Automated Map Generation for the Physical Travelling Salesman Problem

Diego Perez, *Student Member IEEE*, Julian Togelius, *Member IEEE*,
Spyridon Samothrakis, *Student Member IEEE*, Philipp Rohlfshagen, *Member IEEE*,
Simon M. Lucas, *Senior Member IEEE*

***Abstract*—This paper presents a method for generating complex problems that allow multiple non-obvious solutions for the Physical Travelling Salesman Problem (PTSP). PTSP is a single-player game adaptation of the classical Travelling Salesman Problem that makes use of a simple physics model: the player has to visit a number of waypoints as quickly as possible by navigating a ship in real time across an obstacle-filled two-dimensional map. The difficulty of this game depends on the distribution of waypoints and obstacles across the two-dimensional plane. Due to the physics of the game, the shortest route is not necessarily the fastest, as the ship's momentum makes it difficult to turn sharply at speed. This paper proposes an evolutionary approach to obtaining maps where the optimal solution is not immediately obvious. In particular, any optimal route for these maps should differ distinctively from (a) the optimal distance-based TSP route and (b) the route that corresponds to always approaching the nearest waypoint first. To achieve this, the evolutionary algorithm CMA-ES is employed, where maps, indirectly represented as vectors of real numbers, are evolved to differentiate maximally between a game-playing agent that follows two or more different routes. The results presented in this paper show that CMA-ES is able to generate maps that fulfil the desired conditions.**

I. Introduction

Procedural content generation in games is a growing research field motivated by a real need within game development, and a research goal to enable new kinds of interactive techniques [26]. Techniques developed in the field, especially evolutionary techniques, have been employed elsewhere with great success and content generation has even been the focus of some competitions such as the Mario AI Championship [22]. Sometimes, procedural content generation needs to tackle the problem of infeasible solutions: not only must the content generated be good for the problem at stake, but it also needs to be valid material for the domain in which it is applied. In cases where a problem of some kind is designed, such as a level, a quest or a puzzle, the design goal is often that the problem should be interesting, in the sense that it can be solved in different ways and that the best solution is not obvious. This paper presents a method for evolving maps that have exactly those properties.

The testbed game used is the Physical Travelling Salesman Problem (PTSP), a single-player game adaptation of the classical Travelling Salesman Problem (TSP)[1] that makes use of a simple physics model: the player has to visit a number of waypoints as quickly as possible by navigating a ship in real time across an obstacle-filled two-dimensional map. While abstract, the game retains certain similarities with some of the most important features of video games: navigation, obstacle avoidance, pathfinding, and the real-time component found in most modern games that forces the player to supply an action every few milliseconds. Additionally, like in many modern video-games, the game is long enough to ensure that the outcomes derived from the actions taken by the agent do not provide enough information to determine if they will lead to a victory or a loss. The PTSP features in a popular game competition organised by the Game Intelligence Group at the University of Essex where competitors submit software controllers to navigate a series of unknown maps [19]. The winner is the controller that manages to visit the most waypoints in the fewest number of time steps across all maps.

All the maps used in the competition have been designed by hand to ensure they are sufficiently interesting and challenging. However, while successful, this approach is time and labour intensive and lacks flexibility and extensibility. Automating the generation of interesting PTSP maps would save much human labour and make it possible to scale up the competition and its use as a benchmark. However, there are several other good reasons to develop an effective map generator for PTSP – in particular the following the reasons motivate the current paper:

- Finding maps that require or advantage particular solution strategies and controller types.
- Finding maps where multiple solutions of the same or very similar quality exist. This is analogous to showing that multiple strategies of similar effectiveness exist for a strategy game, something which is generally considered advantageous.
- Clarifying the relative strengths and weaknesses of controllers by finding and analysing maps in which they beat each other; e.g., both maps where controller A beats

Diego Perez, Spyridon Samothrakis, Philipp Rohlfshagen, Simon M. Lucas, (School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, UK; email: dperez,ssamot,prohlf,sml@essex.ac.uk);
Julian Togelius (Center for Computer Games Research, IT University of Copenhagen, 2300 Copenhagen, Denmark); email: julian@togelius.com);
Copyright (c) 2012 IEEE.

[1]The TSP requires an agent to visit all of n cities exactly once using the shortest route possible, starting and ending the route at the same city.

controller B and where controller B beats controller A.

Concretely, the objective is to produce maps where the optimal order of waypoints is neither the optimal distance-based TSP route nor the nearest-city-first route. In other words, a controller that makes some computational effort on long-term planning should be able to see an improvement, especially if taking the physics of the game into account, against a controller which works in a simple short-term reactive manner. Such maps are relatively easy to construct by hand if the number of waypoints are few and the map's size is modest. However, it is clear that a manual approach does not scale well. It is also limited if many maps are required, which is the case in the PTSP competition.

Additionally, it is interesting to find several routes that obtain similar high quality results following different way-point orders, that are distinct from the optimal distance-based and nearest-city-first TSP routes. Designing maps with these features by hand is a very complicated task, and an automated solution would open the possibility to a 2-simultaneous-player version of the game where challenging maps can be used. Furthermore, automatic content generation may be used to personalise the game towards specific gamers (offline or online), or to create specific maps to test the extreme behaviour of controllers in the competition.

This paper presents the application of an evolutionary algorithm, the Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) [8], to automatically generate content by simulating agents playing the game. The experiments described in this research show how by using different strategies, from naive to more involved ones, it is possible to evolve maps that favour one type of play over others. This allows the creation of levels for the game that are not trivially solved by simplistic approaches. Given the similarity of constrained content generation for games to problems in other application areas, especially design fields, the techniques developed here are very likely to have direct applicability outside of games.

This paper is structured as follows. First, in Section II, a summary of related research in automatic content generation is presented. Section III describes the game and the framework in detail. Then, Section IV describes the technique used to generate maps and the experimental setup, followed by the analysis of the results in Section V. Finally, Section VI presents the conclusions and future work.

II. Related work

We are not aware of any previous work that attempts to create maps for the Physical Travelling Salesman Problem automatically. However, a number of previous papers address the evolution of content for games in general, and the generation of maps for 2D games in particular, as described in the following section.

A. Procedural content generation

Procedural Content Generation (PCG) refers to the generation of game content (e.g. levels, maps, items, quests, puzzles, texts) with none or limited human intervention. PCG has occasionally been featured in video games during the last three decades, but it has only become a topic of academic interest within the last few years [26]. This problem is becoming more relevant owing to its wide applicability to different areas within video-games. One of the earliest uses of PCG for games is the space trading game *Elite* (Acornsoft, 1984), which employed procedural representation of star systems to reduce memory consumption. More recently, PCG has been used extensively for level or map generation in games such as *Spelunky* (Mossmouth, 2012) and those in the *Civilization* (Firaxis Games, 2012), *Diablo* (Blizzard, 1998) and *Borderlands* (Gearbox Software, 2009) game series. PCG has also been used to create literally endless games, with an infinite number of levels and open spaces that extend for as long as the player bothers to look. Some examples are *Elite* (Acornsoft, 1984), *The Sentinel* (Geoff Crammond, 1987) and *Malevolence: The Sword of Ahkranox* (Visual Outbreak, 2012).

In academic PCG research, the focus is often on ways of searching a space of game content for artefacts that satisfy certain criteria or objectives. For instance, Hastings et al. [10] created weapons in the game Galactic Arms Race (GAR) using a collaborative evolutionary algorithm. Whitehead [28] suggests that PCG can improve game aesthetics, and an example of this use of PCG is given by Liapis et al. [15], who study the automatic generation of game spaceships through interactive neuroevolution. PCG can also be used to generate rules in a game or even complete games. This has been extensively addressed by Browne in [7]. The author describes a game description language that can be used to specify the complete rule set that defines a board game, including starting position, valid moves, winning conditions, type of board, number of players, etc. He then proposes an evolutionary algorithm that creates complete new games using the description language developed.

Adaptation to the type of player is another motivation for PCG [29]. There are many different types of players, ranging from hardcore to occasional players, and not all of them play the same types of games. This is especially true in the last few years, when new games have focused on more unexplored genres, such as "family" or fitness games. PCG could be used to adapt the game to the type of player, producing more appropriate content by, for example, adjusting the difficulty to the skill of the player, or the type of challenge to that preferred by the player. We are not aware of any published game that employs PCG for such adaptation to the player's abilities, but it is clearly a possible application for these techniques, considering that the latest movements in the industry focus on targeting a broader audience.

Another important motivation for the use of PCG, especially in the development of high-budget commercial video games, is the possibility of reducing production time and costs. An example of this is the SpeedTree software [12], a tool for automatic creation of vegetation. This tool has been used in numerous recent games, and some games that employ

this algorithm are very popular in the gaming community, such as *Grand Theft Auto IV* (Rockstar Games, 2008) and *Fallout 3* (Bethesda Game Studios, 2008).

Problems of game content generation share many characteristics with design problems in other application areas involving interactive or complex systems. For example, in circuit board design, logistics, and road network planning, intricate path networks have to be designed while taking into account constraints relating to order, speed and interference. Robot and vehicle design problems involve designing for dynamical systems with nondeterministic behaviour. These characteristics also apply to the PTSP maze generation problem, reaffirming the potential for such techniques being applicable to automatically solving other design problems.

The two key considerations when using evolution or some similar search/optimisation algorithm to generate content are content evaluation (fitness function) and representation. A survey defining these problems, approaches to them, and the application of PCG in the literature can be found in [26]. Regarding content representation, this topic is central to several related fields, and surveys have been written from the perspectives of evolutionary computation [24] and of artificial life [3]. Typically, there are many possible ways of representing some types of game content [2], ranging from direct to more indirect approaches [26, p. 4]. For example, a dungeon could be represented in many different ways including the following:

1) Grid cell: each element in the game is specifically placed in a source matrix.
2) List of positions, orientations, and sizes of areas.
3) Repository of segments or *chunks* of levels to combine.
4) Desired properties: number of corridors, rooms, etc.
5) Random number seed: the level generator creates a level taking only a number as input.

In this paper, a relatively direct representation of PTSP maps is employed, corresponding to the second option in the list above.

On the other hand, the generator usually needs a procedure to evaluate the quality of fitness of the generated levels. This function should rate or rank, in order to be able to sort them from best to worst. There are several different ways this could be done:

- Direct evaluation: some objective features are retrieved from the map and a score for the level is provided by an evaluation function.
- Simulation-based evaluation: a programmed bot plays the game and a measure of its performance is taken, which is used to score the maps. This bot can be fully hand coded or it can imitate a human player using machine learning techniques.
- Interactive evaluation: a human plays the game and feedback is obtained from him. This can be done either explicitly, with a questionnaire, or implicitly, measuring features of the gameplay, such as death locations, actions taken, distance travelled, etc.

The research presented in this paper uses simulation-based evaluation, as agents are used to play PTSP maps.

B. Constrained optimisation

While many search/optimisation problems differentiate solution quality on a continuum, others feature *constraints*, so that some candidate solutions are not just bad, but *infeasible*. For example, a Mario level where some gaps can't be bridged or a PTSP instance where some waypoints cannot be reached are clearly infeasible. In evolutionary computation, several specialised techniques for handling constraints in optimisation have been developed [18]. Several different approaches can be discerned, including the "naive" approach of simply giving infeasible solutions a score of zero and more "sophisticated" solutions such as repairing infeasible individuals or keeping separate populations of feasible and infeasible individuals. In search-based procedural content generation, the two-population approach has previously been used for generating game content such as platform game levels [23] and spaceships [14], [15].

This paper takes a relatively naive approach to constraint handling, as described in section IV-A. This is done in order to be able to use the CMA-ES as the evolutionary algorithm. This algorithm has previously been shown to be extremely effective for continuous optimisation [8], but has not to our best knowledge been applied to procedural content generation before. The use of a well-tested off-the-shelf continuous optimisation algorithm, rather than a specifically tailored constraint optimisation algorithm, carries considerable benefits in terms of ease of experimentation and replicability.

C. Game AI competitions

In recent years, a number of competitions have been arranged in association with conferences on AI and games, such as the IEEE Conference on Computational Intelligence and Games, and the Artificial Intelligence and Interactive Digital Entertainment conference. In general these competitions work in the following manner: competitors submit agents, written in a programming language such as Java, which connect to a competition evaluation engine provided by the competition organisers. The winner is generally the competitor whose software played the game best. Some of the most popular competitions have been based on well known games, such as Ms. Pac-Man [21], [17], Super Mario Bros [13], or lesser known games in a well-defined and popular genre, such as the car racing game TORCS [16]. However, in some competitions the player does not submit a high-performing controller, but rather a controller that plays the game in an as human-like manner as possible [11], or even a level generator that generates as entertaining levels as possible for particular players [22].

III. The Physical Travelling Salesman Problem

The Physical Travelling Salesman Problem (PTSP) is a modification of the well known combinatorial optimisation

problem, the Travelling Salesman Problem (or TSP). In the TSP, a set of cities are distributed and the costs of travelling from one to any other are known. The objective is to find the route that takes an agent or salesman to visit all cities once with the minimum overall cost. The PTSP game, introduced by Perez et al. [19], converts the TSP into a real-time game, where the player must govern a ship to visit, as quickly as possible, a determined number of waypoints scattered around a map full of obstacles.

A. The problem

The PTSP is a single player real-time game where the ship and the waypoints are positioned in a two dimensional continuous grid. When the game starts, a tick counter, initialised at 1000, starts decreasing at a rate of one unit per time step. The ship must visit one of the remaining waypoints of the map before this counter reaches 0. If the ship is not able to do that, the game ends. Otherwise, and if there are more waypoints in the map to collect, the counter is reset to 1000 and the ship must visit another one, until all waypoints have been visited. Hence, the score of the game is stated by the number of waypoints visited and the total number of time steps taken. The objective of the game is to visit all the waypoints of the map as quickly as possible. One solution is considered to be better than another if the number of waypoints visited by the first one is higher than the number of waypoints visited by the second solution. In case of a draw, the solution that involves less time spent is the winner.

Furthermore, the PTSP is a real-time game: an action must be supplied every 40 milliseconds or the ship will apply the default action (idle), and there is an initialization time of 1000 milliseconds at the beginning of the game. The ship is governed by applying one out of the six available actions at a time, resultant of a combination of two different inputs: acceleration (that can be on or off) and rotation (left, right or straight). These actions can be considered as forces that change position, velocity and direction of the ship. The physics of the game include inertia, by keeping the ship's velocity from one time step to the next, and friction, so the ship eventually stops if no acceleration actions are supplied.

Additionally, the levels of the PTSP contain multiple obstacles that make the navigation an important element of the problem to solve. These obstacles do not damage the ship, although they do modify its velocity by reducing the speed and producing an elastic collision when the ship hits the walls.

The PTSP poses a problem at two different levels: solving the order in which to visit the waypoints, and navigating through the map to reach them. A priori, these problems might sound independent, but the fact is that they are closely related: solving the TSP using the costs between the waypoints based exclusively on the distances may obtain optimal paths that are not the optimal routes for the PTSP. The physics of the ship (especially its inertia), and how the navigation is performed, have a big impact on the time taken for travelling from one waypoint to another. Ignoring this may cause the ship to describe suboptimal routes to solve the game.

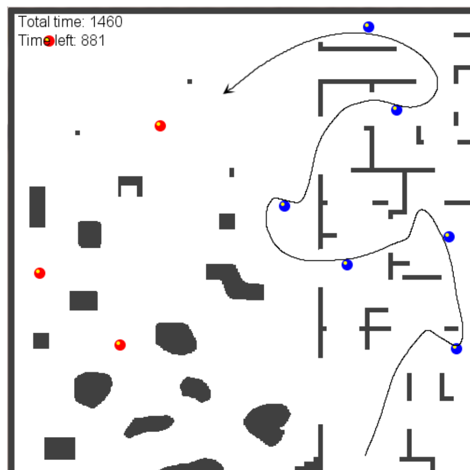

Fig. 1. Example of a PTSP map distributed with the framework.

B. Maps

The PTSP maps are two dimensional levels where all the entities of the game (ship, waypoints and walls) are located. The format of the maps is based on the one used by Nathan Sturtevant, from games such as Warcraft, Starcraft or Baldur's gate. These maps have been often used by many researchers in the literature [25]. The number of waypoints in a map employed in this study is 10, although this is not a fixed number and can be changed to create maps with more waypoints. An example of a PTSP map is depicted in Figure 1.

To consider a map a valid PTSP level, the following conditions must be addressed:

- Connectivity of waypoints: there must be a valid path between each pair of waypoints in the map. This can be achieved by computing the A* path between them using the pathfinding library, as shown in Section III-C.
- Initial position for the ship: the position generated to place the ship at the beginning of the game must avoid any collision with close walls.
- Positions for the waypoints: as in the previous case, each waypoint's position must be obstacle free.
- Ship vs. waypoint distance: the initial position of the ship and all waypoints must be significant to avoid too quick visits. The minimum distance is set to 10 times the radius of the ship.
- Waypoint vs. waypoint distance: the distances between every pair of waypoints cannot be too low to avoid multiple visits at once. The minimum distance is set to 5 times the radius of the ship.

C. Framework utils

The benchmark includes the possibility of executing different controllers and maps, which can be read from static

files or created dynamically from data structures. The PTSP framework includes path finding and line of sight features, that are used by the controllers employed in this study.

The path finding library builds a grid graph placing nodes in the navigable parts of the maze. A node will be added to the graph if a certain position is free of obstacles. Starting from the top left corner of the map, different positions are checked by adding a certain value to the coordinates (granularity), until the bottom right corner is reached. The adjacent nodes of the graph are linked to each other following an eight-way connectivity scheme: each node can be connected to eight neighbour nodes: up, right, down, left and the four diagonals. The controller can query the graph for shortest paths from any position in the map to another.

Regarding the line of sight feature, the framework includes a method to find out if there is a clear line of sight between two positions in the map. In other words, this means that no obstacles are found in a straight line between those positions, considering the radius of the ship as the width of the line.

D. The PTSP Competition

The PTSP software has been used to run a competition for the WCCI (World Congress on Computational Intelligence) and CIG (Computational Intelligence and Games) 2012 [19]. In the competition, the participants can download the code and submit a controller that tries to get the best score over a set of 20 different maps. The objective of the competition is to see the different approaches that are proposed to create controllers for this game. These must show a good balance between completeness (visiting as many waypoints as possible) and speed (time spent to visit the waypoints).

Each controller is evaluated 5 times in each one of the 20 maps of the final stage of the competition, and only the three best results, considering the number of waypoints visited, are used to compute the average score in that map. Points are awarded for each map, depending on the performance demonstrated, and the participant with the highest sum of points wins the competition.

A total of 64 maps were created manually for the PTSP Competition. This took a significant amount of time and effort in order to create valid and challenging levels. One of the objectives of this study is to show a way of generating PTSP maps automatically, in order to minimise the creation costs and obtain maps with certain challenging features, as described in Section IV.

IV. Experimental Study

The objective of this research is to obtain maps where controllers that invest time in finding a waypoint order that takes the physics of the game into account are most likely to obtain better results than others that follow a more naive approach (nearest waypoint or distance-based TSP routes).

A solution is obtained by dividing the problem into two very well differentiated tasks: 1) calculate the order of waypoints; and 2) drive the ship to visit the waypoints in the order specified. In the terminology used in this paper, *controller* must be seen as the combination of the *route planner* (which obtains the order of waypoints) and the *driver* (which effectively applies the actions to move the ship).

This section of the paper presents the algorithm used to evolve maps in Section IV-A. Then, the route planners are described in Section IV-B, followed by the explanation of how the proposed routes are evaluated, in Section IV-C. Finally, the procedure to evaluate individuals (or maps) is detailed in Sections IV-D and IV-E.

Element	Data	Description (Coordinates)
Lines	$(x_o, y_o, x_d, y_d) * L$	Starting and ending positions.
Rectangles	$(x_o, y_o, x_d, y_d) * S$	Top-left and bottom-right corners.
Ship	(x, y)	Ship's starting position.
Waypoints	$(x_w, y_w) * 10$	Waypoints' positions.

TABLE I
Representation of an individual or map.

A. Evolutionary algorithm: CMA-ES

In this first attempt at automatic generation of maps for the PTSP, an evolutionary algorithm is proposed. This section describes the algorithm employed and the representation of maps as individuals in the population.

Each one of the maps considered in this study is composed of a set of floating point values that encode the starting position of the ship, the waypoints and the location and size of obstacles such as lines and rectangles. Table I details the representation of an individual of the population. The order of the rows indicates the order of elements in the string of values.

As can be observed, two different parameters are needed to define the contents and length of an individual: the number of lines (L) and the number of rectangles (R). The experiments described in this paper are performed with maps that contain $L = 15$ lines and $R = 8$ rectangles, although initial experimentation shows that it is also possible to evolve maps with different values for these parameters. With these settings, the length of the individual is then 114, and each one of these genes takes a real value in the range $[0.05, 0.95]$. When the genome is read to create a map, these values are scaled to the size of the map, which for this study is established to 500×500 pixels.

An important feature that must be determined for each individual is whether it encodes a feasible map or not, according to the rules described in section III-B.

Some infeasible maps are able to be repaired. One of the reasons why a map can be invalid is because at least one of the waypoints or the starting position is in the same position as an obstacle, or too close to one. Another possibility is that two of these entities (waypoints and starting position) are too close to each other. If a map is invalid because of one of these reasons, a *repair mechanism* tries to change the location deterministically, moving one of these entities along the vertical and horizontal axis until a valid position is found.

It might be the case that this simple repair procedure is not able to fix the problem in the map, or that another of the problems described in section III-B is the cause of its infeasibility (i.e.: unreachable waypoints). In this case, the map is considered invalid and it is flagged as such.

The evolutionary algorithm employed in this research is the Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES). CMA-ES is an algorithm specially suited for high dimensional continuous domains [8]. CMA-ES is based on an iterative process that updates a multivariate normal distribution (MND). The population at a given generation is obtained by sampling from the distribution, $\mathcal{N}(m, C)$, which is uniquely defined by the distribution mean $m \in \mathcal{R}^n$ (that determines the translation of the distribution) and the covariance matrix $C \in \mathcal{R}^{n\times n}$, which defines the shape of the MND. Each individual x_i is sampled from this distribution according to a step-size σ, so that $x_i \sim m + \sigma\mathcal{N}(0, C)$. At each iteration, the values of m, σ and C are updated in order to minimize the fitness of the individuals drawn from the MND. For a more detailed description of the algorithm, the interested reader is referred to [8].

If an individual happens to be infeasible, the algorithm creates a new randomly initialized map (drawn from the MND) and checks for its feasibility again, repeating this process until a feasible one is created. In this way, a population is always composed only of feasible individuals. Therefore, CMA-ES creating infeasible individuals only affects how quickly the experiments are run. Although it would be possible to design a more involved repair mechanism that would reduce the number of rejected individuals, the experiments performed in this research show that the number of infeasible individuals sampled from the multivariate distribution reduces as the algorithm converges towards a solution (with a rate of infeasible individuals less than 5%). This phenomenon is not new, and has been reported previously in the literature [5].

The number of generations of each experiment is established at a maximum of 1000 generations, although a fitness-based stopping criterion can finish the run earlier: if the range of the best fitness obtained during the last $10 + (30 \times N/\lambda)$ generations is smaller than 10^{-13}, the experiment stops. For the experiments presented here, N is the problem dimension (114) and λ the population size (100), so this condition must be fulfilled in 44 consecutive generations. This termination criteria, known as *TolHistFun*, is a default stopping condition of CMA-ES, and has been used in the literature before [4], [9]. The default value of the population size in CMA-ES is $4+3\times log(N)$, which for this problem would be 18. However, the population size used in the experiments presented here is set to 100, a value determined empirically.

B. Route planners

The route planner is in charge of determining the order of waypoints that the driver must follow during the game. The route planner considers the cost of travelling from position A to B as the distance of the path given by the graph, which is calculated using the A* algorithm.

Three different variants of route planners have been employed in this study:

- **Nearest-first TSP:** This TSP solver, and the route it produces, is referred to in this paper as N_{TSP}. The order of waypoints is obtained by applying the nearest first algorithm to solve the TSP. That is, from the current location, the algorithm sets the closest waypoint as the next waypoint to visit, repeating this procedure until all waypoints are in the plan.
- **Distance TSP:** This TSP solver, and the route it produces, is referred to in this paper as D_{TSP}. The planner uses the Branch and Bound (B&B) algorithm to determine the order of waypoints, using the length of the A* as the cost between each pair of waypoints.
- **Physics TSP:** This TSP solver, and the route it produces, is referred to in this paper as P_{TSP}. As in the previous case, the B&B algorithm is employed for the order of waypoints, but in this case the cost between two waypoints is affected by physical conditions such speed and orientation of the ship.

The *Distance TSP* route planner resembles the optimal TSP path, and it would be the perfect choice if certain physics conditions, such as the ship's inertia, were not present in the game. In contrast, *Physics TSP* has been prepared in order to take into account the nature of the game. As has been suggested before, inertia and navigation should be taken into consideration to calculate the optimal PTSP route. The overall idea is based on the fact that the ship can benefit from visiting waypoints that are in the same straight (or quasi-straight) line, even if the distance between them is not short, as this way the ship maintains its velocity. In other words, minimizing changes of direction within the route - that would cause the ship to lose inertia and speed - is important when computing the route.

In this case, the cost of the path is obtained by an approximation of the time needed to drive the route, as described in Algorithm 1. Given an order of waypoints r, a path p is first obtained where each node is in line of sight with the following in the path (GETINSIGHTPATH($route$) function). This way, the controller would be able to drive between each pair of nodes in a straight line. Starting with an initial speed of 0, the algorithm traverses the path whilst calculating the time taken for the ship to go from one node to the next. This calculation uses the real physics of the game, so the time taken between two nodes is completely accurate, given the selected action sequence, though usually not optimal.

The overall calculation is, however, an approximation, owing to the way the speed is kept between each straight line segment. The dot product of the angle of two consecutive segments is calculated and used to decrement the speed at each turn. If this value is 1 (0 degrees), the penalization (pen) value is 1 and the speed does not decrease. The penalization increases exponentially with the angle, reaching 0 (complete

Algorithm 1 Route cost estimator.

```
function HEURISTICSOLVER(r)
    p ← GETINSIGHTPATH(r)
    speed ← 0
    for all Node n_i in p do
        d ← EUCLIDEANDISTANCE(n_i, n_{i+1})
        t ← TIMETOTRAVEL(d, speed)
        dot ← DOT(V(n_i, n_{i+1}), V(n_{i+1}, n_{i+2}))
        pen ← GETPENALIZATION(dot)
        speed ← speed * pen
        totalTime ← totalTime + t
    return totalTime
```

stop) when the turn to make is of 180 degrees. The final estimated cost of the route is the sum of the time taken to drive all segments of the path given.

C. Evaluating routes: drivers versus estimations

The driver is the agent that makes the moves in the game, trying to visit all waypoints in the order specified by the route planner. The main problem of evolving maps using drivers to evaluate the different routes is the computational cost it involves. Two different approaches have been taken to evaluate the routes provided by the route planners:

- **Estimated Cost (EC)**: The estimated cost of a route r, $EC(r)$, indicates an upfront value that determines the quality of a route in the map. It is calculated by applying Algorithm 1 on the route given. There is no need for a driver to actually play the game to determine the EC of a route, and the cost is an estimation of the time taken by any driver that follows it.
- **Cost (C)**: The cost of a route r, $C(r)$, is determined by using a driver to complete the game following the order of waypoints provided by the route planner. This value is calculated by actually playing the game, and it takes into account the time spent (*TotalTime*), the remaining waypoints still to be visited when the game finished (*RemWaypoints*) and a penalty *P* equal to the maximum time allowed to visit the next waypoint in the route (1000 time steps). Hence, the cost C of driving in a map using a route r is:

$$C(r) = TotalTime + RemWaypoints * P \qquad (1)$$

In both cases, the smaller the values of EC and C, the better the route, as they represent the time taken to complete the game. The main advantage of using a driver is that the evaluation is reliable in terms of playing the game (one can be certain that the map obtained produces a determined output for the driver used to evolve it), while an estimation is an abstraction that might contain errors and be inaccurate. However, using a specific driver impacts on the time needed to evaluate a route and might lead to maps that fit the navigation style of that particular driver. The estimation, on the other hand, provides a faster indication of the cost of the route, which is not tied to any particular driving style. An interesting trade off that has been employed in this study is to use the estimation for the evolutionary algorithm and, once the run has finished, play the game using a real driver in the maps obtained, in order to verify that the results are conclusive and the maps obtained have the desired properties.

The driver presented in this study is the Monte Carlo Tree Search (MCTS) driver, based on an earlier implementation fully described in [20]. Apart from the fact that this implementation provided good results in the past, MCTS was also used in this game by the winner of both editions of the PTSP competition, and it seems to be, to date, the strongest driver for this game.

MCTS is a stochastic algorithm that combines the strength of Monte Carlo simulations at exploring the search space with a tree search policy that selects among the available actions to take. This policy, known as the Upper Confidence Bounds for Trees (UCT), exploits the most promising parts of the search space while exploring other actions that do not seem to lead to optimal solutions. This trade off allows the creation of an asymmetric tree that grows towards interesting parts of the search space.

The vanilla algorithm is divided into four steps that are repeated in a loop: first, in the *selection* step, the UCT policy decides a move in the search tree, balancing between exploration and exploitation, until the action chosen has no representative node in the tree. Then, during the *expansion* step, a new node is added to the tree and a Monte Carlo simulation is run until the end of the game is reached, constituting the *simulation* step. Finally, the reward of the simulation (win, loss or score) is propagated up to the root during the *back-propagation* step. A more extensive description of the algorithm, its variants and applications, can be found in [6].

The nature of PTSP poses some interesting challenges to MCTS. The most important one is due to its real-time feature: the amount of simulations that can be performed at each cycle is very limited, and the end of the game is so far away in time that it is usually impossible to reach within the time limitation. For this reason, a score function value is used at the end of each simulation in order to provide a quality measure of the state reached. This function returns a score based on the waypoints visited, the time taken during the game, the distance to the next waypoint in the route and the number of collisions with obstacles. The interested reader may refer to previous work [20] for details of the MCTS driver implementation.

D. Evaluating maps: fitness functions with 3 routes

The first objective of this research is to be able to obtain maps where the results obtained with the same driver using distinct routes are different. The quality of a map will be better if it rewards more involved routes than simpler waypoint orders.

Each individual (or map) of the evolutionary algorithm is evaluated measuring the EC values (as described in

Section IV-C) of the different routes that the route planners provide. Let us say that in a given map, the three route planners provide three different routes: N_{TSP}, D_{TSP} and P_{TSP} (as defined in Section IV-B). Then, the objective is to achieve:

$$EC(N_{TSP}) > EC(D_{TSP}) > EC(P_{TSP})$$

In other words, the estimated cost of using the nearest-first TSP route planner is higher than using the distance TSP route planner, and this cost is also worse than employing the physics TSP route planner.

In order to achieve this, two different fitness functions are employed and defined in this section. Given these routes, it is possible to define the fitness as the result of the following equation[2]:

$$f_3 = -Min(EC_N - EC_D, EC_D - EC_P) \quad (2)$$

An analogous fitness can be defined using the real cost (as defined in Equation 1) of a driver playing the game as:

$$f_3' = -Min(C_N - C_D, C_D - C_P) \quad (3)$$

CMA-ES is set up to minimize this fitness, so high negative values are better. As this fitness measures the distances between the costs of a naive and a more complex route (both $EC_N - EC_D$ and $EC_D - EC_P$), it must be understood as the amount of time steps saved when using a more involved route instead of a simpler one for solving the problem.

E. Evaluating maps: fitness functions with 5 routes

Additionally, it is also a purpose of this research to obtain several near-optimal solutions so the best route is not too obvious. In other words, the evolutionary algorithm must be able to generate maps where a group of N routes of the type P_{TSP} (those obtained with the Physics TSP solver) produce a similar performance among them, but all better than a D_{TSP} route, which is still better than an N_{TSP} one.

The P_{TSP} route planner presented in Section IV-B provides only one route: the best achievable, considering the cost between waypoints, derived from taking the physics of the game into account. However, other routes can be derived from this one applying the operators *2-Opt* and *3-Opt*. These operators exchange 2 (or 3, respectively) nodes in the path to create a new solution. Hence, if M additional routes are needed, the first step is to calculate the best one with Algorithm 1, as usual. Then, *all* possible derivatives from this route are obtained applying *2-Opt* and *3-Opt*. They are sorted by ascending cost and the M best ones of this new group of routes are selected.

The fitness function is similar to the one described in Equation 2, substituting the best P_{TSP} route for the i_{th} one in the group of routes. For instance, let us say that the objective is to create maps where 3 routes of type P_{TSP} obtain similar performance when followed by a determined driver. If any of these 3 routes is followed, the performance must be better than following a route of type D_{TSP}, and this should outperform an N_{TSP} route. An initial P_{TSP} route is obtained with Algorithm 1 (i.e. $P_{TSP} \equiv r_0$), and *2-Opt* and *3-Opt* operators are employed to derive all routes from this one. These new routes are sorted by ascending cost (i.e.: r_1, r_2, ..., r_m), being $P_{TSP} \equiv r_0$ better than all these by construction. The fitness function is then defined as follows:

$$f_5 = -Min(EC_N - EC_D, EC_D - EC(r_2)) \quad (4)$$

As in the previous section, a fitness for real drivers can be defined such as:

$$f_5' = -Min(C_N - C_D, C_D - C(r_2)) \quad (5)$$

By obtaining maps that maximize the cost difference between these three routes, the algorithm provides individuals where route r_2 is better than route D_{TSP}. As r_0 and r_1 are by definition better than r_2, the resultant maps are *separating* the costs of D_{TSP} and the group (r_0,r_1,r_2).

The experiments described in this paper show that it is possible to evolve maps that distinguish between three (as in Equation 2) and five (Equation 4) routes. Initial experiments have shown that it is also possible to evolve maps with 8 different orders of waypoints, five of them forming the group of P_{TSP} routes r_0 to r_4.

[2]For the sake of clarity, $EC_X = EC(X_{TSP})$ and $C_X = C(X_{TSP})$

V. Results and Analysis

This section details the results of the experiments performed during this research. A total of 40 independent runs have been executed for each one of the two batches of experiments (for 3 and 5 routes). The following parameters have been set up:

- CMA-ES as described in Section IV-A. The initial mean m of the MND is set to 0.5 and the step size $\sigma = 0.17$ ($\sigma = m/3$, in order for CMA-ES to converge with ± 3 standard deviations). The size of the population, 100, was determined experimentally.
- Three different route planners are employed: N_{TSP}, D_{TSP}, P_{TSP}.
- Each route is evaluated using the heuristic cost estimator (Algorithm 1): no driver plays the game to evaluate the routes during evolution.

A. Heuristic Cost Estimation: 3 routes

For these experiments, the fitness function used is f_3 (Equation 2) to evaluate the maps. The three routes used are the ones provided by the three different route planners.

1) Evolution of fitness: Figure 2 shows the evolution of the averaged fitness during the experiments run. This picture shows the average (plus standard error), of the best individual of each generation across all experiments run for this setting.

As can be seen, there is a clear evolution in the fitness of the runs. At the beginning of the experiments, the best individuals obtain a fitness close to $f_3 = -100$ (as explained earlier, this represents the minimum difference of cost between

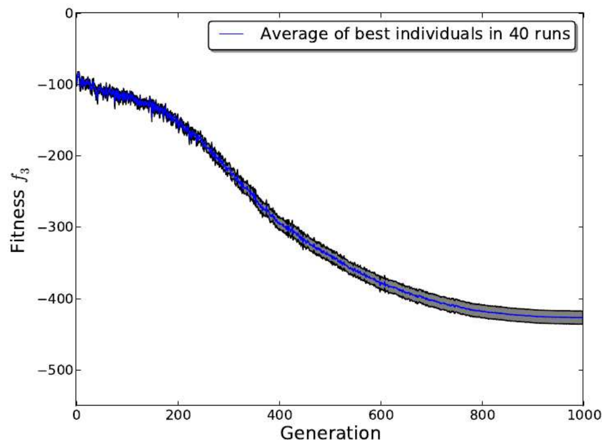

Fig. 2. Averaged evolution of the best individuals per generation of the 40 runs executed with the heuristic cost estimator for routes, employing 3 routes per map. Shadowed area indicates the standard error of the measure.

each pair of the 3 routes used to evaluate the maps.). By the end of the executions, the average of the best individuals achieve a fitness of $f_3 = -427 \pm 9.36$. Approximately 80% of the runs were stopped by the *TolHistFun* termination criteria, as explained in Section IV-A, not reaching the maximum number of generations set at 1000.

The MCTS driver, defined previously in Section IV-C, has been used to drive the best maps of the final generation of all experiments. The routes taken have been N_{TSP}, D_{TSP} and P_{TSP}, and each one has been played five times (adding up to $3 \times 5 \times 40 = 600$ games played). The average fitness, as described in Equation 3, obtained by the MCTS driver in these maps is $f_3' = -321.44 \pm 24.99$. This value cannot be directly compared with the fitness achieved by the estimated cost heuristic, because they are obtained by a different procedure (playing the game versus not playing it). However, it does show that the maps obtained are sensible: the difference of taking any pair of the given routes is of at least 321.44 ± 24.99 time steps on average.

A different way to analyze this result is to compute the relative difference of fitness between the routes taken. This is done in the following way: if the time spent by driving route P_{TSP} is taken as a reference, it is possible to calculate the increment of time spent by following routes N_{TSP} and D_{TSP}. Whereas P_{TSP} takes a reference value of 1.0, D_{TSP} obtains a worse performance of 1.43 ± 0.032, and N_{TSP} spends even more time to obtain a value of 1.692 ± 0.038.

This analysis is interesting because it provides a more detailed view of the time taken per route. As can be seen, the driver that takes route D_{TSP} spends $(43 \pm 3.2)\%$ more time than following P_{TSP}, and taking N_{TSP} spends $(69 \pm 3.8)\%$ more game steps than the physical route. This measure also provides a sense of order, that matches the goal of the experiments: taking route P_{TSP} is better than driving through D_{TSP}, which is still better than following N_{TSP}.

The maps obtained by the runs can also be compared with those maps hand-crafted for the WCCI 2012 PTSP Competition. Table II shows the estimated costs of the three possible routes and the fitness associated with each one of the maps distributed with the competition framework. As can be seen, the fitness of the competition maps is very different from the results obtained at the end of the experiments presented in this paper. Indeed, sometimes it is even better not to follow the routes proposed for the physical TSP route planner.

Map	$EC(N_{TSP})$	$EC(D_{TSP})$	$EC(P_{TSP})$	$Fitness f_3$
1	1298	1103	1091	-12
2	1010	952	922	-30
3	1032	1001	990	-11
4	1146	1146	1152	6
5	1558	1415	1415	0
6	1197	1269	1121	72
7	1070	1070	1070	0
8	1357	1281	1281	0
9	1463	1415	1337	-48
10	1366	1014	1014	0

TABLE II
ESTIMATED COSTS f_3 ON MAPS OF THE 2012 PTSP COMPETITION.

If all maps from the WCCI 2012 PTSP Competition (64 maps: the ones from the framework and the other 54 used to rank the entries) are taken into account, the average fitness of the maps using the estimated cost is $f_3 = -12.06 \pm 4.06$, showing that it is not straightforward to create maps by hand in which the most involved route planning leads to a clear victory. It is worth mentioning, however, that the maps designed for the competition were not only created with the aim of being a challenge, but also with the objective of making them aesthetically pleasing, a feature not considered in this research.

It is also interesting to compare how the evolved maps differ from randomly created maps. In the case of three routes, the fitness of 100 randomly initialized maps (after being repaired) is, using the estimated cost, -4.24 ± 1.95; very different from the -427 shown in Figure 2.

2) A representative example: Figure 3 shows the map and three routes obtained in one of the runs explained in this section. The MCTS verification step produced averages of 2649.75 ± 78.03, 2037.6 ± 43.82 and 1658.0 ± 37.93 for each one of the routes N_{TSP}, D_{TSP} and P_{TSP}, respectively. According to the relative performance, it can be seen that the D_{TSP} and N_{TSP} routes spend respectively around 22% and 59% more time steps than P_{TSP} to visit all waypoints and complete the game. This picture also backs up a concept previously mentioned in this paper: routes that visit waypoints in a straight (or almost straight) line can take advantage of the speed of the ship, even if the distance travelled is higher. In this case, it can be seen that the P_{TSP} route has much fewer changes in direction than the other two and a good balance between travelling a long distance (as the N_{TSP} route) and the shortest possible distance (given by D_{TSP}).

Figure 4 shows the evolution of the fitness f_3 of the best individual of each generation during this particular run. It is

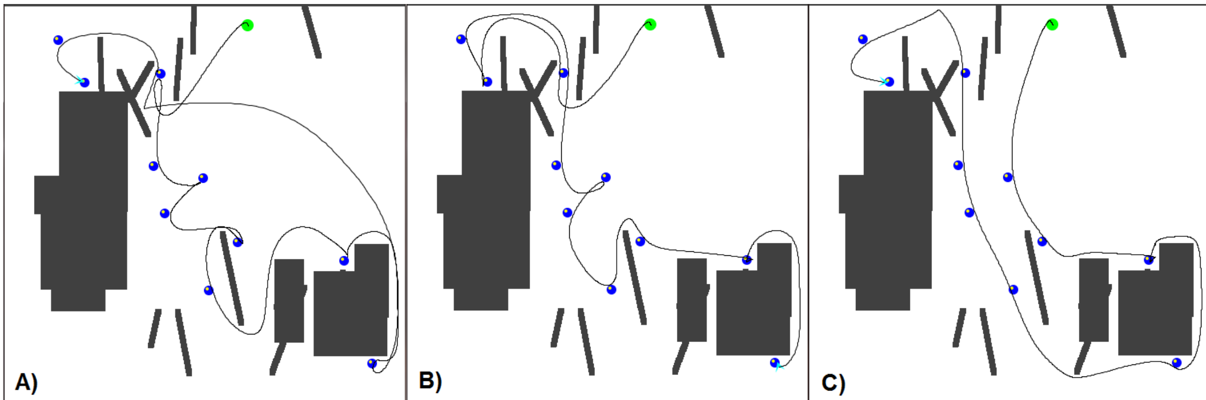

Fig. 3. Example of a map and its three routes evolved with CMA-ES. The trajectories, followed by the MCTS driver, are shown in the following order, from left to right: A) N_{TSP} route, with an average of 2649.75 ± 78.03 time steps; B) D_{TSP} route, with an average of 2037.6 ± 43.82 time steps; C) P_{TSP} route, with an average of 1658.0 ± 37.93 time steps.

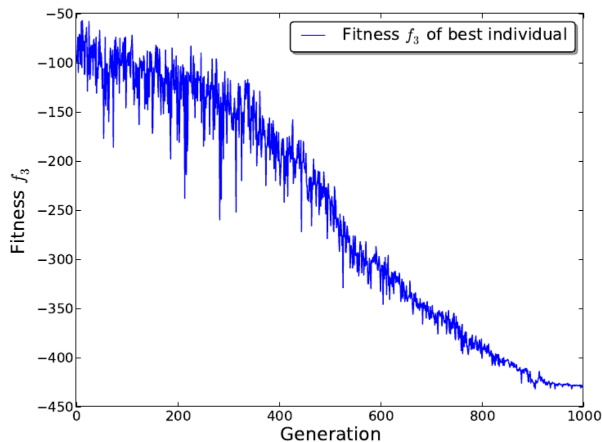

Fig. 4. Evolution of the fitness of the best individual during one of the 40 runs performed with 3 routes.

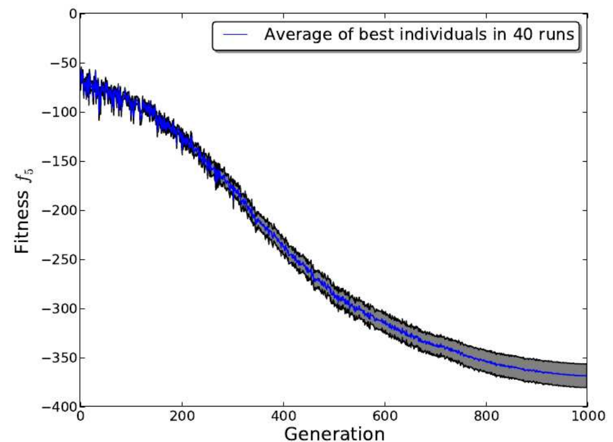

Fig. 5. Averaged evolution of the best individuals per generation of the 40 runs executed with the heuristic cost estimator for routes, employing 5 routes per map. Shadowed area indicates the standard error of the measure.

easier to observe in this figure, rather than in Figure 2, that during the first half of the generations given, the algorithm explores the search space, obtaining very different values for the f_3 measure. During this time, the best individual fitness decreases slowly. Once half the generations are past, the fitness values become less variable and decrease rapidly, converging towards a solution and reaching a stable fitness value of -430 by the end of the run.

B. Heuristic Cost Estimation: 5 routes

This section analyzes the results obtained after performing 40 runs of the algorithm to obtain maps that differentiate among 5 routes. The only difference in the setup with respect to the 3 route case is that now the fitness function employed is f_5 (from Equation 4). As explained in Section IV-E, r_2 is the second best route derived from P_{TSP} using the *2-Opt* and *3-Opt* parameters. Figure 5 shows the average of the best individuals on each generation in the 40 runs performed.

As in the 3 routes case, there is a clear improvement of the fitness. In this case, the average fitness of the best individuals of the last population is $f_5 = -369 \pm 12.08$. It is interesting to observe, comparing Figures 2 and 5, how the fitness obtained in this case is not as good as the ones obtained before for 3 routes. This is logical, owing to the fact that this scenario is more complex than the previous one, as the quality of the route r_2 is, by definition, the second best route obtained with the P_{TSP} solver. Although the progress of fitness is clear and significant, it visibly converges slower than when the P_{TSP} route is employed to calculate fitness. In this case, around 60% of the runs converged to a solution before reaching the maximum of 1000 generations.

As before, a verification phase with the MCTS driver has been performed to analyze the resultant maps, employing the fitness function declared in Equation 5. In this case, the average fitness obtained by the MCTS driver is $f'_5 = -196.27 \pm 23.45$, which is again smaller than its counterpart

N_{TSP}	D_{TSP}	P_{TSP}	r_1	r_2
1.53±0.022	1.39±0.02	0.99±0.01	1.01±0.01	1.0±0.0

TABLE III
RELATIVE AVERAGE PERFORMANCE IN BEST MAPS OF 40 RUNS.

Map	$EC(N_{TSP})$	$EC(D_{TSP})$	$EC(r_2)$	$Fitness f_5$
1	1298	1103	1126	23
2	1010	952	1020	68
3	1032	1001	1024	23
4	1146	1146	1183	37
5	1558	1415	1470	55
6	1197	1269	1172	72
7	1070	1070	1142	72
8	1357	1281	1368	87
9	1463	1415	1415	0
10	1366	1014	1135	121

TABLE IV
ESTIMATED COSTS f_5 ON MAPS OF THE 2012 PTSP COMPETITION.

for 3 routes, but still demonstrates that the maps obtained have the desired properties. As before, Table III shows the relative average fitness for the results obtained with the MCTS driver.

In this case, this table also shows the results obtained with the other 2 physical routes (P_{TSP} and r_1). It is clear that the proposed algorithm is able to find maps where three different routes (given by the physical TSP planner: P_{TSP}, r_1 and r_2) provide a similar performance, all of them better than the route planned by D_{TSP}, which is still better than N_{TSP}.

A comparison with hand-crafted maps from the 2012 WCCI PTSP Competition has also been made. Table IV shows the results of the heuristic estimated cost and the fitness values in the maps distributed with the competition framework. If all 64 competition maps are to be compared, the average fitness is $f_5 = 39 \pm 4.89$, which is a value much worse than the one obtained in the experiments described here.

Again, the results obtained in this section can be compared with randomly created maps. The fitness of these random maps, according to the fitness function f_5, is 24.0 ± 3.45. It is straightforward to see that the problem with 5 routes is more complex than using 3, as there is a clear difference between the fitness of random maps on both scenarios. A positive value like this indicates that the physics-based route r_2 is a worse choice than following the nearest waypoint first approach. This shows that it is not trivial to create maps with the features desired.

These comparisons are examples of one of the main advantages of using the technique described in this paper. If map designs are not good enough (as happened in the PTSP Competition), it may be better to follow a naive route (N_{TSP}) and obtain the same results as taking a more complex approach (D_{TSP}), as in map 7, or even better, as in map 6. This can be extrapolated to other games, where simulated based PCG can avoid unfair situations in which simpler players exploit deficiencies in levels to beat better players.

Average	N_{TSP}	D_{TSP}	P_{TSP}	r_1	r_2
Time	2279.0	1972.2	1626.0	1591.75	1650.2
Relative	1.38	1.19	0.98	0.96	1.0

TABLE V
PERFORMANCE OF THE MCTS DRIVER IN A RUN WITH 5 ROUTES.

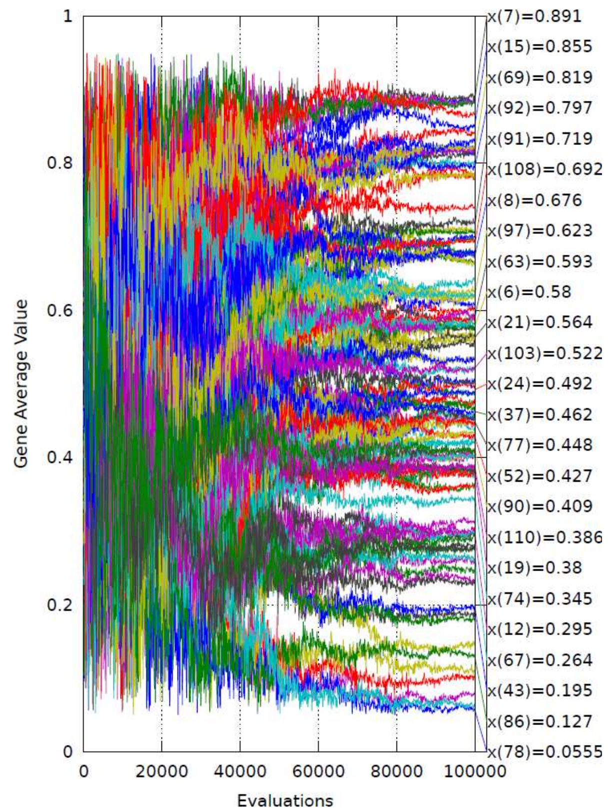

Fig. 7. Evolution of the values of some of the 114 genes of the individual. The average value of each gene is presented (between 0.05 and 0.95) against the number of evaluations performed by CMA-ES. The column on the right shows the final values for the mean m of the genes shown here.

1) A representative example: Figure 6 shows an example of one of the runs with 5 routes and the map evolved by CMA-ES. After running the MCTS verification step, the average fitness obtained is shown in Table V. As can be seen, the three physical routes provide a similar performance, while the distance and nearest ones need more time to be completed.

Figure 7 shows the average of (some of the 114) genes of the individuals during the evaluations performed in the run. It is interesting to see how, during the first half of the experiments, these values have a high variance, which corresponds to the exploratory phase of the algorithm. However, close to the end of the run, these genes stabilize and the standard deviation of each one of them is reduced significantly, showing that the algorithm is converging on a solution.

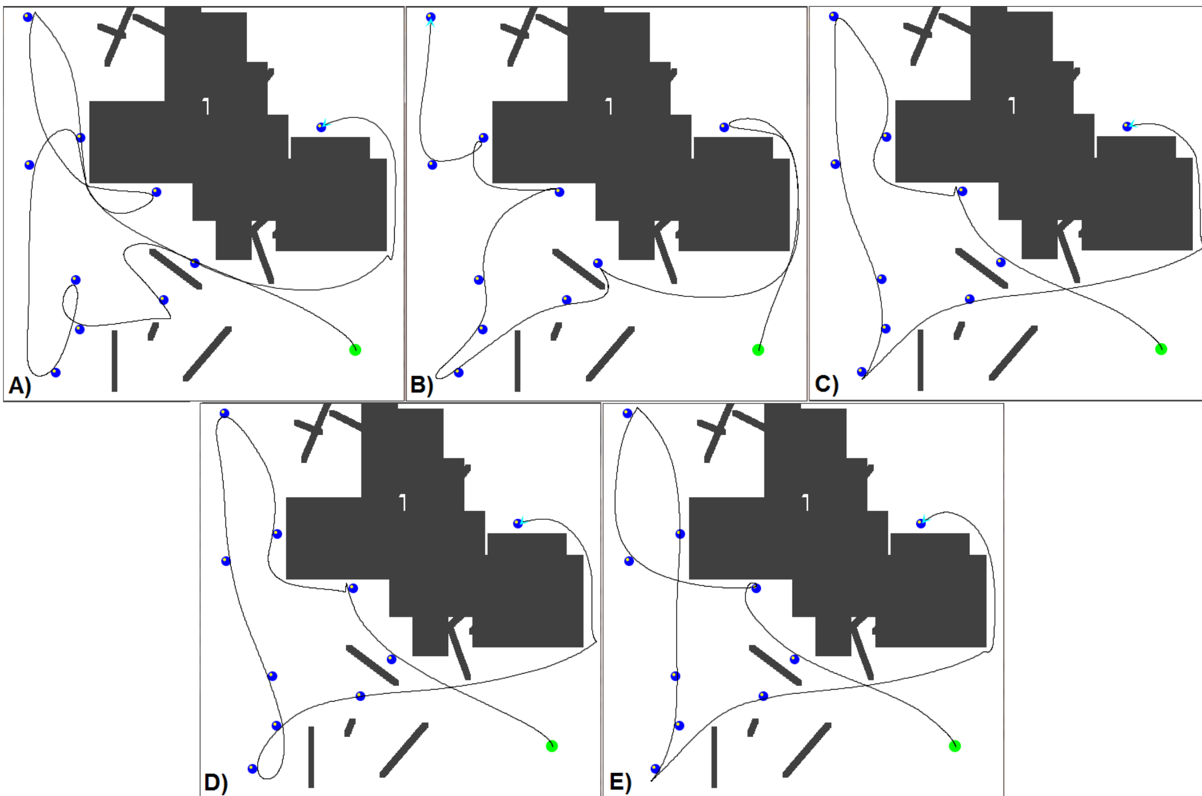

Fig. 6. Example of a map and its five routes evolved with CMA-ES. The trajectories, followed by the MCTS driver, are shown in the following order, from left to right, top to bottom: A) N_{TSP} route, with an average of 2279.0 ± 50.33 time steps; B) D_{TSP} route, with an average of 1972.2 ± 34.29 time steps; C) P_{TSP} route, with an average of 1626.0 ± 60.96 time steps; D) r_1 route, with an average of 1591.75 ± 60.97 time steps; E) r_2 route, with an average of 1650.2 ± 86.1 time steps.

VI. Conclusions and Future Work

This paper introduced an evolutionary algorithm capable of automatically generating high-quality maps for the PTSP game. This is, as far as the authors know, the first attempt at creating maps for this game in a procedural manner. Additionally, to the best of our knowledge, this is the first paper that employs CMA-ES for PCG in general.

The results presented in this research show the success of our technique, being able to create maps that fulfil the conditions required regarding the routes of waypoints that can be followed: naive and simple approaches are easily outperformed by those route planners that take the physics of the game into account when generating the routes. Additionally, the experiments performed in this research show that it is possible to generate maps where several trajectories provide solutions that are far better than the ones obtained by naive routes.

This research can be extended in several ways. For instance, the number of waypoints that a map contains could be increased, resulting in more challenging maps. Another possibility is to allow the evolutionary algorithm to modify slightly the rules of the game. For example, the algorithm could tweak time steps requested to visit each waypoint for a particular map (instead of the default 1000 value). The map generation algorithm could therefore adjust this value for each map in order to modify the difficulty of the level.

It would be interesting to attempt to find maps that maximally differentiate between different controllers. This could then be extended to a competitive co-evolution scenario, where controllers are evolved to beat the best maps and maps are evolved to differentiate between controllers, spurring a form of arms race.

Another interesting feature to look at, that has not been contemplated in this research, are the aesthetic aspects of the map. Seeing some of the maps presented in the figures of this paper, one might argue that they are not aesthetically very pleasant. The inclusion of new shapes, such as circles (or even more complicated figures), can lead to other types of maps through evolution. Another possibility is to include more obstacles once the evolutionary process has finished, adding different shapes in parts of the maze where the ship is unlikely to go (in order not to disrupt the routes and hence invalidate the evolved map).

Additionally, it would be desirable to undertake a thorough empirical study of different representations for the obstacles of the map, including the ones described here (lines and rectangles) and others such as Compositional Pattern-Producing Networks (CPPNs), Bezier curves, cell-based approaches or turtle-based graphics. It would be worthwhile to see how the representation affects the evolution of maps that are able to differentiate between several controllers.

Since the PTSP game is also available to be played by humans, another possibility is to create maps that are fun to play and research what features in the maps make a PTSP map entertaining.

Although this paper is centred on the PTSP, the conclusions and algorithms described here can also be applicable to other games and domains. As shown in this paper, it is not trivial to create game levels that are possible to solve and, at the same time, require an involved technique to be completed. Simulation based PCG allows the creation of maps or levels where a simplistic approach is not good enough to tackle the game, or other more complex behaviours clearly outperform the simpler ones. A game/level designer would not like to create levels with loopholes that can be easily exploited by naive approaches. Additionally, this research shows that it is possible to employ simulation based PCG techniques to evolve mazes with distinct levels of difficulty, or even propose maps where different solutions play with a similar performance, but much better than simpler approaches.

In a scenario where the use of PCG is extended to evolve certain parameters that affect the rules of the game, it would be possible for evolution to discard those rules that make the game unbalanced. A good example here is designing opposing armies for online strategy games. If some army units provide an army with a strong advantage (e.g. a very fast unit), it unbalances the game and detracts from the player experience. This balancing is currently performed by hand through a combination of game design and testing, but one can envisage a procedure where the system tweaks the rules of a game to improve certain playability measures.

The work presented in this paper can also be thought of as a Monte Carlo version of Reverse Game Theory/Mechanism Design [27]. The goal of mechanism design is to tweak parameters or discover how one can set up games in a certain manner so that, if all players act according to some predefined strategy (which can be adaptive), a desirable outcome can be achieved (e.g. total societal wealth will be increased). Mechanism design has been used to propose auctions that had a direct impact on real life [1]. The method presented here could possibly find uses in scenarios where analytical solutions are hard or impossible to find, thus providing another tool in an organization's toolbox. For example one can easily imagine a situation where an organisation can set up internal markets where players try to maximise their individual payoffs, however this maximisation should result in higher overall performance for the organisation.

Using the approach presented in this paper, the overall goal of the organisation would be encoded as a fitness function and the market mechanism as the game which agents would compete in. Evolution would then change the game (by presumably optimizing some parameters) in order to move towards higher overall payoffs. However, further research, this time closer to Complex Systems or Computational Sociology, is needed before one can draw conclusions about the suitability of these approaches for studying socio-economic phenomena.

ACKNOWLEDGMENTS

This work was funded by the EPSRC grant EP/H048588/1, as part of the project *UCT for Games and Beyond*.

REFERENCES

[1] Richard Adams. What is mechanism design theory? www.guardian.co.uk/business/2007/oct/15/ukeconomy.economics2, October 2007.

[2] Daniel Ashlock, Colin Lee, and Cameron McGuinness. Search-Based Procedural Generation of Maze-Like Levels. *IEEE Transactions in Computational Intelligence and AI in Games*, 3:260–273, 2011.

[3] Daniel Ashlock, Cameron McGuinness, and Wendy Ashlock. Representation in Evolutionary Computation. In *IEEE Congress on Evolutionary Computation*, 2012.

[4] A. Auger and N. Hansen. A Restart CMA Evolution Strategy With Increasing Population Size. In *Proceedings of Conference on Evolutionary Computation*, 2005.

[5] Zyed Bouzarkouna, Didier Yu Ding, and Anne Auger. Using Evolution Strategy with Meta-models for Well Placement Optimization. In *Proceedings of the 12th European Conference on the Mathematics of Oil Recovery ECMOR 2010*, 2010.

[6] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1:1–43, 2012.

[7] Cameron Browne. *Automatic Generation and Evaluation of Recombination Games*. PhD thesis, Queensland University of Technology, 2008.

[8] N. Hansen. The CMA Evolution Strategy: A Comparing Review. In J.A. Lozano, P. Larranaga, I. Inza, and Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.

[9] N. Hansen. Benchmarking a BI-Population CMA-ES on the BBOB-2009 Function Testbed. In *Workshop Proceedings of the GECCO Genetic and Evolutionary Computation Conference*, 2009.

[10] E. Hastings, R. Guha, and K. O. Stanley. Evolving Content in the Galactic Arms Race Video Game. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[11] Philip Hingston. A New Design for a Turing Test for Bots. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2010.

[12] Interactive Data Visualization Inc. (IDV). Speedtree. http://www.speedtree.com/, 2012.

[13] Sergey Karakovskiy and Julian Togelius. The Mario AI Benchmark and Competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1:55–67, 2012.

[14] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Neuroevolutionary Constrained Optimization for Content Creation. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 71–78, 2011.

[15] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Adapting Models of Visual Aesthetics for Personalized Content Creation. *IEEE Transactions on Computational Intelligence and AI in Games*, Special Issue on Computational Aesthetics in Games 2012:213–228, 2012.

[16] Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A. Pelta, Martin V. Butz, Thies D. Lönneker, Luigi Cardamone, Diego Perez, Yago Saez, Mike Preuss, and Jan Quadflieg. The 2009 Simulated Car Racing Championship. *IEEE Transactions on Computational Intelligence and AI in Games*, 2:2:131–147, 2010.

[17] Simon M. Lucas. Ms Pac-Man Competition. *ACM SIGEVOlution Newsletter*, pages 37–38, 2007.

[18] Zbigniew Michalewicz. A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. In *Evolutionary Programming*, pages 135–155, 1995.

[19] D. Perez, P. Rohlfshagen, and S. Lucas. The Physical Travelling Salesman Problem: WCCI 2012 Competition. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2012.

[20] Diego Perez, Edward J. Powley, Daniel Whitehouse, Philipp Rohlfshagen, Spyridon Samothrakis, Peter I. Cowling, and Simon Lucas. Solving the Physical Travelling Salesman Problem: Tree Search and Macro-Actions. *IEEE Transactions on Computational Intelligence and AI in Games*, (submitted):to appear, DOI: 10.1109/TCIAIG.2013.2263884, 2013.

[21] P. Rohlfshagen and S.M. Lucas. Ms Pac-Man versus Ghost Team CEC 2011 Competition. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 70–77. IEEE, 2011.

[22] N. Shaker, J. Togelius, G. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi, G. Smith, and R. Baumgarten. The 2010 Mario AI Championship: Level Generation Track. *IEEE Transactions on Computational Intelligence and AI in Games*, 3:4:332–347, 2011.

[23] N. Sorenson, P. Pasquier, and S. DiPaola. A Generic Approach to Challenge Modeling for the Procedural Creation of Video Game Levels. *IEEE Transactions on Computational Intelligence and AI in Games*, 3:229–244, 2011.

[24] Kenneth O. Stanley and Risto Miikkulainen. A Taxonomy for Artificial Embryogeny. *Artificial Life*, 9(2):93–130, 2003.

[25] Nathan Sturtevant. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4:144–148, 2012.

[26] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG)*, 3:172–186, 2011.

[27] Hal R Varian. Economic Mechanism Design for Computerized Agents. In *First USENIX Workshop on Electronic Commerce*, pages 13–21, 1995.

[28] Jim Whitehead. Toward Procedural Decorative Ornamentation in Games. In *Proceedings of the Workshop on Procedural Content Generation in Games*, 2010.

[29] Georgios N. Yannakakis and Julian Togelius. Experience-driven Procedural Content Generation. *IEEE Transactions on Affective Computing*, 2:147–161, 2011.

Diego Perez received a B.Sc. and a M.Sc. in Computer Science from University Carlos III, Madrid, in 2007. He is currently pursuing a Ph.D. in Artificial Intelligence applied to games at the University of Essex, Colchester. He has published in the domain of Game AI, participated in several Game AI competitions and organized the Physical Travelling Salesman Problem competition, held in IEEE conferences during 2012. He also has programming experience in the video-games industry with titles published for game consoles and PC.

Julian Togelius Julian Togelius is an Associate Professor at the IT University of Copenhagen (ITU). He received a BA in Philosophy from Lund University in 2002, an MSc in Evolutionary and Adaptive Systems from University of Sussex in 2003 and a PhD in Computer Science from University of Essex in 2007. Before joining the ITU in 2009 he was a post-doctoral researcher at IDSIA in Lugano. His research interests include applications of computational intelligence in games, procedural content generation, automatic game design, evolutionary computation and reinforcement learning; he has around 80 papers in journals and conferences on these topics. He is an Associate Editor of the IEEE Transactions on Computational Intelligence and AI in Games and the current chair of the IEEE CIS Technical Committee on Games.

Spyridon Samothrakis is currently pursuing a PhD in Computational Intelligence and Games at the University of Essex. His interests include game theory, computational neuroscience, evolutionary algorithms and consciousness.

Philipp Rohlfshagen received a B.Sc. in Computer Science and Artificial Intelligence from the University of Sussex, UK, in 2003, winning the prize for best undergraduate final year project. He received the M.Sc. in Natural Computation in 2004 and a Ph.D. in Evolutionary Computation in 2007, both from the University of Birmingham, UK. Philipp completed a series of post doctoral positions in evolutionary computation and games and is now a Principal Scientist working for SolveIT Software in Adelaide, Australia.

Simon Lucas (SMIEEE) is a professor of Computer Science at the University of Essex (UK) where he leads the Game Intelligence Group. His main research interests are games, evolutionary computation, and machine learning, and he has published widely in these fields with over 130 peer-reviewed papers. He is the inventor of the scanning n-tuple classifier, and is the founding Editor-in-Chief of the IEEE Transactions on Computational Intelligence and AI in Games.