RESOLUTION OF THE TSP PROBLEM WITH SEARCH IN STATE SPACE AND GENETIC ALGORITHMS

Guillermo Facundo Colunga. UO236856@uniovi.es. Sistemas Inteligentes. Grado en Ingeniería Informática. EII. Universidad de Oviedo. Campus de los Catalanes. E-33007. Oviedo

Abstract. This paper contains an experimental study of the two main methods available to solve the TSP problem, search in state spaces and search by means of genetic algorithms. For state spaces search A* and a modification of it, static weighted A* (PEA*), will be used with different heuristics. The genetic algorithm will use a simple codification that will be based in permutations. The aim of the paper is to show and compare the throughput of these two methods over different instances of the TSP problem. And if possible, compare the obtained results.

Keywords: TSP, A, PEA, Genetic Algorithms, Heuristics, Search in State Spaces.

1 Introduction

The search has a big place in AI (the kernel of an Intelligent System is a search algorithm). With state spaces algorithms we have some admissible algorithms (always find the best solution) and others that are only complete (always find a solution but maybe not the best one). The aim to use the complete algorithms is that sometimes problems are so big that is better to get a reasonable result in a reasonable amount of time rather than not find any result due to the complexity of finding the best one. For A* will be compared the different heuristics taking into account its admissibility and consistency, taking into account which ones have better information.

Furthermore, search by genetic algorithms can be really efficient when the size of the problem grows, for that reason we will use an implementation of GA that uses codification by means of simple permutations and will compare the results obtained with the same instances, and bigger if possible, of the TSP problem used to test the spaces search algorithms.

The experimental results will be based in the aima-java software that implements the algorithms described at [1].

2 The TSP problem

TSP problem, also known as the traveling salesman problem, is a very famous problem where a salesman has to visit N cities, but it can only pass through each city one single time. Moreover, the city where he starts must be the same where he finishes. And then comes the problem, find the shortest path that meet the previous restrictions. In other words, we have a salesman and cities. Then we must:

1. Travel all the cities.
2. Start and finish at the same city.
3. Pass just one, and only one time by each city.
4. The path that the salesman follows must be the shortest one.

With the above restrictions we may think that this problem can be solved by brute force, for example by computing all the possible paths and then selecting the shortest one. But that is just not possible when the problem grows as the number of possible paths is the factorial of the number of cities (N!). For 4 cities we will have 24 paths and for 15 cities, for example, we would have 1,307,674,368,000 possible paths.

3 Heuristic search algorithms

As described at [2] Intelligent Search Algorithms or informed ones use some problem specific knowledge and take advantage of it to improve its performance.

3.1 A* algorithm

Was proposed by Peter E. Hart, Nils J. Nilsson and Bertram Raphael at [3] and it is very well defined at [4] and [5]. Its formal properties are well explained at [6]. It is a particular case of the Best First (BF), where $f(n) = g(n) + h(n) \cdot g(n)$ defines the best possible cost from the initial state to n to the moment. Notice that h(n) therefore is a positive estimate of the minimum cost from n to the nearest final state. The h function has to defined in such a way that it adds information about the problem domain. Its cost is not something not to take care about, and that's why ideally should be polynomial. At the aima-java prototype we have 3 different versions (normal, rectify and reinsert), described at [6].

3.2 Static weighted PEA* algorithm

It is a version of the A* algorithm where $f(n) = g(n) + (1+\varepsilon)\cdot h(n)$, where ε is an small value ($\varepsilon \geq 0$). It is inside the ε-acceptable algorithms, those who probably will find a solution which cost won't exceed (1+ ε) C, where C is the cost of the best solution. This kind of algorithms are not admissible but they reduce the time that it takes to find a solution and at the same time they bound the cost of the solution found by means of ε.

4 Genetic algorithms

They are based in probability and natural evolution. Solutions are coded by chains of symbols, called chromosomes. The starting point is a fixed size population of chromosomes and it is evolved though some generations (by means of the cross operator). On each "evolution" it is intended to keep those qualities from the previous generation that are relevant. To evaluate the quality of a chromosome, it is coded (to transform it as a solution) and then it is applied a fitness function, that, it is the unique that adds knowledge about the problem. GA are well described at [7].

5 TSP problem resolution

As we have seen before the TSP problem cannot be solved by means of brute force, therefore we must model the search state space and the different heuristics that we will explore for the heuristic search algorithms and define the codification schema, evaluation function, and the mutation and cross operators for the genetic algorithms.

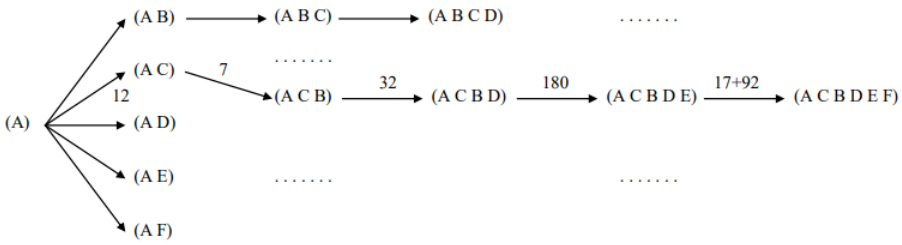
5.1 Resolution with search of state spaces

With search in space of states we can represent our problem both, as a tree or as a graph. Depending on the representation the solution will be found in one way or in another.

5.1.1 Tree representation

If the problem is represented as a tree some important considerations must be taking in to account:

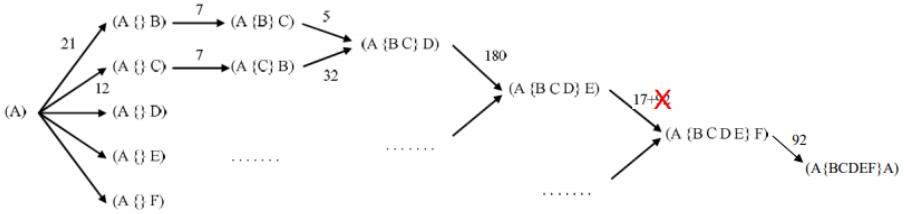
1. The number of target nodes is exponential (factorial). Each one will represent a solution.
2. Each state represents the path followed from the initial state to it.



5.1.2 Graph representation

If the problem is represented by means of a graph the number of nodes is heavily reduced. Some remarks of using a graph to represent the problem are:

1. There is only one target node/state.
2. Then, the solution is the path from the initial state to the target one.
3. Each node represents a sub-problem, but not the way to get to it form the initial.
4. Each path from a node to the target node represents a possible solution for the problem represented by the node.



As can be easily seen the graph representation for the A* algorithm improves its performance and therefore this representation of the problem will be the used during the experimental analysis.

5.1.3 Heuristics

When we use informed search, we must define the heuristics that will be used by the algorithms to improve its performance. In hour case those will be:

$h_0(n)$ = Null heuristic. It is admissible and consistent.
$h_1(n)$ = Minimum cost of the arcs that touch the last visited city and each non visited city.
$h_2(n)$ = Sum of the minimum arcs that touch the last visited city and each non visited city.
$h_3(n)$ = Average of the arcs that touch the last visited city and each non visited city.
$h_4(n)$ = The minimum spanning tree over the problem weighted graph.

5.2 Resolution with genetic algorithms

As described at the introduction we will use a permutation of a codification of the cities. Therefore, the cities will be coded as 1, ..., n-1. Then:

1. A chromosome is a permutation of the cities {1, 2, ..., n-1}.
2. The cross operator will be OX, order crossover.

That way the initialization will be performed by random permutations that generally will not be good, then the different mutations will be done with the cross operator defined previously and the evaluation will be done through the fitness function $Fitness(T) = {}^{1}/_{Cost(T)}$.

6 Experimental analysis

In order to compare the results of the search in space of states it will be use as a metric the number of expanded nodes, instead of the time taken[1] to find a solution. And the cost of the solution found, for those algorithms that end its execution in a reasonable time.

6.1 Data Set

The data set employed to perform the measurements is composed by instances of the TSP problem for which the cost of the best solution is known.

Table 1. Instances of the TSP problem used as data set for the experimental analysis.

# of cities	Instance	
	Cost	Problem name
6	115	Problem_1 (aima-java)
6	21	Problem_2 (aima-java)
6	85	Problem_3 (aima-java)
10	126	Problem_4 (Appendix I)
10	85	Problem_5 (Appendix I)
10	122	Problem_6 (Appendix I)
15	157	Problem_7 (Appendix I)
15	337	Problem_8 (Appendix I)
15	1377	Problem_9 (Appendix I)
17	2085	TSPLIB gr17
21	2707	TSPLIB gr21
24	1272	TSPLIB gr24
48	5046	TSPLIB gr48
96	55209	TSPLIB gr96

6.2 Experimental results

We will try to execute A*, PEA* and the Genetic version algorithms for all our instances. After, we will try to record the relevant metrics that allow us to compare the performance of the different methods.

[1] This is because the time is relative, it depends on the machine where the algorithm is executed and it also changes from execution to execution.

6.2.1 A algorithm*

Table 2. Number of expanded nodes / solution found cost by A* over a graph search space for the different instances.

Instance name (size) (cost)	heuristic				
	h_0	h_1	h_2	h_3	h_4
Problem_1 (6) (115)	28/115	21/115	21/115	48/115	12/115
Problem_2 (6) (21)	81/21	50/21	44/21	81/25	16/21
Problem_3 (6) (85)	55/85	40/85	36/85	81/103	28/85
Problem_4 (10) (126)	2273/125	1956/126	1711/126	2305/128	746/126
Problem_5 (10) (85)	1415/85	931/85	619/85	2305/95	122/85
Problem_6 (10) (122)	1136/122	946/122	743/122	2305/257	231/122
Problem_7 (15) (157)	74707/157	73416/157	72714/157	114689/165	55063/157
Problem_8 (15) (337)	63180/337	55648/337	41813/337	114689/739	12918/337
Problem_9 (15) (1377)	79477/1377	70112/1377	63843/1377	114689/2831	30870/1377
TSPLIB gr17 (17) (2085)	524289/2085	490436/2085	411108/2085	524289/2402	30891/2085

For instances gr21, gr24, gr58, gr96 the algorithm did not find a solution in a reasonable time.

From the Table 2 we see the results of the execution of A* in a graph search space configured with the different heuristics and for all the instances for which a result could be obtained. From it we can see that all the heuristics are admissible and consistent, except h_3 that is only admissible but not consistent.

Table 3. Mean of expanded nodes by A* over a graph search space for the different instance sizes.

Instance size	heuristic				
	h_0	h_1	h_2	h_3	h_4
6	54.6	37.0	33.6	70	18.6
10	1608	1277.6	1024.3	2305	336.3
15	72454.6	66392	59456.6	114689	32950
17	524289	490436	411108	524289	30891
24	The algorithm does not finish in a reasonable time or has not enough memory.				
58					
96					

Table 3 reduces the information of Table 2 and shows how the different heuristics are informed. We can already see that h_4 is the most informed heuristic as it always needs to expand less nodes to find the best solution. But to illustrate that, see Figure 1.

From these results can be seen perfectly that, without considering h_3, that it is not consistent, $h_0 < h_1 \leq h_2 \leq h_4$. As these heuristics are monotonous, we can assure that h_4 is the heuristics that has the best performance for this problem and A* implementation. Hence, as h_4 dominates other heuristics then every node expanded by h_4 will be expanded by h_2, h_1 and h_0.

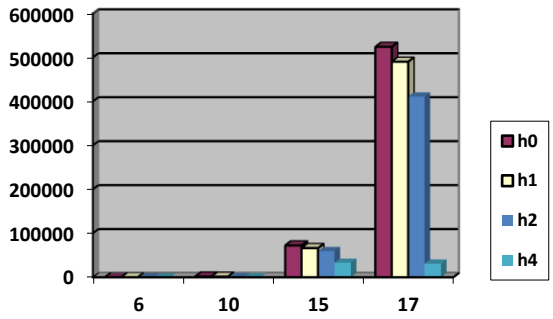


Figure 1. Mean of nodes expanded by A* for h_0, h_1, h_2 and h_4 over instances of size 6, 10, 15 and 17.

Figure 1 is very representative of how different heuristics work. Here we can see that the number of nodes expanded by h0, h1 and h2 grow exponentially meanwhile the number of nodes expanded by

6.2.2 PEA algorithm*

Table 4. Mean of the cost and the # of expanded nodes by PEA*, with h2, for the instances of size 10 and 15 and for ε value between 0 and 5.

	ε value							
	0 (A*)	0.1	0.2	0.3	0.5	1	2	5
Obt. cost	111	111	111	111	111	111	114.6	121
#nodes exp.	1608	1566.6	1525.3	1481.6	1386	1024.3	397	344.3
Obt. cost	623.6	623.6	623.6	623.6	623.6	623.6	625	690
#nodes exp.	72454.6	71360	70235	69106.3	66847	59487	38958	12913

At Table 4 we can see that the results of PEA* for the different ε values. There is not any theory result that assures that increasing ε will decrease the number of expanded nodes. We can also see that for ε = 5 the obtained costs differ from the best ones because the algorithm distances from A*. Even though, the error still is bounded by $(1+ \varepsilon)\cdot C^*$, as can be seen, at the 15 cities instances with ε = 5: $(1+5)\cdot 623.6 = 3741.6 \geq 690$. We must consider that greater ε values allow us to solve bigger problems, for example ε = 5 allow to solve up to 96 cities instances. With this results a good ε for the problem could be between 0.1 and 1.

6.2.3 Genetic algorithms

Table 5. Average (of 10 executions) of the cost of the best individual in the initial population (Cost pop. initial), cost of the best individual in the final population (Final population cost) and time in milliseconds (Time (ms)) obtained by the genetic algorithm with the crossing operator in two OX points (AG op. crossing b.), and with the bad crossing operator and without climbing the fitness (AG op. cross m.), for instances of TSP cost 115 (single instance) and 2085 (gr17 of TSPLIB).

Kind of GA and instances		Results		
		Cost initial pop.	Cost final pop.	Time (ms)
GA op.	Cost 115	115	115	9054.5
Cross b.	Cost 2085	3453	2311.4	40348
GA op.	Cost 115	157	136	6160.8
Cross m.	Cost 2085	3481.9	3481.9	15255.6

Table 12 shows the results of the TSP problem when using AG with the following features: coding with permutations, initial random population, population size and number of generations 150, probability of crossing 1 and of 0.2 mutation, two different crossing operators (two-point crossing and bad crossing (no transfers to the children relevant characteristics)), operator of selection by roulette (with fitness scaling, except for bad crossing) and with elitism. With the crossing operator at two points the results are acceptable: for cost instance 115 find the optimum in the initial population, while for the 2085 cost instance, the cost average obtained in the initial population is 3453 and ends with an average cost of 2311.4. With the bad operator the results are much worse: for the instance of cost 115 starts from a cost of 157 and ends with a cost of 136, while for the of 2085 part of a cost of 3481.9 and does not get any improvement (does not converge).

7 Conclusions

For the TSP problem and after our experimental study we can conclude the following things.

1. We can perfectly apply to this problem different search techniques, in this case we have used search in space of states and genetic algorithms, both serve as a search mechanism for finding a valid solution.
2. When talking about search in space of states we can use different algorithms to find a solution for a problem, in this case we used A* and PEA*, that as seen is and static weighted ponderation of A. And thanks to that static weighted ponderation it allows us to find error-bounded solutions incredibly faster that A* would.
3. A* works along heuristics and during the experimental study we've seen how important a well-defined heuristic is. In our case we'd used h_0, h_1, h_2, h_3 and h_4, where h_4 was the most informed one and therefore the one that obtain solutions more efficiently.

4. Another valid search technique is Genetic Algorithms. Here we've seen that they can obtain solutions for the problem in an acceptable time and cost. Their performance can be compared to the one obtained by PEA* for problems up to size 17. But looking at the results obtained when the problem size grows up and up genetic algorithms become more and more necessary because other techniques require too much time.

Therefore, on A* heuristics are very important, they should be admissible and consistent. But sometimes the cost of finding a solution is too high and other techniques are also valid, for example PEA* that allows to find a near-optimum solution in a reasonable time. And for the times where the problem is just too big for search in space of states we can use Genetic Algorithms to find a good solution.

References

1. Pearl, J., Heuristics, Morgan Kauffman, San Francisco, CA, 1983.
2. Endriss, U., Search Techniques, Universitatea Babeş-Bolyai, 2018.
3. Korf, Richard. Depth-first Iterative-Deepening: An Optimal Admissible Tree Search". Artificial Intelligence 27: 97–109 (1985).
4. Peter E. Hart, Nils J. Nilsson & Bertram Raphael: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics 4(2):100–107, 1968.
5. Nilsson, N., Principles of Artificial Intelligence, Tioga, Palo Alto, CA, 1980.
6. Palma Méndez, J. T. y Marín Morales R. Inteligencia Artificial: Técnicas, métodos y aplicaciones, McGraw-Hill, Murcia, 2008.
7. Kramer, O. Genetic Algorithm Essentials, Springer, 1st ed. 2017.

Appendix I

Problem_4: https://gist.github.com/thewilly/7ed5cd8a6576eeaa5375fc0c2a480357
Problem_5: https://gist.github.com/thewilly/87117214f3ae7bc6f56878ccb038fc3e
Problem_6: https://gist.github.com/thewilly/4e4a490e55188d7f2fa560515b078dfb
Problem_7: https://gist.github.com/thewilly/3e93eb2cf8fa923d3001064664fbc68e
Problem_8: https://gist.github.com/thewilly/bef5321d65010f0f632a0eef013a0ef8
Problem_9: https://gist.github.com/thewilly/8f72020ece064a21da2efa9d30b238ba