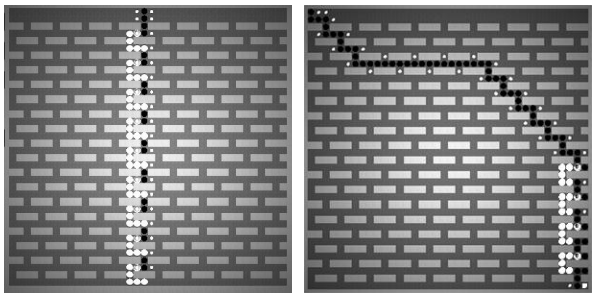
Pathfinding Based on Pattern Detection Using Genetic Algorithms

Ulysses O. Santos[1] Alex F. V. Machado[1] Esteban W. G. Clua[2]

[1]Instituto Federal de Educação Tecnológica do Sudeste de Minas Gerais,
Departamento de Computação, Brazil.
[2]Universidade Federal Fluminense, Instituto de Computação, Brazil.



(a) (b)

Fig. 12 – PPGA simulation in a tablet. Black circles represent the closed list, small white circles the open list and large white is the sub-path generated by GA".

Abstract

This paper presents a novel method to optimize the process of finding paths using a model based on Genetic Algorithms and Best-First-Search for real time systems, such as video games and virtual reality environments. The proposed solution uses obstacle pattern detection based at online training system to guarantee the memory economy. The architecture named Patterned based Pathfinding with Genetic Algorithm (PPGA) uses a learning technique in order to create an agent adapted to the environment that is able to optimize the search for paths even in the presence of obstacles. We demonstrate that the PPGA architecture performs better than classic A* and Best-First-Search algorithms in patterned environment.

Keywords: Pathfinding, Best-First-Search, Genetic Algorithm, optimization, electronic games.

Authors' contact:
ulyssesdvlp@gmail.com
alexcataguases@hotmail.com
esteban@ic.uff.br

1. Introduction

The search of the paths between two points is a fundamental problem for the area of electronic games, but it is also relevant for other areas [Stodola and Mazal 2010]. In this field the most used methods are A* [Russell and Norvig 2003] and Best-First Search [Russell and Norvig 2003]. Due to the demands of a real-time processing and hardware limitations, these traditional methods can cause problems in performance. Therefore many optimizations are studied for the same [Dechter and Pearl 1985][Leigh et al. 2007][Rios and Chaimowicz 2011].

The traditional algorithm A* uses f(n)=g(n)+h(n) heuristic, where g(n) is the cost to reach the node n, h(n) is the estimated cost for reach the goal node from the node n. This calculation can be performed by Manhattan distance. At each iteration neighboring nodes of the current node has its value f(n) calculated and added to the open list (O). After the node from the open list with the lowest value of f(n) is removed from it, added to the closed list (C) and becomes the current node. When the current node is the destination node (G) the algorithm returns the path found. From this algorithm we can find the optimal solution. As can be seen in Fig. 1, the usage of compound heuristic provides a considerable expansion of the nodes, and these are kept in memory during processing. By applying these methods in multi-agents systems such as RPGs and strategy games in real time, memory consumption becomes much higher.

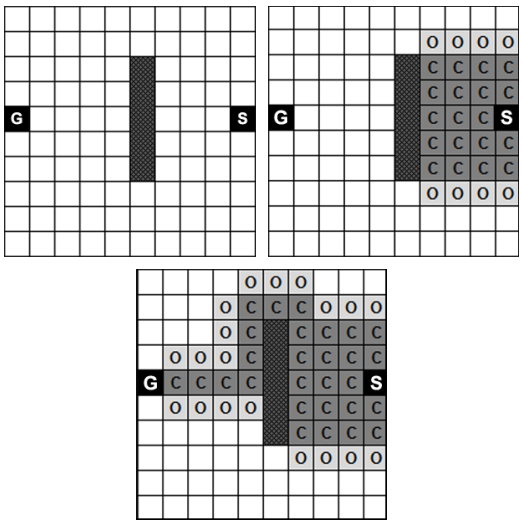


Fig.1. A-Star route sample.

The algorithm IDA * [Korf 1985] is designed for the efficient use of memory. It is based on depth-first search using the same heuristic as algorithm A*, ensuring that finds the shortest path and decreasing the number of nodes expanded impacting the amount of memory required by the search. Although requiring less memory than the A*, it takes longer to find the path.

The Best-First-Search algorithm works similarly to A*, but the heuristic adopted is simple and greedy (f (n) = h (n)), tending to expand only the nodes that are close to the goal one, thus the amount of memory required is less than the A* algorithm (as shown in Fig. 2) as a result the solution will not necessarily optimal.

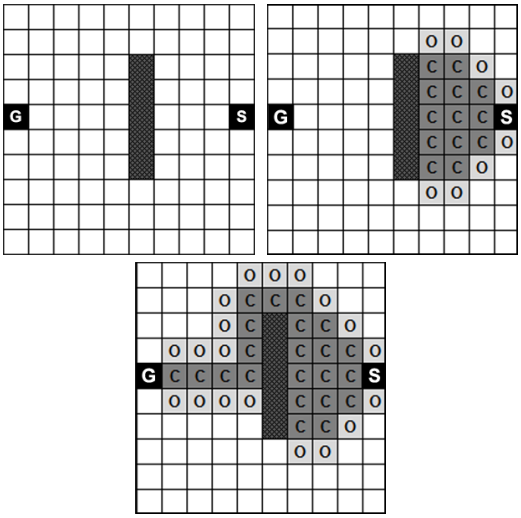


Fig.2. BFS route sample.

This paper proposes a promising algorithm, which the same as IDA*, optimizes the expense of A* memory, but as the Best-First, without guaranteeing the definition of the optimal path. For this, we designed an architecture based in Genetic Algorithms [Mitchell 1996] and in the Best-First Search to optimize the search for paths with a reduction in the number of nodes expanded, and obtaining a path that is not always optimal. With this technique, the agent is able to adapt itself in an homogeneous maze, using an heuristic that let the agent calculates a path thought a considerable number of open nodes, in a single step, without having to expand them.

In our previous work [Machado et al. 2011] we proposed an approach that aims to reduce the number of nodes accessed using a module with GA but with a local search heuristic adopted by A* to move through the environment. Compared with the traditional A* in patterned environments with obstacles, our previous method showed superior performance, but in environments without any kind of pattern obstacles, the performance was far below, not being suitable to many practical cases. Unlike the RTP-GA, our new approach ensures a minimal loss compared to the traditional Best-First in maps without pattern, maintaining a satisfactory standard advantage in environments with obstacles.

We show in [Machado et al. 2011] that the decrease in the number of access nodes is proportional to the processing gain. As investigated by [Korf 1985] and [Russell and Norvig 2003] this relationship is not always proportional. Thus, to avoid this error, in this work we suggest only space optimization (memory).

This papers is organized as following: chapter 2 presents a survey about related works of pathfinding, followed by the PPGA algorithm specifications and features on chapter 3. At chapter 4 we present our tests and comparative analysis between the PPGA, BFS and A* algorithm. Finally, at chapter 5, we present our conclusion and future work.

2. Related Work

Searching for patterns in maps in order to optimize the pathfinding algorithms is a well-known technique. [Demyenand and Buro 2006] shows that the Triangulation Reduction A* (TRA*) method has many benefits, including accurate representation of polygonal environments, reduction of the search space and refinement of the optimal path. Although this algorithm uses real-time learning, it has severe constraints regarding the environment. To use the TRA* we must provide a description of polygonal obstacles in the environment, which can be easily applied to maps with barriers and holes but it is not suitable for labyrinths and mazes.

The use of meta-heuristics for adaptation to the environment is a topic addressed by several authors. [Leigh et al. 2007] proposed the GA-Manufactured Maneuvering Algorithms (GAMMAs) that was able of finding paths more than 1000% faster than the traditional A*. In presented architecture the GA was able to adapt to the obstacles (called risk zones), approach the human-made path and define the best way to evaluate the path. However GAMMA depends on a prior offline training, that consists in exploring some maps in order to extract some patterns before the optimization process.

[Burchardt and Salomon 2006] implemented a Genetic Algorithm to search for paths for robot routing. The algorithm runs until the agent achieves its goal. There is a constant update process on the environment. The fitness function is based on the length of the path with penalization on collisions. The gene encoding follows the pattern:

Length	X_1	Y_1	X_2	Y_2	X_3	Y_3

The path is encoded as equidistant distances between the starting and ending point, allowing the occurrence of a deviation. However, this approach only works when a nearly direct route exists.

[Bulitko and Lustrek 2006] call as "lookahead" the incomplete search method, which is similar to the min-max based algorithms used in two-person games. It conducted a limited depth search, expanding the space centered on the agent, and heuristically evaluates the

distance between the agent and the destination. However, due to the "lookahead" pathology [Bulitko and Lustrek 2006], determining the optimal depth for the search procedure is not an easy task.

As in [Demyenand and Buro 2006] and [Burchardt and Salomon 2006], this work investigates partial solutions in the map, trying to minimize the problems defined by [Bulitko and Lustrek 2006] when using this incomplete search method. We intend to hold a learning process through an online training (as opposed to [Leigh et al. 2007]), based on a codification of the route, such as presented in [Burchardt and Salomon 2006].

Therefore we decide to use Genetic Algorithms for solving the pathfinding problem combining two approaches of the literature, presented in [Leigh et al. 2007] and in [Burchardt and Salomon 2006]. The difference between our model and the one presented in [Leigh et al. 2007] is that we use an online training instead of an offline one. Our approach also improves the model presented in [Burchardt and Salomon 2006] since we define a chromosome able to adapt to any environment that present obstacle patterns.

We generate possible sub-paths (or "lookahead") to be randomly applied in a map, evaluate the sub-paths in the map, and then adapt the sub-paths in order optimize the amount of memory necessary to generate the path. In Fig. 3 we show an example of obstacle avoidance:

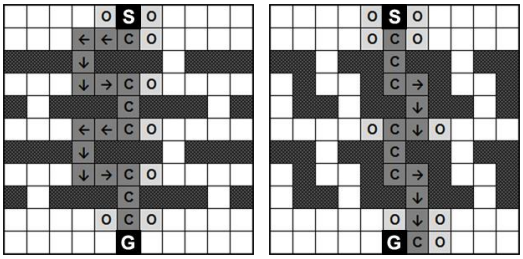


Fig 3: Two examples of obstacles adapting with Patterned based Pathfinding with Genetic Algorithm. The arrows represent the pattern detected by the GA, treated in this way sub-paths, which do not require evaluation of the neighbors.

3. PPGA

Patterned Based Pathfinding with Genetic Algorthm (PPGA) uses the classical Best-First-Seach with its greedy heuristic to find the path. When the criterion for the use of GA is reached, the fittest individual elected by the same will be used in the environment.

The Genetic Algorithm iterates over a population of solutions that are called individuals or chromosomes. On each generation (algorithm iteration), new individuals are generated by crossover and mutation process and only the most suitable individuals survive for the next generation. The individual fitness is calculated by a fitness function, or objective function. Thus, in each generation, only the individuals with the best values of objective function are selected to be in the next generation.

It is important to note that, in the presented architecture, the quality of individuals (paths) is measured not only by the objective function, but also by the environment, ensuring an adaptation to changes in the map.

The GA use a chromosome modeling based on vectors of movements. Its evaluation function uses a heuristic based on Best-First Search to evaluate the best individual. Some restrictions have been adopted to avoid the generation of invalid individuals, for better use of the GA. These restrictions are used in any environment. The individual chosen as the fittest by GA, which is a sub-valid path, is projected into the environment and its nodes are stored on the closed list of the Best-First without the need to expand them. Thus we obtain a reduction in the number of checked nodes.

As the main processing of PPGA achieve gains at the presented pattern environments, in order to guarantee the solution it was necessary to develop a module for identifying when a maze has this feature. In those cases, the GA module is not used.

In the following subsections the procedures for this case will be detailed.

3.1 Chromosome representation

As one of the goals of the algorithm is to detect obstacle patterns in the map, the proposed Genetic Algorithm uses a model based on four movement vectors, two horizontal and two vertical. This model represents a sub-path, which can consist of up to four straight lines, two for each direction, which guarantees the boundary of obstacles. As the maze gets narrow, the distance of the vectors (lines length) tends to decrease, and vice-versa. If the obstacles can be overcome by contouring them, these vectors tend to find this pattern. Each of these four vectors has a distance and a direction, as expressed in Tab 1:

Type	Interval	Description
Float	0.1 to 1	Distance of Movement 1 (DM1)
Float	0.1 to 1	Distance of Movement 2 (DM2)
Float	0.1 to 1	Distance of Movement 3 (DM3)
Float	0.1 to 1	Distance of Movement 4 (DM4)
Integer	-1 to 1	Movement Direction 1 (M1)
Integer	-1 to 1	Movement Direction 2 (M2)
Integer	-1 to 1	Movement Direction 3 (M3)
Integer	-1 to 1	Movement Direction 4 (M4)

Tab. 1 – Chromosome encoding

The interval where the movement distances may vary represent a percentage of the width (in the case of horizontal vectors) and height (in the case of vertical vectors) of the map.

The directions are given by:

- M1 and M3: 0 no movement, -1 down (subtracts the y-axis) and +1 up (increase in y-axis).
- M2 and M4: 0 no movement, -1 to the left (subtracts the x-axis) and +1 to the right (increase in x-axis).

3.2 Fitness Function

The objective function (FA) is used to check each candidate path. It takes into consideration the following terms:

- The Manhattan Distance heuristic (MH), which has the largest weight in the formula. Its equation is given by MH = | x1-x2 | + | y1-y2 |. The higher the value of MH the worst the situation, since it means that the agent is far from the final target;
- Total Movement (MT), calculated by the sum of half of the distance moved. The weight of MH is smaller than the one of MH to serve as a tie-breaker, especially in the elimination of cycling paths.

This objective function must be maximized, meaning that the higher the value the better the candidate. To ensure this feature, we adopted a roof value (upper limit) given by:

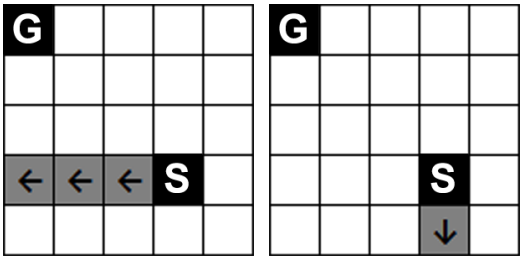
Roof = (Width +Height)* 3

This value represents the sum of the maximum of four movement vectors plus the maximum value of the Manhattan distance. If a path is not valid, the convergence ensures that the result is the lowest possible (in this case, zero). The objective function is given by:

FA = (Roof - MH - MT * 0,1)

3.3 Movement Instances

In order to show the types of changes that may happen, we will consider, with no loss of generality, a 5x5 dimension map. Also let "S" be the initial point of the agent and "G" be the final point (exit). The path under consideration will be represented by arrows in the map. The following situations may be represented in our chromosomes (Fig. 4):



(a) (b)

Fig. 4 – Straight movement.

Fig. 4 shows a visual representation of the chromosomes exposed in Tab. 2.

DM1	DM2	DM3	DM4	M1	M2	M3	M4
0,2	0,6	0,4	0.1	0	-1	0	0
0,1	0,9	0,7	0.1	-1	0	0	0

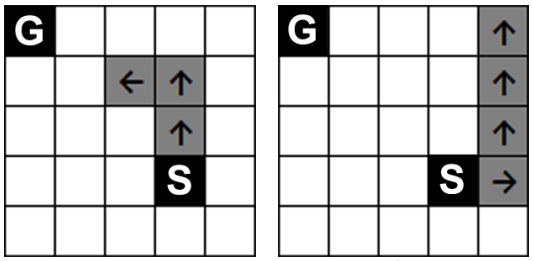
Tab. 2 – Values of straight movement

Tab. 2, line 1, presents an individual in which only the M2 field has a valid value. Thus it represents a single movement to the left on the map (due to the negative value). The distance traveled is a fraction of the map size. As DM2 (relative to M2) has value 0.6 and the map has width 5, the total distance traveled in this movement is given by 0.6 x 5 = 3 squares, as shown in Fig. 4 (a) . The chromosome exposed in Tab. 2 (b) , line 2, also shows a movement with a single direction, presented in Fig. 4 (b). The Manhattan distance value of both chromosomes, i.e. the number of squares between the resulting point of the movement and the maze exit, are 3 and 7, respectively. The objective functions of both chromosomes are shown:

FA(a) = (30 – 3 – 3 * 0,1) * 1 => 26,7
FA(b) = (30 – 7 – 1 * 0,1) * 1 => 22,9

Since the first chromosome of Tab. 2 has a higher FA it is considered better than the second one.

In the next set of examples we will evaluate a movement with a single deviation (Fig. 5) represented by the chromosomes of Tab. 3:



(a) (b)

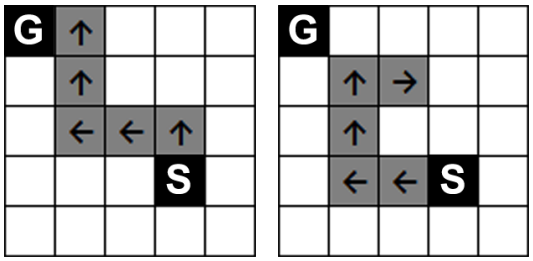
Fig. 5 – Movement with deviation.

DM1	DM2	DM3	DM4	M1	M2	M3	M4
0,4	0,2	0,5	0.9	1	-1	0	0
0,6	0,2	0,8	0.1	0	1	1	0

Tab. 3 – Values of deviation movement.

The first chromosome has two nonzero directions: M1 vertically and M2 horizontally. Therefore we will have two line segments forming a path with deviation. A similar situation occurs in the second chromosome. Both chromosomes are graphically exposed in Fig. 5.

Tab. 4 presents chromosomes with two deviations, shown in Fig. 6:



(a) (b)

Fig. 6 – Movement with two deviations.

DM1	DM2	DM3	DM4	M1	M2	M3	M4
0,4	0,2	0,5	0.9	1	-1	1	0
0,6	0,2	0,8	0.1	0	-1	1	1

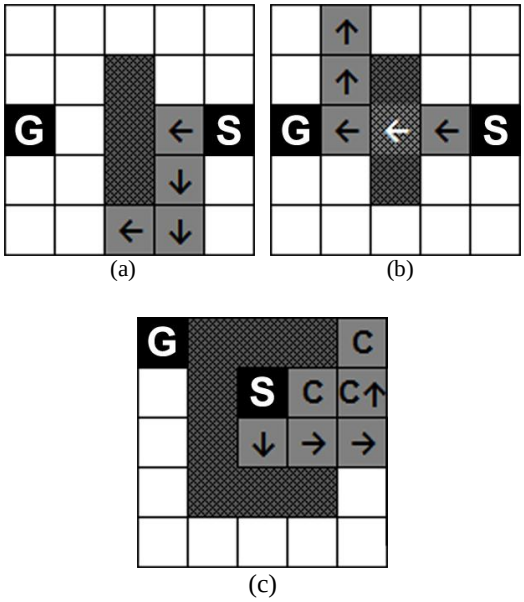
Tab. 4 – Values of the two deviations movement

3.4 Restrictions, Validations and Adaptions

In order to guarantee better results for the sub-paths we defined some fixed and dynamic restrictions. The first restrictions deny the generation of sub-paths without movements, i.e., with all directions from M1 through M4 with null values. These also avoid the generation of paths where their segments are superimposed, which occurs when M2 is null and M1 and M3 have opposite values and when M3 is null and M2 and M4 have opposite values.

In the second case, related to the dynamic restrictions, there is dependence about the output of the map environment and it was defined that there will not be a generation of a sub-path at the direction of the last collision. For instance, if there is an obstacle at the top neighbor of the last visited point, there will not be a generation of a sub-path for the upper way.

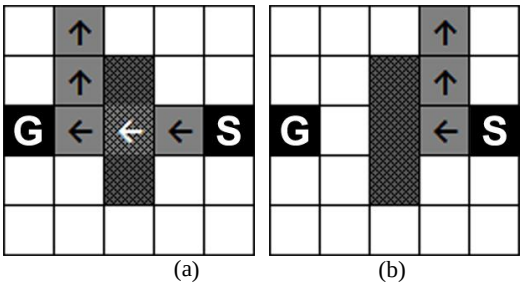
After the chromosomes evaluation, ranking and ordering, each one is tested until the best valid result is executed at the environment (in the case that there is a valid one). This validation consists of verifying each node of the established path by the individual and in the case of an inexistence of a collision, it is positive validated. In case of a collision, the path is adapted and in case of the last node belongs to the closed list or has the F value greater than the current one, this sequence will be considered invalid and removed from the population, as illustrated in Fig. 7.



(a) (b) (c)

Fig. 7. Validation of the paths: (a) is a valid sub-path; (b) is a sub-path to be adapted; (c) is invalid due its last node is at the closed list

The adaptation process was implemented in order to take individuals from a population, decreasing the incidence of cases where any valid individual is found. After this process, the individual will receive a new fitness value and may be chosen to be used in the field test. In this case the test will not be realized for this individual since from that point all adapted individuals are valid.



(a) (b)

Fig. 8. Adaptation of the sub-path

The situation shown in Fig. 8 - (a) represents a sub-path that passes through obstacles. Its codification is listed in Tab. 5.

DM1	DM2	DM3	DM4	M1	M2	M3	M4
0,4	0,6	0,4	0.9	0	-1	1	0

Tab. 5 – Codification of the Fig. 8 – (a) Sub-Path

In Fig. 8 - (b) the adapted sub-path is replaced by this codification (Tab. 4).

DM1	DM2	DM3	DM4	M1	M2	M3	M4
0,4	0,2	0,4	0.9	0	-1	1	0

Tab. 5 – Codification of the Fig. 8 – (b) Sub-Path

3.5 Identification of Environment without Pattern

To avoid an excessive usage of processing in maps that do not have patterns of obstacles, an adopted measurement was the destruction of the GA module once its event is identified.

We defined two metrics for evaluation: the percentage of the distance traveled (D), calculated by the actual distance to the destination ($MD_{current}$) divided by the total distance from the start to the destination (MD_{total}) using the heuristic calculation of Manhattan, as can be seen in Equation 1, and the success rate (S), calculated by dividing the total number of times the reused "sub-path" of the previous generation (C_{first}) divided by the total number of times it failed (U_{first}), shown in Equation 2.

In order to identify if the map is patterned (result of the equation is Pattern=1) or not (Pattern=0), the limits of the success rate and the percentage of the distance have been set through a predetermined calibration, being respectively 0.4 and 0.5 as stated in Equation 3.

In case that the pre-determined percentage of the distance between the start and the end points and the success rate is higher than the value stated previously, the GA will continue to be used. If this rate is a low value, the module GA should be disabled, then the algorithm considers that there is not a pattern of obstacles in the region in which the search occurs.

$$D = \frac{MD_{current}}{MD_{total}} \quad \text{Equation - 1}$$

$$S = \frac{\sum C_{first}}{\sum U_{first}} \quad \text{Equation - 2}$$

$$Pattern = \begin{cases} 0, & D < 0{,}5 \ \ and \ \ S < 0{,}4 \\ 1 \end{cases} \quad \text{Equation - 3}$$

3.6 PPGA Architecture

As the PPGA was developed based at a modification of the BFS, we introduced a module that uses the Genetic Algorithm in order to calculate the sub-paths that are used along the processing. Therefore, each time that the module is called, the generated sub-paths will inherit information of the past generations, making possible a considerable gain of performance. The event for calling the GA module happens when none of the values of the open node list of the BFS is better than its actual node. Fig. 9 shows the proposed architecture.

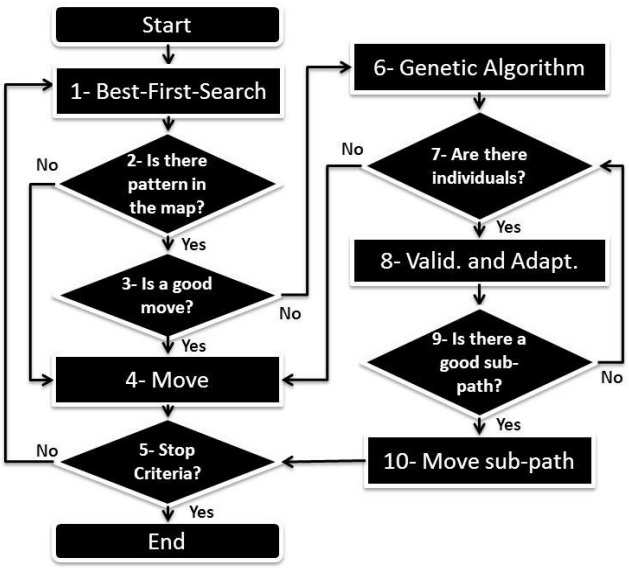


Fig. 9. Patterned based Pathfinding with Genetic Algorithm architecture

1. Best-First-Search – For each BFS iteration list, the actual node, previously added to the closed list, is expanded and the open list node with the lowest value of F is selected (greedy algorithm)
2. Is there a pattern? – In the genetic algorithm module there is a counting variable that defines the proportion of times that the sub-paths were used by the tentatives of usage. If this proportion is low, we define that there is no pattern for the environment and from there on the AG module will not be used anymore.
3. Is a good move? – This verify if the selected node corresponds to a good movement, i.e., if its value is lower than the actual node.
4. Move – adds the selected node to the closed list.
5. Stop Criteria – If the actual node consist on the destiny node, the algorithm ends, other way it continues for the next iteration.
6. Genetic Algorithm – In this module it is obtained only one generation, following the determined restrictions by the following steps:
   a. Generate a initial population of 4 individuals;
   b. Generate the first off-spring by a crossover of the 4 first individuals. The established rate was 50%;
   c. The second off-spring is generated by the mutation of the initial population. The established rate for this stage was 25%;
   d. It is calculated the fitness value for the 12 individuals. The element that was used in the last generation receives the largest value;
7. Are there individuals? – It is verified if there are still chromosomes to be evaluated at the population. In case none of the individuals are valid, the algorithm proceeds maintaining the current node still selected.
8. Validation and adaptation – This stage helps to guarantee the generation and adaptation of good individuals coming from the AG.
9. Is there a good sub-path? – The generated individuals go by this test, one at each iteration. If it is not a good element, this will be taken off from the list of population.
10. Move sub-path – The element will be used as a sub-path composed by the nodes that will be added to the closed list. The actual node will become the last node of the sub-path. This element will be inserted to the next generation of the GA.

Fig. 10 illustrates part of this architecture. In (a) the BFS does not have a neighborhood that is better than its current state, so it will be called the GA module. In (b) the valid sub-path is projected. In (c) and (d) the BFS ends the search.

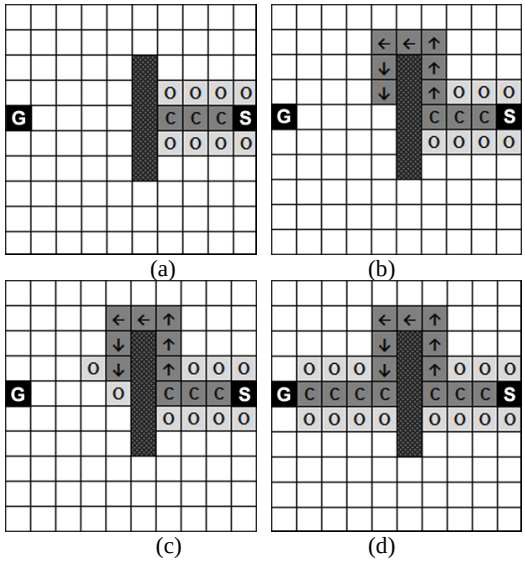


Fig. 10 – Part of the stages of execution of the PPGA among a maze after a training step.

4. Experiments

Since one of the goals of this work is to prepare a test bed for optimization algorithms for the pathfinding problem, suitable programming language and development environment were chosen. Another factor considered in the choice of language and development environment was the possibility of exporting the developed systems to mobile devices such as tablets and mobile phones where the optimized use of the resources is highly recommended.

4.1 Technology

The development technologies used were Unity 3D [UNITY TECHNOLOGIES] through the C # language.

Unity3D 3.x is a game development engine which has the main components needed for designing games developing rapidly, such as the physics engine, collision systems, sound system and a high level programming language based on C, among others.

Unity 3D has an interface for programming in Mono. It incorporates key .NET components, including a compiler for the C# programming language and a complete suite of class libraries.

The direct assignment of a list of values to arrays in C # allowed an easy viewing of maps used in the experiments. Thus the generation of a tile set, which is the display of graphics (or sprites) defined in a two-dimensional array, can be made as shown in Fig. 11.

The modularization of the code and Object-Oriented paradigm allows dynamic instantiation of both the environment and the agents in Unity.

```
string[] map = new string[20]
                        {
                "wwwwwwwwwwwwwwwwwwww",
                "w.S................w",
                "w..................w",
                "w.www.www.www2www.ww",
                "w..................w",
                "www.www.www.www.wwww",
                "w..................w",
                "w.www.www.www.www.ww",
                "w..................w",
                "wwww.www.www.www.www",
                "w..................w",
                "w.www.www.www.www.ww",
                "w..................w",
                "wwww.www.www.www.www",
                "w.1................w",
                "w.www.www.www.www.ww",
                "w..................w",
                "wwww.www.www.www.www",
                "w..............3...w",
                 "wwwwwwwwwwwwwwwwwwww"
                        };
```

Fig. 11 – Dynamic tileset from a map applied to an array in C #. "W" represents Walls (obstacles), "S" Start, "1, 2 ..." the sequence of targets to be achieved.

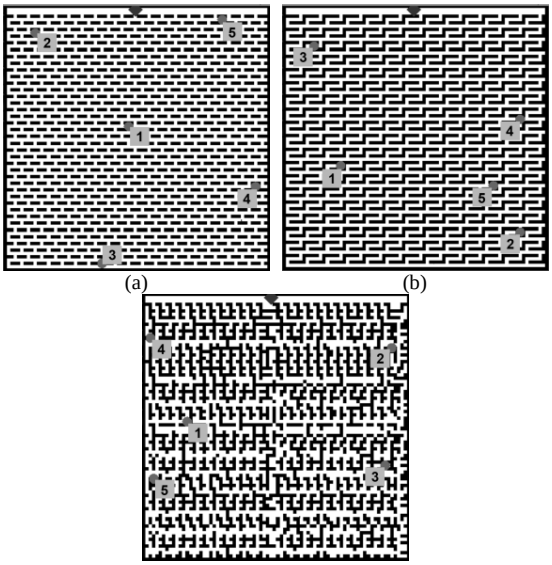
We executed this algorithm in a Tablet with Android Operational System 2.1 éclair, as exposed in Fig. 12. We note that in (a) a pattern was found at the beginning. As predicted, in (b) the algorithm spent some time to activate the module GA since the value of F for all nodes of the open list is smaller than the current node in most of the time.

4.2 Tests

We conducted three tests to compare the costs of memory between A*, BFS and PPGA algorithms. Each test was made in a different map, with constant dimensions of 80 by 80 tiles. In all, to be deterministic (and therefore not generate solutions with different memory costs in an environment without variation), the A* and BFS was performed only once. In contrast the average was calculated for 20 runs of PPGA in each test case. In order to characterize the principle of a dynamic environment, we defined five destinations in each map. Once a target was reached, the open and closed lists of all algorithms were reseted, simulating changes in the environment.

The maps types were:

- With patterns (Fig. 13 (a)) - Aiming to assess the advantages of the adjustment proposed by PPGA;
- Mixed Map (Fig. 13 (b)) - In order to evaluate a balanced case.
- Without patterns (Fig. 13 (c)) - to evaluate the worst case of PPGA.



(a) (b)

(c)

Fig. 13 – Map patterns

4.3 Results and Analysis

The results show that the agents managed to adapt themselves to the surrounding environments even in the presence of obstacles.

One objective of the tests was to check if the agents are able to find a path to the maze exit. In 100% of the cases the agents achieved the goal.

The result of the experiments can be checked in Tab. 7, in which the values represent the number of access nodes.

	Map		
Algorithm	**With Patterns**	**Mixed**	**Without Patterns**
A-Star	16016	27692	18468
Best-First	2048	3088	5796
PPGA	1725	3142	6252

Tab. 7 - Tests Results

In Fig. 14 it is possible to check the comparative projection of BFS and PPGA. We do not show the A* in this comparison because its high values would undermine the view of the difference.

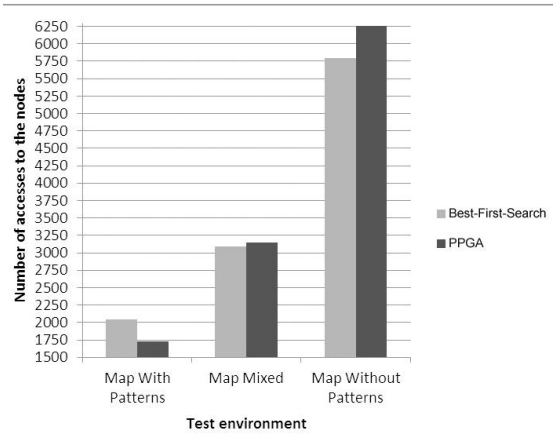


Fig. 14 – Map Patterns Comparison.

The purpose of this study is optimize the search for paths between two points through the reduction of nodes accessed or expanded, but without the guarantee of the shortest path.

The main result of this research was to optimize 16% of the expense of processing the PPGA compared to the Best-First Search in maps with patterns.

In the other two tests the results were balanced (losing by 2% on the mixed map) or slightly worse (losing by 8% on the map without patterns).

In the test without patterns, the proportional loss of processing is due to the fact that in this case the PPGA algorithm tends to generate sub-paths more frequently, because there is no genetic advantage of the previous chromosomes which leads to an overspending of memory. This result could be worse if wasn't the presence of the pattern identification module inserted into the algorithm.

The results were in accordance with the expectation, since the PPGA has a better performance on maps with patterns of obstacles due to the adjustment provided by the Genetic Algorithm.

5. Conclusion and Future Work

This work defines a new method for searching the path between two points in 2D maps. The PPGA uses the classic BFS with a Genetic Algorithm in order to find good solutions in a short period of time.

The proposed GA used "lookahead" based modeling to the chromosome that can be adapted to any kind of map that contains obstacle patterns. Also the "lookahead" pathology is avoided by the iterative nature of the algorithm.

For this work, dynamic environment is defined as one that can suffer alteration in the position of the obstacles or target at any time. For this, our experiments were conducted on large maps with more than one goal. The knowledge gained in the initial executions ensures the adaptation of the agent to new situations with lower latency learning, in practical situations.

Experiments proved that the PPGA showed slightly better than the Best-First Search being compared on maps of three variations of patterns of obstacles. When there are patterns in this distribution, the proposed algorithm achieved its best performance with an average gain of 16% when compared with the BFS. Therefore we consider this architecture promising.

As future work, we will seek improvements in this architecture to optimize the search process, such as varying the number of segments of lines contained in the chromosome and improving the function of identifying patterns in the map. Experiments in maps with other features will be conducted.

Acknowledgments

The authors wish to thank the Programa de Educação Tutorial of the Ministério da Educação (Brazil) for the opportunity to participate on this project. Also thank the Instituto Federal de Educação Ciência e Tecnologia do Sudeste de Minas Gerais – Campus Rio Pomba, for their support with the implementation of the Laborátorio de Multimídia interativa (LAMIF).

REFERENCES

DECHTER, R., AND PEARL, J., "Generalized Best-First Search Strategies and the Optimality of A*." Journal of the Association for Computing Machinery , Vol. 32, No. 3, July 1985, pp. 505-536.

UNITY TECHNOLOGIES: Unity 3D User Manual. At www.unity3d.com/support/documentation/.

LEIGH, R., LOUIS, S. J. AND MILES, C. "Using a Genetic Algorithm to Explore A*-like Pathfinding Algorithms". In: IEEE Congress on Computational Intelligence and Games. CIG – 2007.

DEMYENAND, D. AND BURO, M. "Efficient Triangulation-Based Pathfinding". In: AAAI'06 Proceedings of the 21st national Conference on Artificial intelligence. 2006.

BURCHARDT, H. AND SALOMON, R. "Implementation of Path Planning using Genetic Algorithms on Mobile

Robots". IEEE Congress on Evolutionary Computation. CEC 2006.

BULITKO, V., LUSTREK, M. "Lookahead pathology in real-time path-finding". In Proceedings of the National Conference on Artificial Intelligence. AAAI 2006.

HART, P.E., NILLSON, N.J. AND RAPHAEL, B. "A formal basis for the heuristic determination of minimum cost paths". IEEE Transactions on Systems Science and Cybernetics, 1968.

BJORNSSON, Y., ENZENBERGER, M., HOLTE, R.C. AND
SCHAEFFER, J. "Fringe search: Beating A* at pathfinding on gamemaps". IEEE Computational Intelligence in Games, 2005.

STODOLA, P. AND MAZAL, J. "Optimal location and motion of autonomous unmanned ground vehicles". In World Scientific and Engineering Academy and Society. WSEAS 2010.

MITCHELL, M.; "An introduction to genetic algorithms". The MIT Press, United States; 1996.

KORF, R. Depth-first iterativedeepening: An optimal admissible tree search. Artificial Intelligence, (1985).

RUSSELL, S.J. AND NORVIG, P. Artificial Intelligence: A Modern Approach , Second Edition.

MACHADO, A. F. V. ; CLUA, E. W. ; GONÇALVES, R.; VALE, H. ;SANTOS, U. O. ; NEVES, T. ; OCHI, L. S. "Real Time Pathfinding with Genetic Algorithm". In: SBGames, 2011, Salvador, BA. Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames), 2011.

RIOS, L. H. O. ; CHAIMOWICZ, L. . PNBA*: A Parallel Bidirectional Heuristic Search Algorithm. In: Encontro Nacional de Inteligência Artificial (ENIA 2011), 2011, Natal. Anais do XXXI Congresso da Sociedade Brasileira de Computação, 2011.

RIOS, L. H. O.; CHAIMOWICZ, L. . A Survey and Classification of A* based Best-First Heuristic Search Algorithms. In: Brazilian Symposium on Artificial Intelligence - SBIA, 2010, São Bernardo do Campo. Advances in Artficial Intelligence - SBIA 2010 (LNAI), 2010. v. 6404. p. 253-262.