. The game proceeds in discrete time steps, usually one step every 40 ms (i.e. 25 per second). The player controls a spaceship, similar to the one in the classic video game **Asteroids**. The ship can rotate using a constant angular speed, and can apply thrust in the direction that it is currently pointing. The goal is to minimize three different objectives: time taken to complete the maze, fuel consumed in the process, and damage taken by the ship.

In this challenge like in Platformer AI, the aim is to be able to compare different controllers development methodologies against each other, such as Monte Carlo tree search, evolutionary neural networks, niched pareto genetic algorithms, non-dominated sorting genetic algorithms, strength pareto genetic algorithms, etc. [20].

2.1.4 StarCraft

Blizzard's StarCraft is one the most popular, and fun, examples of the real-time strategy (RTS) genre. In this game, a set of races (Protoss, Zerg and Terrans) can be used to build units that have access to different technology skills, every unit works differently and requires different tactics for a player to succeed.

The enigmatic Protoss have access to powerful units and machinery and advanced technologies such as energy shields and localized warp capabilities, powered by their psionic traits. However, their forces have lengthy and expensive manufacturing processes encouraging players to follow a strategy of the quality of their units

over the quantity. The insectoid Zerg possess entirely organic units and structures, which can be produced quickly and at a far cheaper cost to resources, but accordingly they are weaker, relying on sheer numbers and speed to overwhelm enemies. The Terrans provide a middle ground between the other two races, providing units that are versatile and flexible. They have access to a range of more ballistic military technologies and machinery, such as tanks and nuclear weapons.

There are two different game modes; one against one, and teams. The game's goal is to compete for resources and destroy the enemy. For this reason, once the game has started, the players must recollect raw material and build as quick as possible, factories, buildings, etc. During the game, both players (or teams) are constantly evolving to overcome the opponent, winning land and destroying enemies settlements. To do this it is needed that the player continuously evolves and adapts the strategy in function to the enemy movements.

The CIG StarCraft[1] competitions have shown some relevant advances in the development and evolution of new StarCraft bots. Although human top Starcraft players remain unbeaten, the machines are striving to close the gap between human and artificial intelligence.

Although each race is unique in its composition, no race has an innate advantage over the other. Each species is balanced out, so they have different strengths, powers, and abilities. But their overall strength is the same and therefore an ideal candidate to test different AI approaches like: neural networks, evolutionary algorithms, fuzzy systems, swarm intelligence and artificial immune systems [14].

2.2 The Lemmings Video Game

Lemmings are creatures that need to be saved. In each level, lemmings start in a specific point of the stage and must be guided to the exit point by the player. They live in a two-dimensional space and are affected by gravity. They start walking in a specific direction until they find an obstacle. In this case the lemming will change the direction and walk back. In the case where the Lemming encounters a hole, it will fall down. The only two ways, considered in this paper, by which a lemming can die is by falling beyond a certain distance, or by falling from the bottom of the stage.

In order to make lemmings to reach the exit point, players have a set of "skills" that must be given (not necessarily all of them) to the lemmings. Using these skills, lemmings can modify the environment creating tunnels, or bridges, and thus creating a new way to reach the exit. There are eight different skills which are the following:

Climber. A lemming given the climber skill can scale vertical walls.

Floater. This skill allow the lemming to open an umbrella if it falls beyond a high distance, avoiding its death.

[1] `ls11-www.cs.uni-dortmund.de/rts-competition/starcraft-cig2013`

Exploder. The lemming will explode after a short delay.

Blocker. Using this skill, a lemming will halt and the rest of Lemmings will turn around.

Builder. The Lemming with this skill will build a bridge of a specific length.

Basher. To create horizontal tunnels if the environment allows it.

Miner. This skill is similar to the previous one, but in this case the tunnel is dug in diagonal direction.

Digger. The lemming will dig vertically downwards until it founds air or a solid material.

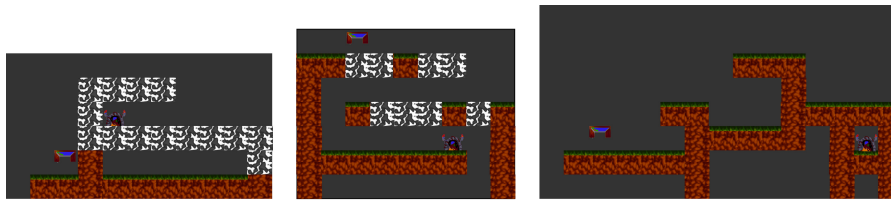
In the lemmings' world, there is a huge number of materials, but all of them can be grouped in two different classes: the ones that can be modified (i.e. they can be dug) and the ones that cannot be altered. In the former type, skills like *Basher*, *Miner* and *Digger* are allowed. In the case that a lemming is digging and finds a material that cannot be dug, the lemming will stop digging and start walking. Furthermore, each game level has its own skill configuration, where each skill can be used (i.e. assigned) a maximum number of times. It is not necessary to use all of the skills in the levels. Based on both kind of materials, editable and non editable, three different complexity levels have been designed:

Easy. These levels use both kinds of materials, and the human-like solution is a short path (few lemmings actions) with few skills that are required to reach the exit. When a non editable material is used, the lemmings colonies are "guided" to the exit because those skills related to "digging" abilities cannot be used (therefore the search space is reduced). Figure 1 a) shows an example of an easy level.

Hard. This type of level only uses editable materials, and the solution to reach the exit needs to be taken from a large number of skills and actions (large solution paths). Figures 1 c) and 2 a) show the representation of two different hard levels.

Medium. These levels use a combination of previous ones as Figure 1 b) shows. In these kinds of levels, both materials can be used and the solutions can be a mixture of actions. In the level, it is possible to find parts with a high level of freedom for the lemmings (they can use all of the available skills), and some other parts, where the number of skills, that can be used, is reduced.

The Lemmings game can be considered an interesting research video game problem specially for optimization algorithms. Three main objectives are necessary to optimize in this game: to save the maximum number of lemmings in each level, to minimize the use of skills needed to reach the exit of the level, and finally to find the best path that allows to save as many lemmings as possible using less skills.



a) b) c)

Figure 1. a) Easy level, b) medium level, c) hard level with editable and non editable material.

2.3 Summary on Video Games Platforms for AI/CI

Finally, there is a summary of the main features and characteristics from previous sections. This summary contains the following:

Competition. The name of the video game competition.

Platform. Examples of framework, or challenges, that have been used in the corresponding competition.

Algorithm. Some of the AI and CI algorithms that have been used in the competitions (it is not an exhaustive list).

Goals. The main goals that must be reached for each competition.

Although a wide number of algorithms and techniques from CI (i.e. neural networks, evolutionary strategies or fuzzy logic) and other AI methods (from heuristic search to statistical methods) have been successfully applied, the Lemmings video game provides two new interesting features. On the one hand, the video game provides different kinds of terrains, that the algorithm must take into account to avoid a premature death of the lemming, or to decide an adequate selection from the available skills. This characteristics provides an interesting "context" that should be handled by the algorithm (for instance, by using a constraint-based modelling of the environment or a meta-heuristics to select the best skill). On the other hand, the game itself needs from the management and control of a *colony* of lemmings. It is necessary to coordinate those lemmings to look for the best solution. However, the optimum solution is based on a mixture of different goals, so multi-objective algorithms can be easily applied in this domain.

- Competition: Ms Pac-Man
  - Platform: Software Kit [1], CSharp Kit, Alternative, Kit Unified Toolkit
  - Algorithms: genetic programming, coevolution
  - Goals
    - Ms Pac-man: to maximize the score of the game

    - Ghost Team: to minimize the average score obtained against it by the different Ms Pac-Man controllers

- Competition: Platformer AI
  - Platform: Level Generation Track [2], Turing Test Track [3]
  - Algorithms: evolutionary neural networks, coevolution, genetic programming, fuzzy logic, temporal difference learning, human ingenuity, hybrids of the above, artificial evolution
  - Goals
    - Procedural content generation Pac-Man controllers
    - To imitate human behaviour

- Competition: PTSP
  - Platform: the PTSP Framework [4]
  - Algorithms: evolutionary neural networks, niched Pareto genetic algorithm (NPGA), nondominated sorting genetic algorithm (NSGA), strength Pareto genetic algorithm (SPGA), Monte Carlo tree search
  - Goals
    - To minimize the fuel consumed in the process
    - To minimize the damage taken by the ship
    - To minimize the time taken to complete the maze

- Competition: StarCraft
  - Platform: AIUR for Protoss [7], BTHAI for Terran [8], Nova for Terran [10], Skynet for Protoss [5], BWAPI for all razes [9], Adjutant for Terrain [4]
  - Algorithms: evolutionary algorithms (EA), fuzzy systems (FS), swarm intelligence (SI), artificial immune systems (AIS), neural networks (NN)
  - Goals
    - Expansion around the map
    - Create legions
    - Adapt the strategy in function to the enemy movements
    - Destroy the enemy or enemies

- Competition: Lemmings
  - Platform: no available platform [38]
  - Algorithms: ant colony optimization (ACO), genetic algorithms (GA)
  - Goals. To minimize three different objectives:
    - The use of skill necessary to pass the level
    - Find the best path that allows to save more lemmings and use less abilities
    - Save the maximum number of lemmings in each level

3 CI MODELLING FOR VIDEO GAMES PROBLEM-SOLVING

3.1 Constraint-Based Graph Modelling

The proposed video game, *The Lemmings Game*, can be seen as a *constraint satisfaction problem* (*CSP*) where the variables denoted as X are the different positions of the levels and the possible values D are the skills that lemmings can execute in each position. The set of constraints C is composed by the number of lemmings that must be saved, the maximum number of skills that can be applied in each level, or the different destination from a given position taking into account the applied skill (i.e. given a position the set of possible destination nodes is different whether the skill is **Builder** or **Digger**).

In order to execute an ACO algorithm to solve a CSP, authors model the CSP as a graph $G = (V, E)$:

$$\begin{aligned} V &= \{\langle X_i, v\rangle \mid X_i \in X \text{ and } v \in D(X_i)\} \\ E &= \left\{(\langle X_i, v\rangle, \langle X_j, w\rangle) \in V^2 \mid X_i \neq X_j\right\}, \end{aligned} \tag{1}$$

where the nodes V represent the variable/value pairs ($\langle variable, value\rangle$) and E represents the edges connecting those nodes whose variables X are different. The problem with this representation is the size of the resulting graph. On the one hand, if the problem has N variables and each of them can take M different values, the resulting graph will contain $N * M$ nodes. On the other hand, the graph is almost fully-connected (so each node will be connected to the rest of nodes), for this reason the number of edges is $(N * M) * (M * (N - 1)) = N^2 * M^2 - N * M^2 \cong N^2 * M^2$. This characteristic makes that problems composed by many variables or by variables that could be assigned with many different values, become really difficult to model due to the size of the resulting graph.

In this work, the model used to represent CSP as a graph is the one described in [37]. This model is based on a simplification of the graph that groups different variables with similar meaning, and creates as many nodes as identified groups. The improvement in the size of the resulting graph is produced because with this approach the number of nodes is drastically simplified (instead of having a node for all pairs $\langle X_i, v\rangle$, the resulting graph has as much nodes as groups of variables). If the Lemmings level is mapped into a graph using the classical approach for each position, the resulting graph would have eight nodes (each of them represents the action that can be applied in the corresponding position). With the approach used in this work, each node only represents a position and the ants are in charge of selecting a specific skill to be applied in this position.

This simplification entails two disadvantages:

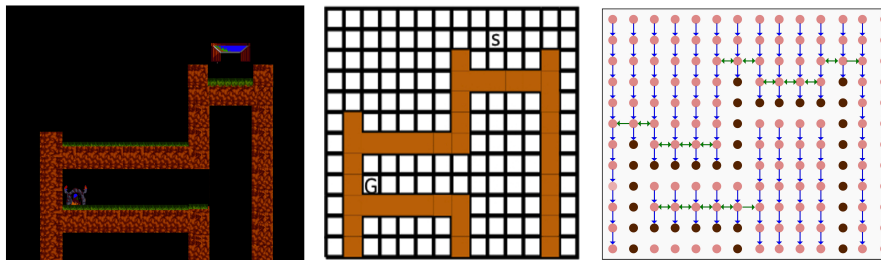
1. The behaviour of the ants is more complex. In the classic approach, ants travel through the graph and the action of visiting a node represents an assignation in the problem. For example, if an ant visits the node $\langle X_i, v_i\rangle$, the ant is assigning

the value v_i to the variable X_i. In the simplified model, used in this work, each time any ant visits a node, the ant must select a variable (position) contained in the node and assign a value (skill) for this variable taking into account the different constraints.

2. The different combination of pairs $\langle variable, value \rangle$ instead of being represented in the number of nodes contained in the graphs, are represented in the number of pheromones that ants can deposit in the graph. This problem is really important in problems composed by variables with a large set of values for the variables.

The adaptation of Lemmings level into the simplified approach is performed in two different phases. First of all, the level is represented in a two-dimensional representation that contains information about the starting point, the exit point and the terrain information of the level. Figure 2 shows an original Lemmings level (Figure 2 a)) and the simplification of this level into a two-dimensional representation (Figure 2 b)). Once the two-dimensional representation is extracted, this representation is mapped into a constraint-based graph.

The constraint-based graph contains as many nodes as squares are contained in the two dimensional representation and the edges represent the allowed movement that ants can perform. It is important to note that the application of different skills in the graph will produce the creation of new edges in the graph, thus ants deal with a dynamic graph. An example of the constraint-based graph is shown in Figure 2 c).



a) b) c)

Figure 2. A hard Lemmings level. a) shows one of the Lemmings level designed for the experiments carried out in this paper, b) shows a two-dimensional representation of this level where only the starting and exit point, and the walls are represented. Finally, c) screenshot shows the constraint-based graph model for this level.

Using this representation, the two described problems arise, nevertheless these problems are not very important in the application domain of this work for following reasons:

1. Although the behaviour of the ants is more complex, the resulting behaviour is still a swarm-behaviour that allows the simulation with a high number of ants executing at the same time in a machine.

2. The number of pheromones in the graph is not important due to the number of possible skills that can be applied in each position is, at maximum, eight.

3.2 Genetic Algorithm Approach

The GA applied in this work initializes individuals with a random genotype length. The maximum length of the genotype depends on the maximum time of the level or the maximum genotype length allowed. This genotype is a list of genes where each gene ($\langle T, S \rangle$) contains the skill (S) that is going to be executed in the step T. Both values (step and skill) are selected randomly depending on the maximum time given to solve the level, and a total number of remaining skills. The genotype represents different decisions that the player could make. This genotype is then evaluated against the level. The lemming starts its execution applying the skills specified in the given steps.

The goal is to maximize the fitness function represented by Equation (5) that it is composed by: Equations (2), (3) and (4). Equation (2) represents the number of lemmings saved in the level, and it takes into account the total number of lemmings available in the level, the number of blockers and the number of exploded lemmings. Equation (3) takes into account the total time that is given and time spent by the lemming to solve the level while Equation (4) is used to favour those paths that use less actions, or less skills. For this reason this last equation uses the total number of available actions and the number of actions that the lemming has used.

$$S(Ind) = TotalLemmings - BlockersUsed(Ind) - ExplodedLemmings(Ind) \quad (2)$$

$$T(Ind) = MaxTime - Time(Ind) * S(Ind) \quad (3)$$

$$A(Ind) = TotalActionsAvailable - ActionUsed(Ind) \quad (4)$$

$$F(Ind) = \frac{T(Ind) + A(Ind) + S(Ind)}{MaxTime + TotalActionsAvailable + TotalLemmings} \quad (5)$$

Although ACO will use the same function (Equation (5)) to evaluate the goodness of the paths, only GA can produce a negative fitness value. This negative value is obtained in the case where the individual produces an invalid path (i.e. in the evaluation the lemming is not able to reach the exit point or the lemming dies trying it). In this case, the fitness value is $F(Ind_i) - 1$, in such a way, paths really close the optimal solution will have a fitness value close to 0 while the corresponding values for the worst path will be close to -1.

3.3 Ant Colony Optimization Approach

This work uses a classical ACO to search for the best paths of the levels. In this case, the nest of the colony is located in the node that represents the level starting point and the food is located in the node that represents the level exit point.

From the nest, ants start building their own local solution while they travel through the graph. In order to do that, each ant executes the behaviour described in Algorithm 1.

Algorithm 1: Ants behaviour

```
1  S ← possibleSkills(currentPosition)
2  P ← pheromoneInformation(currentPosition)
3  skill ← selectSkill(S, P)
4  if (skill ≠ null)) then
5      if canBeExecuted(skill, currentPositions) then
6          updateRemainingSkill()
7          putPheromone()
8          updateCurrentAction(skill)
9      end
10 end
11 execute(currentAction)
```

Line 1 in this behaviour corresponds with the problem heuristic. In this work, two different heuristics have been used called *Random Heuristic* and *Common-Sense Heuristic*. Using the random heuristic, each ant can execute any skill in any position.

The second heuristic used in this work is called *Common-Sense Heuristic*. In this case, ants can perceive the environment (i.e. ants know the type of terrain of the surrounding nodes) and filter the skills that they can apply depending on this environment. For example, given an ant if the type of the node where the ant is placed and their surrounding are *Air*, the ant understands that the lemming is falling and a possible skill to apply is **Floater**, but not **Builder**.

Once the ants have the values for the different skills, corresponding to the heuristic function and the pheromones, the decision of selecting one of them is computed using Equation (6).

$$p_{ij} = \begin{cases} \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{u \in \mathcal{N}_i^k} \tau_{iu}^{\alpha}\eta_{iu}^{\beta}} & \text{if } j \in \mathcal{N}_i^k \\ 0 & \text{if } j \notin \mathcal{N}_i^k \end{cases} \tag{6}$$

where $\mathcal{N}_i^k$ is the set of feasible nodes connected to node i, with respect to the ant k. τ_{ij} represents the pheromone value of travelling from node i to node j and η_{ij} represents the heuristic value of moving from node i to node j. Finally α and β are two parameters that control the influence of the pheromones and the heuristic function exerts in the ants behaviour. On the one hand, if $\beta \gg \alpha$, ants will be guided basically by the heuristic function. On the other hand, if $\beta \ll \alpha$, the ants will follow the paths found first and thus the algorithm will show a rapid convergence to suboptimal paths.

Finally, when any ant finishes its path (because the ant has reached the food, the ant is trapped or because by following this path the corresponding lemming will

die), ant undoes the followed path updating the corresponding pheromones with a value that represents the goodness of this path. In this case, this value is obtained by the same function as the one used in the GA (Equation (5)). When the ants reach the nest, they forget the followed path and start a new search.

4 EXPERIMENTAL RESULTS

In this section, some experimental results are given. The aim of the experiments is to analyse the influence of the environment in the behaviour of GA and ACO algorithms.

4.1 Experimental Setup

Parameter	Value
Population size	100
Max. genotype length	20
Generations	500
# Offsprings	1
Crossover rate	90 %
Crossover type	One point
Mutation rate	1 %
Elitism	No

Table 1. Configuration of the experiments with GA

Fourteen different levels have been designed to measure the efficiency of corresponding algorithms. The complexity of the levels is based on the size of a level, different blocks contained into each level, the distance from the entry point to the exit point, the number of skills needed to solve the level, the type of terrains contained in the levels, etc. In this work, three different complexity levels are considered: easy, medium and hard.

Parameter	Value
# Ants	100
# Steps	500
Evaporation rate	1 %
α	1
β	1

Table 2. This table describes the configuration for the ACO experiments. The parameters showed in this table are the number of ants that compose the colony, the number of the simulation steps, the values for α and β that determine the influence of the heuristic function and the pheromones value exert in the ants selection process, and the evaporation rate that is used in the *stigmergy* process

All the experiments have been repeated 50 times, and they have been analysed using GA, ACO with random heuristic and ACO with common-sense heuristics. From these experiments, the number of different found paths is used to compare the performance of these algorithms.

The configuration of the different parameters used in the GA are shown in Table 1 while Table 2 shows the configuration for the ACO experiments.

4.2 Experimental Analysis

Until now, experimental results reveal that both, GA and ACO, can be applied to search the paths that solve the levels. Nevertheless an important questions can be formulated: *How many different successful paths can these algorithms find?* This is a question about how many different solutions this algorithm can find. This information is showed in Table 3. The values correspond to the number of different paths found in 50 executions of the algorithms.

Level	Complexity	Genetic Algorithm	ACO (Random Ant)	ACO (Common-Sense)
1	Easy	3 219	3 868	2 516
2	Easy	12 629	4 463	4 042
3	Easy	2 364	446	1 330
4	Easy	649	2 128	2 560
5	Medium	370	1 130	2 487
6	Medium	162	575	2 520
7	Medium	54	35	157
8	Medium	1	12	228
9	Medium	7	348	326
10	Hard	2	15	32
11	Hard	0	3	7
12	Hard	2	18	62
13	Hard	164	26	35
14	Hard	1	4	23

Table 3. Number of different solutions found by the algorithms described

Dealing with easy levels, all algorithms founds a large set of different paths. In general, genetic algorithms and ACO with the random heuristic finds more paths than ACO with the common-sense heuristic. This is produced because GA and the random ACO are able to explore different actions that allows them to find different paths.

Taking into account medium and hard levels, ACO (with both heuristics) finds more paths than GA. This is an expected result because ACO gradually builds the solutions and GA needs to create the whole path at the beginning. When the size of the level increases, the difficulty of building a valid path from the beginning increases.

Finally, in general when the problem is medium or hard, ACO with the common-sense heuristic finds more paths than ACO with the random heuristic. This is produced because the common-sense heuristic uses the environment of the ant to filter the set of possible actions to execute. On the contrary, the random heuristic makes ants to explore solutions that produce the death of the lemming.

5 CONCLUSIONS

This paper analyses the possibility of applying genetic algorithm and ant colony optimization to generate automatic game level solver tools. The application domain of this work is the well-known Lemmings game, where lemmings have to apply different skills in order to reach the exit. Fourteen different levels have been designed with different complexity depending on the size of the level, the number of available skills, or the distance between the start and the destination point, amongst others.

Experimental results reveal that both algorithms can successfully be applied to solve the levels. Nevertheless, as it can be seen in Table 3, the genetic algorithms provide less different paths than ant colony optimization, when the levels are medium or harder. This may be produced because GA generates individuals without taking into account the level landscape (i.e. it is a blind generation of individuals). On the other hand, ACO uses the terrain information to apply different skills at specific steps and thus, it provides better results.

Regarding the question whether it is important, or not, the environment for the ant colony optimization, the performance of random ants is compared with the performance of common-sense ants that takes into account the environment in order to select what skill is going to be applied. Taking into account the number of different paths found by both algorithms, random ants find more paths when the level is medium, but when the problem is hard, the common-sense ant algorithm reports more different solutions. This fact demonstrates that level information is very important to make automatic level solvers.

We plan the study of these algorithms applied to a real world application such as *Project Scheduling Problem* or *Traffic Optimization*, and to test whether we obtain good results.

Acknowledgments

This work is supported by the Spanish Ministry of Science and Education under Project Code TIN2014-56494-C4-4-P, Comunidad Autonoma de Madrid under project CIBERDINE S2013/ICE-3095, and Savier an Airbus Defense & Space project (FUAM-076914 and FUAM-076915).

REFERENCES

[1] Ms Pac-man Software Kit. http://dces.essex.ac.uk/staff/sml/pacman/PacManContest.html.

[2] Platformer AI Level Generation Track. http://www.marioai.org/LevelGeneration.

[3] Platformer AI. Turing Test Track. http://www.marioai.org/turing-test-track.

[4] PTSP Framework. http://www.marioai.org/turing-test-track.

[5] Skynet for Protoss. http://code.google.com/p/skynetbot/.

[6] Starcraft Adjutant for Terran. http://code.google.com/p/adjutantbot/.

[7] Starcraft. AIUR for Protoss. http://code.google.com/p/aiurproject/.

[8] Starcraft. BTHAI for Terran. http://code.google.com/p/bthai/.

[9] Starcraft. BWAPI for All Razes. https://code.google.com/p/bwapi/.

[10] Starcraft. Nova for Terran. http://nova.wolfwork.com.

[11] Abraham, A.—Ramos, V.: Web Usage Mining Using Artificial Ant Colony Clustering and Linear Genetic Programming. The 2003 Congress on Evolutionary Computation (CEC 03), 2003, Vol. 2, pp. 1384–1391.

[12] Acan, A.: GAACO; A GA + ACO Hybrid for Faster and Better Search Capability. In Ant Algorithms. Springer Berlin, Heidelberg, 2002.

[13] Berghman, L.—Goossens, D.—Leus, R.: Solving Mastermind Using Genetic Algorithms. Computers & Operations Research, Vol. 36, 2009, pp. 1880–1885.

[14] Blickle, T.—Thiele, L.: A Comparison of Selection Schemes Used in Evolutionary Algorithms. Evolutionary Computation, Vol. 4, 1996, pp. 361–394.

[15] Blum, C.—Merkle, D.: Swarm Intelligence. Introduction and Applications. 1st edition, Springer, 2008.

[16] Bonabeau, E.—Dorigo, M.—Theraulaz, G.: Swarm Intelligence. From Natural to Artificial Systems. Oxford, 1999.

[17] Bäck, T.—Schwefel, H.: An Overview of Evolutionary Algorithms for Parameter Optimization. Evolutionary Computation, Vol. 1, 1993, No. 1. pp. 1–23.

[18] Chen, X.—Wang, H.—Wang, W.—Shi, Y.—Gao, Y.: Apply Ant Colony Optimization to Tetris. Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2009, No. 1, pp. 1741–1742.

[19] Chunlin, J.: A Revised Particle Swarm Optimization Approach for Multi-Objective and Multi-Constraint Optimization. GECCO, 2004.

[20] Coldridge, J.—Amos, M.: Genetic Algorithms and the Art of Zen. Technical report, Manchester Metropolitan University, 2010.

[21] Cole, N.—Louis, S. J.—Miles, C.: Using a Genetic Algorithm to Tune First-Person Shooter Bots. Proceedings of the Congress on Evolutionary Computation, 2004, No. 1, pp. 139–145.

[22] Colorni, A.—Dorigo, M.—Maniezzo, V.: Distributed Optimization by Ant Colonies. European Conference on Artificial Life, 1991, pp. 134–142.

[23] Cormode, G.: The Hardness of the Lemmings Game, or Oh No, More NP-Completeness Proofs. Proceedings of Third International Conference on Fun with Algorithms, 2004, pp. 65–76.

[24] Daz-Pernil, D.—Gutirrez-Naranjo, M. A.—Perez-Jimenez, M. J.—Riscos-Nuñez, A.: A Linear Solution for Subset Sum Problem With Tissue P Systems with Cell Division. Lecture Notes in Computer Science, Vol. 4527, 2007, pp. 170–179.

[25] Das, S.—Biswas, A.—Dasgupta, S.—Abraham, A.: Bacterial Foraging Optimization Algorithm. Theoretical Foundations, Analysis, and Applications. Foundations of Computational Intelligence, Vol. 203, 2009, pp. 2355.

[26] Kumar Das, T.: Bio-Inspired Algorithms for the Design of Multiple Optimal Power System Stabilizers. SPPSO and BFA. IEEE Transactions on Industry Applications, Vol. 44, 2008, No. 5, pp. 1445–1457.

[27] Demaine, E. D.—Hohenberger, S.—Liben-Nowell, D: Tetris Is Hard, Even to Approximate. Proceedings of the 9th International Computing and Combinatorics Conference, 2003.

[28] Dorigo, M.—Gambardella, L. M.: Ant Colonies for the Travelling Salesman Problem. 1997.

[29] Dorigo, M.: Ant Colony Optimization. A New Meta-Heuristic. Proceedings of the Congress on Evolutionary Computation, IEEE Press, 1999, pp. 1470–1477.

[30] Eiben, A. E.—Smith, J. E.: Introduction to Evolutionary Computing. Springer-Verlag, 2009.

[31] Engelbrecht, A. P.: Computational Intelligence. An Introduction. Wiley Publishing, 2nd edition, 2007.

[32] Dressler, F.—Akan, O. B.: Bio-Inspired Networking. From Theory to Practice. IEEE Communications Magazine, November 2010, pp. 177–183.

[33] Farooq, M.: Bee-Inspired Protocol Engineering. From Nature to Networks. Springer Publishing Company, 2008.

[34] Fogel, D. B.: Evolutionary Computation. Toward a New Philosophy of Machine Intelligence. IEEE Press, 1995.

[35] Forrest, S.: Genetic Algorithms. Principles of Natural Selection Applied to Computation. Science, Vol. 261, 1993, No. 5123, pp. 872–878.

[36] Fu, T.—Liu, Y.—Zeng, J.—Chen, J.: An Improved Genetic and Ant Colony Optimization Algorithm and Its Applications. In: Huang, D.-S., Li, K., Irwin, G. (Eds.): Intelligent Control and Automation, Springer, Berlin, Heidelberg, Lecture Notes in Control and Information Sciences, Vol. 344, 2006, pp. 229–239.

[37] Gonzalez-Pardo, A.—Camacho, D.: A New CSP Graph-Based Representation for Ant Colony Optimization. 2013 IEEE Conference on Evolutionary Computation, 2013, Vol. 1, pp. 689–696.

[38] Gonzalez-Pardo, A.—Camacho, D.: Environmental Influence in Bio-Inspired Game Level Solver Algorithms. 7th International Symposium on Intelligent Distributed Computing (IDC 2013), Studies in Computational Intelligence, Springer, Berlin, Heidelberg, Vol. 511, 2014, pp. 157–162.

[39] Guangdong, H.—Ping, L.—Qun, W.: A Hybrid Metaheuristic ACO-GA with an Application in Sports Competition Scheduling. Eighth ACIS International Confer-

ence on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007.

[40] Houston, R.—White, J.—Amos, M.: Zen Puzzle Garden is NP-Complete. Information Processing Letters, Vol. 112, 2012, pp. 106–108.

[41] De Jong, K. A.: Evolutionary Computation – A Unified Approach. MIT Press, 2006.

[42] Karaboga, D.: An Idea Based on Honey Bee Swarm for Numerical Optimization. Technical Report TR06, Erciyes University Press, Erciyes, Vol. 129, 2005, No. 2.

[43] Karaboga, D.—Basturk, B.: A Powerful and Efficient Algorithm for Numerical Function Optimization. Artificial Bee Colony (ABC) Algorithm. J. Global Optimization, Vol. 39, 2007, pp. 459–471.

[44] Kendall, G.—Spoerer, K.: Scripting the Game of Lemmings with a Genetic Algorithm. Proceedings of the Congress on Evolutionary Computation, 2004, Vol. 1, pp. 117–124.

[45] Kennedy, J.—Eberhart, R.: Particle Swarm Optimization. Proceedings of the Congress on Evolutionary Computation, Vol. 4, 1995, pp. 1942–1948.

[46] Luca Lanzi, P. (Ed.): Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG 2009), Milano, Italy, September 7–10, 2009, IEEE, 2009.

[47] Lee, Z. J.—Su, S. F.—Chuang, C. C.—Liu, K. H.: Genetic Algorithm with Ant Colony Optimization (GA-ACO) for Multiple Sequence Alignment. Applied Soft Computing, Vol. 8, 2008, No. 1. pp. 55–78.

[48] Lim, C. U.—Baumgarten, R.—Colton, S.: Evolving Behaviour Trees for the Commercial Game Defcon. In EvoGames, 2010.

[49] Lucas, S.: Computational Intelligence and AI in Games. A New IEEE Transactions. T-CIAIG, 2011.

[50] Martin, E.—Martinez, M.—Recio, G.—Saez, Y.: Pac-mAnt. Optimization Based on ant Colonies Applied to Developing an Agent for Ms. Pac-Man. Proceedings of the Symposium on Computational Intelligence and Games (CIG), 2010, Vol. 1, pp. 458–464.

[51] Miikkulainen, R.—Bryant, B. D.—Cornelius, R.—Karpov, I. V.—Stanley, K. O.—Yong, C. H.: Computational Intelligence in Games. Computational Intelligence, Principles and Practice, 2006.

[52] Miller, B. L.—Goldberg, D. E.: Genetic Algorithms, Selection Schemes and the Varying Effects of Noise. Evolutionary Computation, Vol. 4, 1996, No. 2. pp. 113–131.

[53] Nemati, S.—Basiri, M. E.—Ghasem-Aghaee, N.—Aghdam, M. H.: A Novel ACOGA Hybrid Algorithm for Feature Selection in Protein Function Prediction. Expert Systems with Applications, Vol. 36, 2009, No. 10, pp. 12086–12094.

[54] Pilat, M.—White, T.: Using Genetic Algorithms to Optimize ACS-TSP. In: Dorigo, M., Di Caro, G., Sampels, M. (Eds.): Ant Algorithms, Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2463, 2002, pp. 101–172.

[55] Potvin, J. Y.: A Review of Bio-Inspired Algorithms for Vehicle Routing. CIRRELT, Vol. 30, 2008.

[56] Revello, T. E.—McCartney, R.: Generating War Game Strategies Using a Genetic Algorithm. Proceedings of the Congress on Evolutionary Computation, 2002, Vol. 1, pp. 1086–1091.

[57] Reynolds, C. W.: Flocks, Herds and Schools. A Distributed Behavioral Model. SIGGRAPH Comput. Graph., Vol. 21, 1987, pp. 25–34.

[58] Ridge, E.—Curry, E.: A Roadmap of Nature-Inspired Systems Research and Development. Multiagent Grid Syst., Vol. 3, 2007, No. 3–8.

[59] Ruan, X.—Gong, D.: A Hybrid Approach of GA and ACO for TSP. Proceedings of the 5 World Congress on Intelligent Control and Automation, 2004.

[60] Shaker, N.—Togelius, J.—Yannakakis, G. N.—Weber, B. G.—Shimizu, T.—Hashiyama, T.—Sorenson, N.—Pasquier, P.—Mawhorter, P. A.—Takahashi, G.—Smith, G.—Baumgarten, R.: The 2010 Mario AI Championship, Level Generation Track. IEEE Trans. Comput. Intellig. and AI in Games, Vol. 3, 2011, No. 4, pp. 332–347.

[61] Togelius, J.: Mario AI Competition.

[62] Feng, X.—Lau, F. C. M.—Gao, D.: A New Bio-Inspired Approach to the Travelling Salesman Problem. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 5, 2009, pp. 1310–1321.

[63] Xu, Y. L.—Lim, M. H.—Ong, Y. S.—Tang, J.: A GA-ACO-Local Search Hybrid Algorithm for Solving Quadratic Assignment Problem. Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, 2006.

[64] Li, Y.—Mascagni, M.: A Bio-Inspired Job Scheduling Algorithm for Monte Carlo Applications on a Computational Grid. IMACS-World Congress, 2005.

Antonio Gonzalez-Pardo is a Post-Doctoral researcher at BCAM-TECNALIA. He received his Ph.D. degree in computer science (2014) from Universidad Autónoma de Madrid, his B.Sc. degree in computer science from Universidad Carlos III de Madrid (2009) and his M.Sc. degree in computer science from Universidad Autónoma de Madrid (2011). Currently he is a post-doctoral researcher in a join position offered by Basque Center for Applied Mathematics (where he belongs to the Machine Learning group) and also by TECNALIA (where he participates with the Optima group). He is involved with AIDA interest research group at EPS-UAM. His main research interests include computational intelligence (genetic algorithms, PSO, swarm intelligence...), multi-agent systems and machine learning techniques. The application domains for his research are constraint satisfaction problems (CSP), complex graph-based problems, optimization problems and video games.

Fernando Palero received his B.Sc. degree in computer science from Universidad Politécnica de Valencia (2009) and his M.Sc. degree in computer science from Universidad Autónoma de Madrid (2014). He is involved with AIDA interest research group at EPS-UAM, his main research interests include computational intelligence (genetic algorithms and swarm intelligence), and multi-agent systems.

David Camacho is currently working as Associate Professor in the Computer Science Department at Universidad Autónoma de Madrid (Spain) and leads the Applied Intelligence & Data Analysis group (`http://aida.ii.uam.es`). He received his Ph.D. degree in computer science (2001) from Universidad Carlos III de Madrid, Spain. His research interests include data mining (clustering), evolutionary computation (GA & GP), multi-agent systems and computational intelligence (swarm computing), automated planning and machine learning. His work has appeared in leading computer science journals. He serves in the editorial boards of International Journal of Bio-Inspired Computation, International Journal of Computer Science & Applications, Applied Artificial Intelligence, and Engineering Letters Journal, among others.