C. de Valdes Salas, 11
Tel. 98 510 27 96
eii@uniovi.es
33007 Oviedo

- HEURISTIC SEARCH -

Guillermo Facundo Colunga. uo236856@uniovi.es.

Abstract. Find an objective state in a space of possible states is a common problem on computer science. To solve this problem exists many approaches. This paper provides the theory of some of this options, the experimental analisis of them and finally the conclusions taken after the carefully examination of those results. The primary target of this paper is the young computer science student, however, anyone interested is able to comprehend and learn from it.

Keywords: heuristic search, blind search algorithms, intelligent search algorithms, genetic algorithms, eight puzzle problem, tsp.

I. Introduction

On computer science a common problem is to find an objective state in the space of possible states. A state could be a leave in a tree and the space of possible states would be the whole tree. If the algorithm to travel thought the tree to the leave has no information we will talk about blind search algorithm, is the algorithm has some information that helps it to improve its search then we will talk about informed search algorithms.

For the experiments of this paper we will use the project aima-java that has implementations for informed and un-informed algorithms. Both kinds of algorithms will be described and evaluated using the 8-puzzle problem.

First we will evaluate the un-informed algorithms with Breath-First search, Uniform Cost search, Depth-First search and the iterative version of the Depth-First search, on instances of costs 5, 10, 15, 20, 25 and 30. And for two different spaces, trees and graphs.

For the intelligent algorithms we will explore the results of their execution with different heuristics.

Finally we will describe and explain the results obtained in the experiments in terms of execution time, cost of the found path and expanded nodes.

II. Blind search algorithms

The blind search search algorithms or un-informed ones have no information about the state of the problem and therefore they don't follow any heuristic strategy. Instead they just know if the a state is valid or not to guide the search process.

As un-informed algorithms we will explore: Breath-First search, Depth-First and Uniform Cost search.

II.I. Breadth first search

BFS, as described at [1], is an algorithm for searching a tree or a graph. It starts at the root of the tree (or ran-

dom node on a graph) and explores all the neighbor nodes at the current depth prior movie to the next level. It only stops when finds a solution or explores the whole structure.

II.II. Depth first search

DFS, according to [2], is a search algorithm that as BFS try to find a solution in a tree or a graph. DFS also starts its search at the root of the tree (or random node in a graph) but, then, instead of going level per level, it traverses as depth as possible in the currency branch before moving to the next one. There are many versions of this algorithm like the limited version, that fixes a maximum depth, or the iterative one.

II.III. Uniform cost search

As BFS and DFS, USC it is also a search algorithm for structures like trees or graphs, it also starts from the root node or a random one if it is a graph, but this algorithm works by expanding always the less expensive cost node. In that way always returns the optimal solution, if possible. For more information look [3].

III. Intelligent search algorithms

As defined at [4] Intelligent search algorithms or informed ones use some problem specific knowledge beyond the definition of the problem itself. So, they follow a heuristic strategy. That helps the algorithm to find a more efficient solution that the algorithms deceived in point II (BSA).

III.I A algorithm*

The A* algorithm is an informed algorithm that can be used to find the minimum paths in a set. The ventral idea of the algorithm is to guide best-fist search both by

a) the estimate to the goal as given by the heuristic function h and,

b) the cost of the of the path developed so far.

Let n be a node, $g(n)$ the cost of moving from the initial node to n along the current path, and $h(n)$ the estimated cost of reaching a goal node from n. Define $f(n)$ as follows: $f(n) = g(n) + h(n)$.

This is the estimated cost of the cheapest path through n leading from the initial node to a goal node. A* is the best-first search algorithm that always expands a node n such that $f(n)$ is minimal.

III.II Static ponderation PEA algorithm*

This is a variant of the A* not admissible algorithm (ε-admissible), that can find a solution with a limited error (ε). This algorithm finds a solution lower or equal cost than $(1 + \varepsilon)C*$ if the heuristic is admissible.

And taking in to account the previous formula and the A* definition the evaluation of this algorithm will be: $f(n) = g(n) + (1 + \varepsilon)h(n)$ where $\varepsilon >= 0$.

III.III Greedy best first search

Is a search algorithm which explores a graph by expanding the most promising node chosen according to a specific rule, according to [5].

For more information about Intelligent search algorithms please take a look to [6].

IV. Genetic algorithms

A genetic algorithms (GA) is a metaheuristic, as described at [9], inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Those algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on operators such as mutation, crossover and selection.

In addition toe the above operators those algorithms still can use other heuristics to take the calculation faster and more robust. For example the specialization heuristic penalizes crossover between candidate solutions that are too similar; this forces population diversity

and helps prevent premature convergence to a less optimal solution.

— For the propose of this paper we will use a varian of the GA called GA with elitism, described at [9].

V. Eight puzzle problem

The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. As explained at [7]. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Such a puzzle is illustrated in following *figure 1*.

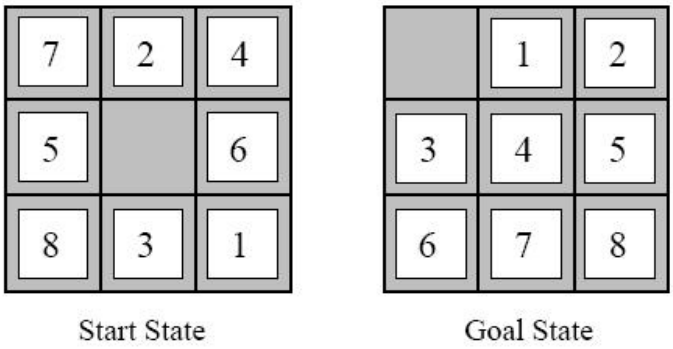

Fig. 1. Example of the initial and the goal states of the eight puzzle.

V.I Problem statement

The aim of the problem is to find an ideal solution in terms of movement to go from the initial state to the desire one, in this case the ordered one. That can be done by brute force by in the aim of the problem is also find the ideal solution in the minimum time. For that we will use different algorithms and heuristics.

V.II Modeling the search state space

In this problem we have a square board (3x3), where we have 9 pieces included the blank space in each of the 9 possible positions. Taking in to the account the previous statement we can state that the number os possible states is 9! (362.800 states). However the number of possible permutations could reach 9!/2 (181.440).

In the figure 1 we can see that the initial state is the starting set with any provision of the pieces and the final state is the certain set that we want to reach. In this case first the blank space and then all the set of pieces sorted from lower to higher. The transition function describes the changes that produce the movement of the piece in any possible direction [8].

V.III Heuristics

We will test this problem with four different heuristics, in order to explore the different possibilities. The heuristics to explore will be:

a) $h_0(n)$: Always returns 0.

b) $h_1(n)$: Number of unordered pieces.

c) $h_2(n)$: Sum of the orthogonal distances of each piece to the goal one.

d) $h_3(n)$: 2* the number of pieces to distance 2.

e) $h_4(n)$: A reasonable estimation, although it is not admissible.

VI. Experimental analisis

So, as described in previous sections, we have different algorithms, data structures and heuristics to solve the eight-puzzle problem. Those will be the different situations under test.

a) Blind search algorithms in graphs.

b) Blind search algorithms in trees.

c) A* algorithm with h_0, h_1 and h_2 on graphs. Implement and evaluate h_{1-2} on graphs. Evaluate h_2 on trees.

d) Evaluate the A* algorithm results with `Graph-Search`, `GraphSearchRectifyExpanded` and `GraphSearchReinsertExpanded`. For h_2 and h_3.

e) Experimentation with PEA* algorithms by using different ε values considering h_2.

f) Evaluate some not admissible algorithms.

Finally we will experiment and evaluate the results of test the TSP problem, as described at [10], with the A* algorithm and the above described genetic algorithms with the `FlexibleGeneticAlgorithm` implementation and the two points cross operator OX implemented at Two`PointsCrossoverNoRepetitionRF`.

VI.I Example's bank

For the domain of this experimental analisis we will use the following instances of the problem provided at the aima-java projects by the Intelligent Systems course teachers at the School of Computer Science of Oviedo.

Cost	Instances		
5	{1, 2, 3, 0, 4, 5, 8, 7, 6}	{1, 0, 2, 8, 6, 3, 7, 5, 4}	{2, 8, 3, 1, 6, 4, 7, 0, 5}
10	{1, 3, 4, 6, 7, 2, 0, 8, 5}	{3, 2, 4, 1, 0, 5, 8, 7, 6}	{8, 1, 0, 2, 5, 3, 7, 4, 6}
15	{6, 3, 5, 2, 1, 0, 8, 4, 7}	{6, 0, 3, 8, 1, 5, 4, 2, 7}	{7, 8, 3, 1, 5, 0, 4, 2, 6}
20	{1, 4, 7, 6, 8, 5, 0, 3, 2}	{6, 4, 0, 2, 8, 1, 7, 3, 5}	{4, 1, 3, 7, 2, 8, 5, 6, 0}
25	{3, 4, 8, 5, 7, 1, 6, 0, 2}	{4, 5, 3, 7, 6, 2, 8, 0, 1}	{2, 7, 8, 5, 4, 0, 3, 1, 6}
30	{5, 4, 7, 6, 0, 3, 8, 2, 1}	{3, 8, 7, 4, 0, 6, 5, 2, 1}	{5, 6, 3, 4, 0, 2, 7, 8, 1}

VI.II Experimental results

In this section we explain the results of each one of the different situations under test described in point VI.

— *During this experimental analisis every figure represents the mean of at least 3 different executions.*

Blind search algorithms

Depth first search. When we evaluate this algorithm over trees with a cost of 5 we obtain a memory limit exception because it goes through an infinite branch and the execution doesn't finish. However for costs lover than 2 it fins a solution expanding very few nodes.

For graphs this algorithm always finds a solution[1]. In terms of performance when we increase the cost of the solution, the time and the number of expanded nodes also are increased.

Depth first limited search. It is important to remember that this version of the depth first search depends on the depth limit. If the limit is set to a low value the algorithm will be able to find an small cost solution really fast, but if the limit is set to a high value the solution found may be not be the smallest one. For example with a limit 5.0 and a cost 5.0 the solution has a cost of 5.0. But if we change the limit to 10.0 the cost of the solution is 9.0, which is not the smallest one.

Iterative depth search. Evaluating this algorithm we notice that the number of expanded nodes, as the time taken to provide a solution increase very high with small variations of the cost. That is that for costs greater than 15 we where unable to obtain a solution.

Breath first search. Over trees we find that the algorithm finds a minimum cost solution for problems with cost up to 15; when the costs are greater the execution of the problem does not end as it goes through an infinite branch and rises an exception.

For graphs and for all the instances described above, it always find a minimum cost solution. But whenever we increase the cost of the solution the time reasonably increases but the number of expanded nodes increases exponentially.

Uniform search. When we evaluate this algorithm over the described instances and tree search spaces we found that the algorithm is not able to provide a solution for instances with cost grater than 15 in a reasonably time.

In graph search spaces it is able to solve all costs instances with minimum cost solution, whenever the cost increases the time does also and the number of nodes expanded increases exponentially.

Intelligent search algorithms

A* algorithm. When we execute the A* algorithm we obtain different result for each convention of

[1] For the costs of the instances described.

different instances and heuristics. As shown in the results, the heuristic h_2 is always the best choice to evaluate this algorithm, it always finds the less cost solution with the lower time and lower # of expanded nodes.The other heuristics evaluated expand more nodes, have a worst performance and some of then even rectify.

Table 1. #Nodes expanded by A* with heuristics h_0, h_1, h_2 and h_{1-2} for instances of cost 5 over graph search spaces.

	heuristic			
	h_0	h_1	h_2	h_{1-2}
# nodes expanded	58	6	5	5
time taken	18	12	13	1

In the previous table *(Table 1)* we can see that the h2 and h1-2 algorithms are the ones that expanded less number of nodes but as more os less all of them have a similar performance we cannot state that none of them is the fastest, for that we represent *Table 2*, where we follow the same procedure but with an instance of cost 30.

Table 2. #Nodes expanded by A* with heuristics h_0, h_1, h_2 and h_{1-2} for instances of cost 30 over graph search spaces.

	heuristic			
	h_0	h_1	h_2	h_{1-2}
# nodes expanded	181365	102851	14027	43298
time taken	723	1037	187	363

Now, in *Table 2* we can for sure state that the heuristic h_2 is the one that, after evaluating the results on different instances of the problem and other heuristics, has the best performance in number of nodes expanded, the time taken to find the solution and the quality of the solution as it is always the less cost solution.

That is reflected in the *figure 1* where we can see that between $h_{0,\ 1\ \text{and}\ 2}$ h_2 is the heuristic with less number of nodes expanded by far.

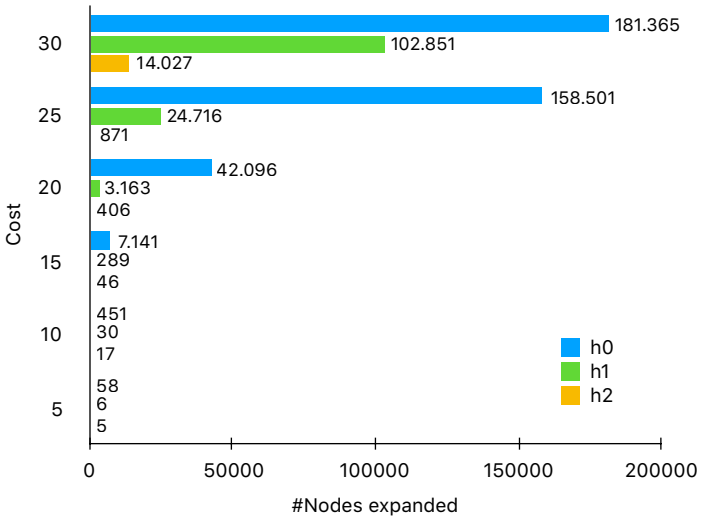

Fig. 2. #Nodes expanded by A* for heuristics $h_{0,1,2}$ and instances of different costs over graph search spaces.

Also, if we compare $h_{1,\ 2\ \text{and}\ 1\text{-}2,}$ h_2 still is the one with the best performance among them.

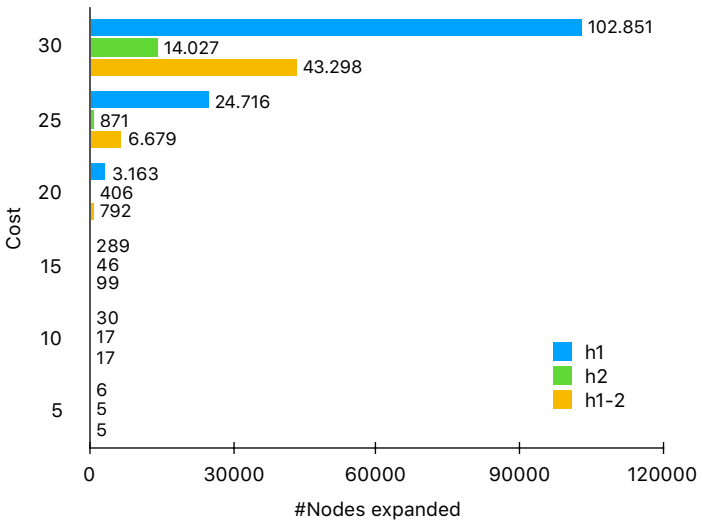

Fig. 3. #Nodes expanded by A* for heuristics $h_{1,2,1\text{-}2}$ and instances of different costs over graph search spaces.

The above results, as said, are for graph search spaces, now let's compare them with tree search spaces.

Table 3. #Nodes expanded by A* with heuristics h_2 for instances of cost 5 and 30 over tree search spaces.

	h_2 Cost 5	h_2 Cost 30
# nodes expanded	5	11221067
time taken	13	111253

As we can see by comparing *table 2* and *table 3* the tree search spaces are not the best ones for big problems. For a cost of 5 we have the same performance as in a graph search space, but for a cost of 30 we get 11.207.040 more nodes expanded and the algorithm needs much time to find a solution.

Below are some more results of the different conventions of costs, heuristics and implementations.

Table 4. #Nodes expanded by A* with heuristics h_2 and h_3 for instances of cost 5 and 10 over graph search spaces.

	heuristic with cost 5		heuristic with cost 10	
	h_2	h_3	h_2	h_3
# nodes expanded	5	12	17	74
time taken	13	13	16	21

Table 5. #Nodes expanded by A* `GraphSearchRectify` with heuristics h_2 and $h_3$for instances of cost 5 and 10 over graph search spaces.

	heuristic with cost 5		heuristic with cost 10	
	h_2	h_3	h_2	h_3
# nodes expanded	5	12	17	88
time taken	12	20	18	29

Table 6. #Nodes expanded by A* `GraphReinsertedExpanded` with heuristics h_2 and $h_3$for instances of cost 5 and 10 over graph search spaces.

	heuristic with cost 5		heuristic with cost 10	
	h_2	h_3	h_2	h_3
# nodes expanded	5	12	17	88
time taken	18	21	19	26

PEA* algorithm. We will evaluate this algorithm with different instances of different costs and different epsilon values. Then we will try to optimize the epsilon value for one single instance of cost 30.

Table 7. #Nodes expanded by PEA* with heuristic h_2 for instances of cost 5 and different epsilon values.

heuristic	epsilon			
h_2	0,1	0,5	1	1,5
# nodes expanded	5	5	5	5
time taken	1	1	1	1
path cost	5	5	5	5

Table 8. #Nodes expanded by PEA* with heuristic h_2 for instances of cost 30 and different epsilon values.

heuristic	epsilon			
h_2	0,1	0,5	1	1,5
# nodes expanded	9347	3971	5140	4306
time taken	190	111	131	88
path cost	30	30	32	34

From the *tables 8 and 9* we can see that different epsilon values affect the number of nodes expanded, the time taken to find a solution and the quality of the solution. Also from the *table 8* we can see that for instances with cost 30 seems that there's an optimum epsilon value between 0,1 and 1,0. As an academic task lets try to find it.

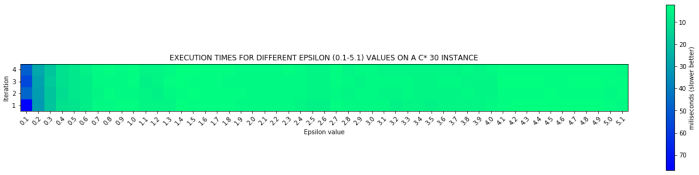

Fig. 4. Response time of PEA* for epsilon values [0.1 - 5.1]. Green color indicates faster response time, blue slower.

From the *figure 3* we can see that the time that PEA* algorithm takes to find a solution starts decreasing at 0.1 and stops reducing at 1.5 (from that point it remains constant).

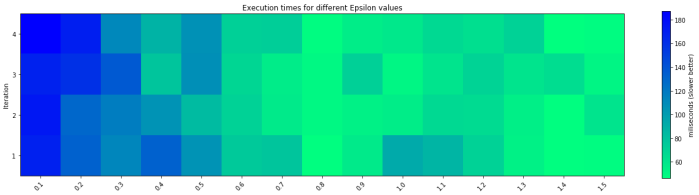

Fig. 5. Response time of PEA* for epsilon values [0.1 - 1.5]. Green color indicates faster response time, blue slower.

At the *figure 4* we can see that response time (r.t.) for PEA* tends to be smaller at a value of 0.8. But we have to take into account the cost of the solutions provided by those r.t.

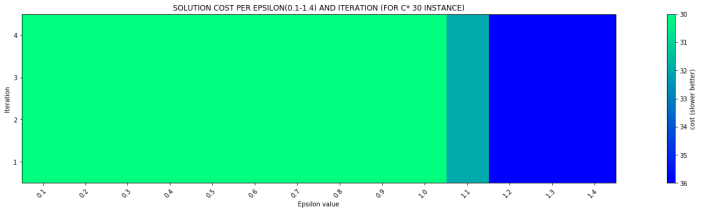

Fig. 6. Cost of the solution provided by PEA* for epsilon values [0.1 - 1.5]. Green color indicates a cost of 30, blue indicates a cost of 36.

With the *figure 5* we found that the solutions computed with an epsilon greater than 1.0 gets a non-optimal solution.

In the *figure 6* we can see the combination fo *figures 4 and 5*; and therefore the result of epsilon optimization for a 30 cost instance. 0.8 is the epsilon value where the cost is optimum and the response tie is the fastest.

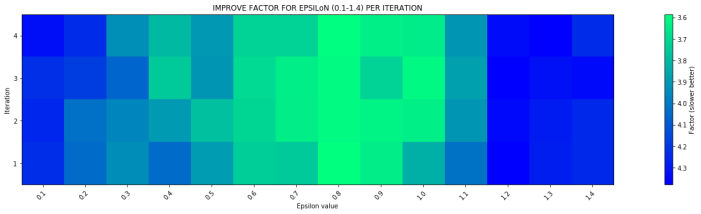

Fig. 7. Combined figures 4 and 5 so that represents the response time of the solution plus its cost. Epsilon values [0.1-1.4]. Green means small time and optimum cost and blue means or slow or non-optimum solution.

Inadmissible algorithms.

Table 9. #Nodes expanded by A* with heuristic h_4 for instances of different cost.

heuristic	costs			
h_4	5	10	15	20
# nodes expanded	5	17	43	258
time taken	19	21	27	20

Table 10. #Nodes expanded by `GreedyFirstSearch` with heuristic h_4 for instances of different cost.

heuristic	costs			
h_4	5	10	15	20
# nodes expanded	5	10	336	169
time taken	24	26	49	33
path cost	5	10	55	42

— *Notice that the above algorithm does not produce optimum solutions for costs greater than 10.*

Genetic algorithms

To study the performance of the genetic algorithms we need something to compare with. That is why we will use the A* algorithm, which always finds an optimum solution. To test both of them, as described, we will use the TSP problem, described at [10].

With **A*** we will use two heuristics, first the heuristic of spanning trees and then with the sum of minimum arcs. And from the following table we can deduce that the spanning trees heuristic obtains better results. This can be produced because the sum of minimum arcs sometimes can give repeated arcs. And to prove that we fount the evidence at the *Table 11*.

Table 11. #Nodes expanded by A* with the spanning trees and sum of minimum arcs heuristics for a 6 and 17 cities problem. With nodes rectification and reinsertion.

# nodes	spanning tree		sum of minimum arcs	
	6 cities	17 cities	6 cities	17 cities
consistent	12	30891	21	411108
rectified	12	30788	21	410830
reinserted	12	30788	22	410830

With **genetic algorithms** we have to take in to account that they are not deterministic, that means that two exactly equal executions will produce a different output, in our case, cost. So, for comparing the results with a deterministic algorithm, as it is A*, we will make 10 executions and take the mean of them. See figure 7 to illus-

trate de dispersion of results over different executions with the same conditions.

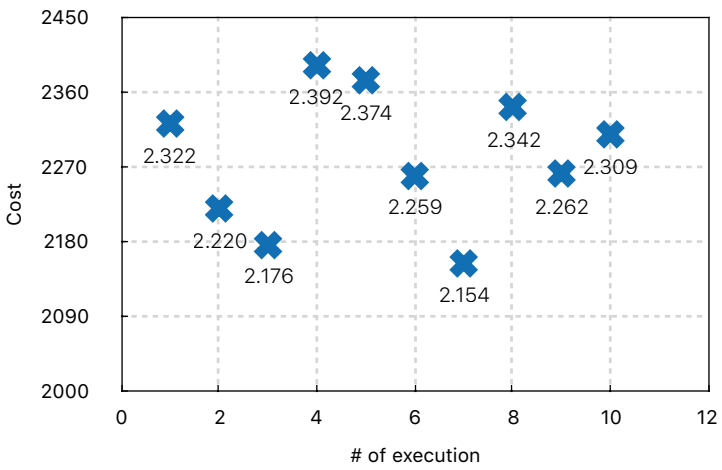

Fig. 8. Costs for different executions over a 17 cities problem in a genetic algorithm.

From the figure 7 we can obtain that the mean is 2.281 meanwhile we know that for this 17 cities problem the optimum cost is 2085. Therefore the genetic algorithm, in this case, does not obtain the best possible solution in terms of cost in none of its executions.

VII. Conclusions

After studying the different search algorithms, comparing them in terms of expanded nodes, time taken to find a solution and the quality of the solution itself, we can state that:

a) When we talk about blind search algorithms, for all of them the best search space is the graph. When they face up trees, as a search space and the cost of the solution is big, we cannot find a solution due to the amount of memory that those algorithms need for those data structures.

b) About the informed search algorithms we can state that obviously they have a better performance than uninformed ones, but depending on the heuristic their result will be completely different. And therefore the heuristic employed can be optimized for a given problem.

c) For the A* algorithm, the heuristic that ensures best results is h_2, since it always finds the best solution in the less time with the less number of nodes expanded.

d) For the PEA* algorithms we can state that they are mot optimistic that the A* versión and that for instances of cost 30 we've fount an optimum value for epsilon of 0,8.

e) For the TSP problem with A* and genetic algorithms we can state that the A* algorithm always find the optimum solution, with best performance with the spanning trees heuristic, meanwhile the genetic one is highly indeterministic an does not find the best solution in none of the executions done during this experiment.

VIII. References

1. Wikipedia. (2018). Breadth-first search. [online] Available at: https://en.wikipedia.org/wiki/Breadth-first_search [Accessed 12 Oct. 2018].

2. Wikipedia. (2018). Depth-first search. [online] Available at: https://en.wikipedia.org/wiki/Depth-first_-search [Accessed 12 Oct. 2018].

3. Agrawal, S. (2018). Artificial Intelligence – Uniform Cost Search(UCS). [online] Algorithmic Thoughts. Available at: https://algorithmicthoughts.wordpress.-com/2012/12/15/artificial-intelligence-uniform-cost-searchucs/ [Accessed 12 Oct. 2018].

4. Endriss, U. (2018). Search Techniques. [online] Universitatea Babeş-Bolyai. Available at: http://www.cs.ubbcluj.ro/~csatol/log_funk/prolog/slides/7-search.pdf [Accessed 12 Oct. 2018].

5. Wikipedia. (2018). Best-first search. [online] Available at: https://en.wikipedia.org/wiki/Best-first_search [Accessed 12 Oct. 2018].

6. University of California, Irvine. (2018). Informed search algorithms. [online] Available at: https://www.ics.uci.edu/~welling/teaching/271fall09/Inf-Search271f09.pdf [Accessed 12 Oct. 2018].

7. GeeksforGeeks. (2018). 8 puzzle problem. [online] Available at: https://www.geeksforgeeks.org/8-

puzzle-problem-using-branch-and-bound/ [Accessed 12 Oct. 2018].

8. Caparrini, F. and Work, W. (2018). Espacios de estados - Fernando Sancho Caparrini. [online] Universidad de Sevilla. Available at: http://www.cs.us.es/%7Efsancho/?e=33 [Accessed 12 Oct. 2018].

9. Wikipedia. (2018). Metaheuristic. [online] Available at: https://en.wikipedia.org/wiki/Metaheuristic [Accessed 13 Oct. 2018].

10. THE UNIVERSITY OF CRETE. (n.d.). The Traveling Salesman Problem. [online] Available at: https://www.csd.uoc.gr/~hy583/papers/ch11.pdf [Accessed 13 Oct. 2018].

Firmado digitalmente por FACUNDO COLUNGA GUILLERMO - 45170929X
Fecha: 2018.10.20 19:30:12 +02'00'