Mathematical and Computational Applications, Vol. 16, No. 1, pp. 53-65, 2011.

SEARCHING FOR THE SHORTEST PATH THROUGH GROUP PROCESSING FOR TSP

İbrahim Meşecan[1], İhsan Ömür Bucak[2], Özcan Asilkan[1]
[1] Epoka University, Computer Engineering Dept., Rruga Durres - Tirana, Albania
[2]Mevlana University, Computer Engineering Dept, 42003 Selçuklu - Konya, Turkey
imesecan@epoka.edu.al, iobucak@mevlana.edu.tr, oasilkan@epoka.edu.al

Abstract- Thanks to its complexity, Traveling Salesman Problem (TSP) has been one of the most intensively studied problems in computational mathematics. Although many solutions have been offered so far, all of them have yielded some disadvantages and none has been able to claim for the best solution. We believe that better solution could be obtained through iterative evaluations, until a certain number of islands are reached, if we could develop an algorithm which grows geometrically. Some algorithms have suggested random solutions and many suggested using the closest neighbors. In many cases islands exist in groups or chains in any length. Therefore they can be connected to any other island rather than the closest one. This can be better identified when we spot out the patterns and island chains. In this paper, we have searched for the identification of patterns and chains. We propose an iterative Group Processing (GP) approach which finds better paths in the 90% of the cases overall as we compare it to Random Logic (RL) programs and most up-to-date Artificial Neural Network based TSP programs.

Keywords- Group Processing, Traveling Salesman Problem

1. INTRODUCTION

Traveling Salesman Problem is one of the well-known problems in informatics, and is called an NP complete (completely not possible) problem. The traveling salesman problem was first challenged by Hopfield as an interconnected network [1]. A salesman must visit n cities while visiting each city exactly once and finishing at the city he started from. The cost of journey $c(i, j)$ to travel from city i to city j is given, and the salesman desires to make the tour whose total cost is minimum [2].

In some other works, the traveling salesman problem is identified to find a minimum cost (or, shortest path or distance, or minimum time) for a given set of vertices (cities) and edges (road) that constitute all the candidate cyclic paths. In other words, the path selected with the minimum cost is known as a Hamilton path which, in addition, must contain all the cities given, each only once, and begin from the specified city to which the tour ends [3]. The number of all the cyclic paths for the symmetric traveling salesman problem is calculated by $(n\text{-}1)!$. $n!$ is approximately equal to $(2\pi n)^{1/2} exp(-n)n^n$ according to Stirling's approximation which is a commonly used to approximate relationship for the evaluation of the factorials of large numbers. It is evident that this approximation requires an exponential order of times to calculate the minimum cost or select the shortest path by the blind search. No polynomial time solution with the order of n is believed to exist because of the reason above, and it is recognized that the traveling salesman problem belongs to a NP-class problem [3].

Assume that there are n cities numbered from 1 to n, and let the first city is the base-city of the salesman to visit. Apparently, there are (n-1)! possible solutions if one wants to identify all the solutions. Someone could probably check them systematically, find the cost for each and every one of these solutions, and finally reach one with the minimum cost. These, at least, require (n-1)! steps. For example, if there are 21 cities, the steps required for that will be (n-1)!=20!=2.432x10^{18}.

There are approximately 31,557,600 seconds in every year. And, if every one billion steps required a second, which is a huge number of operations in a second, then, one would need 77.094 (at least *seventy seven*) years of continued calculations. For 101 cities, this number will be equal to 100! = 9.332x10^{157}. It is easier to see how big the possibility is. Apparently, the exhausting examination of all possible solutions is out of the question. But the question is: "*Whether or not, we can develop an algorithm which grows geometrically, and thus, we check all the possibilities?*"

2. EARLIER METHODS

Some of the earlier methods tried to check all the possibilities that were nearly impossible to find the solutions. Because of huge number of possibilities, many chose to try for randomized search. However this one always opens a door for skepticism since one can never be sure for the best solution. If it is tried more than one, then there is always a question in mind whether or not there could be a better solution. The following subsections present some of the earlier algorithms briefly.

2.1. Exhaustive search and Backtracking

Sometimes the only way to solve a problem is to try all of the possibilities. This method is okay up to 10 or 15 islands, but more than that amount of islands, this method will always be slow. Exhaustive search systematically searches for a solution to the problem from all the options available.

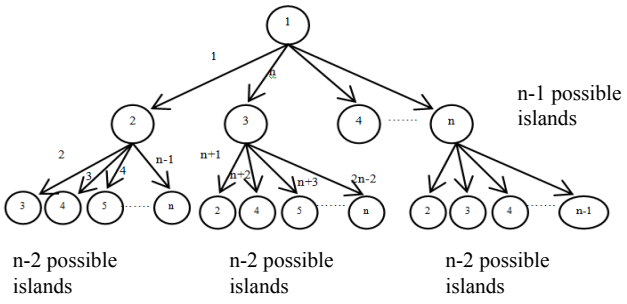

Fig.1: Exhaustive search.

As seen in Figure 1, the second island is selected and all the possible paths with it are tried thereafter. Upon finishing all those paths, that second island is returned and replaced with another one; this time all the possible paths with the third one are tried and it goes on and on.

Going down in a tree corresponds to a forward progress towards creating a more complete solution; going up in the tree corresponds to *backtracking* to some previously

generated partial solution. It might be worthwhile to proceed forward again from this point.

Backtracking can be seen a pruned exhaustive search. For example, if a minimal result is found on a point, and if that minimum is already passed while checking another possible position, the rest of the operations can be skipped. Therefore, backtracking is faster version of exhaustive search which could otherwise take many decades to try all these possibilities. For example, in a city of twenty one, it would take one to five years, not seventy seven years as in exhaustive search.

2.2. The Use of Inversion (2-opt) to Find an Approximate Solution

Goldberg used *inversion* to find an approximate solution to DNA computing [5]. This idea has also been applied to the traveling salesman problem. Some other scientists named this solution as 2-opt [6]. 2-opt is a local search algorithm in which a 2-opt move delete two edges, thus breaking the tour into two parts, and then reconnect those paths in the other possible way. This is equivalent to reversing the order of cities between the two edges [7]. At the outset, all islands are connected randomly as seen in Figure 2:

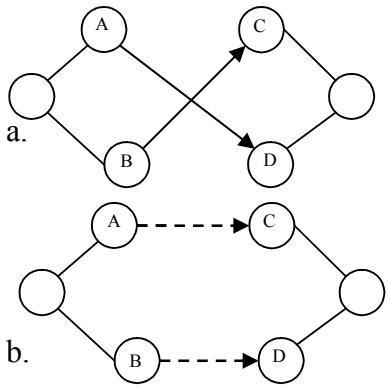

Fig. 2: a. Randomly connected islands at the outset, **b.** The use of inversion (2-opt) to find an approximate solution.

Two random islands like A and B in Figure 2 are untied and connected by inverting their connections. Then, it is checked whether this produces a shorter path or not. If the new path is shorter, it is accepted. If the new path is longer than the previous one, it is accepted with a decreasing possibility determined by $e^{(-(\text{dist2-dist1})/\lambda)}$. Since λ is defined as $0 < \lambda \leq 1$ and is always decreasing close to 0, the power of exponential term, δ=(dist2-dist)/ λ, will grow geometrically. Therefore $e^{-\delta}$ will result in a geometrically decaying number after each step.

From such a huge number of possible paths, it is very easy to enter in a wrong path. Accepting the longer path with a decreasing possibility gives the chance to quit the wrong path. If the new path again produces a longer path, we would easily come back to the old one. However, out of the randomization, one can never be sure to find the shortest path. Therefore, this method is mostly used to find a meaningful shorter path in a meaningful time space.

2.2.1 Complexity

This program repeats each step 1000 times and uses a constant, say λ, to decrease the possibility in an exponential manner. λ is started at 1 and is decreased all

the time by another constant η which is set to 0.95. And, this process is repeated while λ is greater than 0.001 that is equal to a total of 66000 times for each run. To compare the programs, we repeated this operation n times where n is the number of islands. Hence, for 50 islands, we try 3 million 300 thousand times.

2.3. Minimum Spanning Tree

The minimum spanning tree starts from an arbitrary root vertex r and grows until the tree spans all the vertices in V. At each step, a light edge is added to the tree A which connects A to an isolated vertex of $G_A=(V, A)$. By this rule, it only adds the edges that are safe for A. Hence, when the algorithm terminates, the edges in A form a minimum spanning tree. Upon forming the minimum spanning tree as in Figure 3, the TSP chain is called n-constructed and is refined [8].

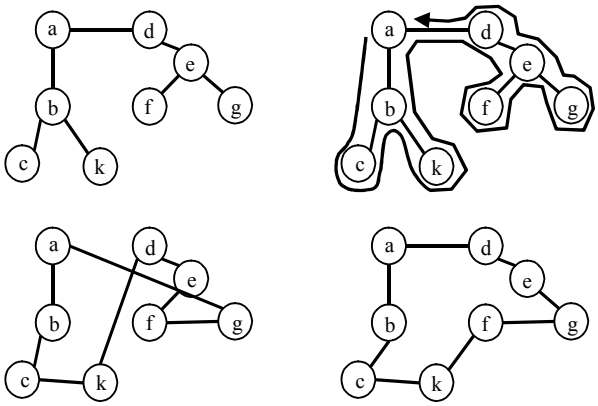

Fig. 3: Minimum spanning tree.

Again, there are too many possibilities, and it is very easy to fall into a wrong path which cannot be quitted from it. Furthermore, the initial assumption of selecting the closest possible island which is supposed to lead to the shortest path is incorrect.

2.4. Artificial Neural Network Solutions

The success in many difficult problems has made the neuro-computing method a very famous one. Angeniol *et al.* [9] first applied the Kohonen's Self Organization Feature Maps (SOFM or practically SOM) to the TSP problem [8]. In SOM, the neurons are allowed to move freely around. After each iteration, the current neuron is moved to another place to produce a better match (i.e., closest match) with its neighbors.

SOM has been applied to a variety of problems. Thus, many improvements have been provided to the algorithm based on the application. In one of these presented by Brocki, Kohonen's SOM has been applied to the TSP problem [10].

2.4.1 TSP with SOM

In 1975, Kohonen introduced a new type of neural network that uses competitive, unsupervised learning. This approach was based on WTA (Winner Takes All) and WTM (Winner Takes Most) algorithms. The most basic competitive learning algorithm is WTA [11].

When an input pattern is introduced, each neuron's synaptic weight is recalculated. The neuron that affects the most is said to be the winner. Therefore, its synaptic weight is modified. However, WTM has better results because more neuron synaptic weights are modified during a one-learning-iteration.

In the beginning the input neurons are placed in a random order and all are connected to output neurons. By continuously modifying the synaptic weights of the input neurons, they are brought to their closest island while producing a meaningfully shorter path.

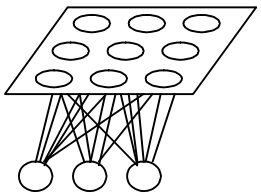

Fig. 4: Connecting input neurons to the cities.

When each city catches only one node from the output layer, the training is finished and the optimum path is obtained. Basic SOM algorithm is described as follows [10]:

```
procedure train_SOM
  begin
    randomize weights for all neurons
    for (i = 1 to iteration_number) do
        begin
            take one random input pattern
            find the winning neuron
            find neighbours of the winner
            modify synaptic weights of these neurons
            reduce the η and λ (the radius for neighbours)
        end
   end
```

This algorithm for SOM applied to the TSP problem to match the input and output layers is far away from the optimum path. Some more improvements need to be done before achieving the optimum path. As a result, it continuously processes according to the neighbours and changes the synaptic weights to find the optimum path. But, in the end, Brocki concludes that SOM approach can generate solutions that are almost less than 10% worse than the optimal tour [10].

3. GROUP PROCESSING (GP)

The idea for group processing has come out from the fact that "For such a geometrically growing question, we need to find a solution that can grow geometrically too, thus, enabling us to check such a huge number of possibilities. The solution has two main parts:

i. Make island groups starting from 1 until n-1, and then search for the best place for these groups,

ii. Try to find conflicts and refine them.

3.1. Closest Neighbour Search?

Many people believe that islands should be connected to their closest neighbours or to one of the close neighbours. Generally, this is an idea for a good start. On the other hand, once the search is limited, the optimum paths cannot be found. Many scientists did their research according to the assumption of the closest neighbours [9, 12]. Some others who, for example, developed α-nearness idea as part of Lin-Kernighan solution

assert that 70-80% of the edges which are in the minimum spanning tree are also included in the optimum paths [6]. Thus, when an edge is included in the minimum spanning tree, the change in the total length of the minimum spanning tree is the α-nearness value. On the other hand, this problem is completely random. Because of the formed groups, any island can be connected to any other island as in the example below. Here is the list for 12 islands:

0 288 149	1 270 133	2 288 129	3 256 141
4 256 157	5 246 157	6 236 169	7 228 169
8 228 161	9 220 169	10 212 169	11 204 169

We have 12 groups of 3 numbers where the first number is the island number; the second and the third numbers are *x* and *y* coordinates. Then, the optimum tour together with the island numbers would be like the following one (See Figure 5):

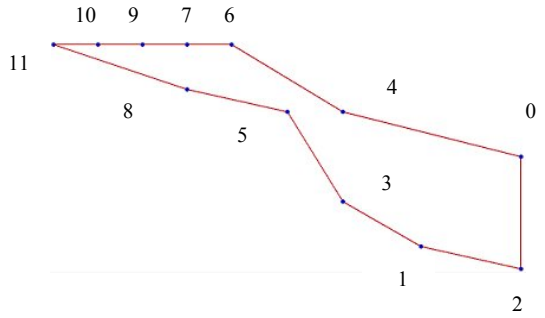

Fig. 5: A possible optimum tour.

If we calculate the nearest neighbours for the islands, the first seven close neighbours of the island 8 will be: 7, 9, 6, 10, 5, 11, 4 where 7 is the closest and 4 is the farthest. For a 12-island set, people may think that the islands should be connected to their first three or four close neighbours. Nonetheless, it is connected to its fifth and sixth close neighbours. As can be seen, the islands can also be connected to other farther neighbours.

Here is another example in Fig. 6 in which islands A and B are connected to their one of the farthest neighbours.

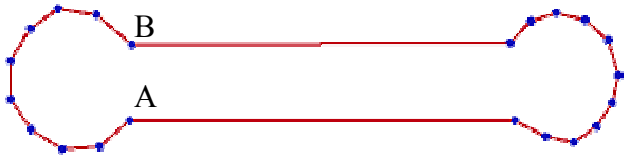

Fig. 6: Another optimum tour possibility considering the farthest neighbours.

Another well-known example is from TSPLIB, 532 cities of the USA: one island is connected to its twenty second closest neighbour [6]. Therefore, out of these and many other examples, we can see that the closest neighbour will not work for many cases. Firstly, it is important to identify the patterns and island chains, then, obtain the optimum path. In Group Processing, we try to identify these patterns.

3.2. Step 1 - How Group Processing works?

Before starting out with the details of the algorithm, let us explain a data structure that we have taken advantage of to reduce the number of possibilities; we use circular linked lists. Like a string, when you pull from one side, the other side is also

affected. Similar to how we omit the rest of the options in backtracking when the current path length exceeds the current minimum, a circular linked list helps us to automatically reduce the number of possibilities.

Group Processing, firstly, assumes the initial order as an initial path. Then, one by one detaches the islands and searches for the best location to attach. Meaningfully, we place the islands to their closest islands. Then, we repeat the same process for the groups of 2 islands, groups of 3 islands, etc.

Let us assume that the initial island positions and order as follows: (141 360), (206 209), (209 295), (320 153), (344 367), (370 208), (374 277), (484 175), (498 229), (120 343).

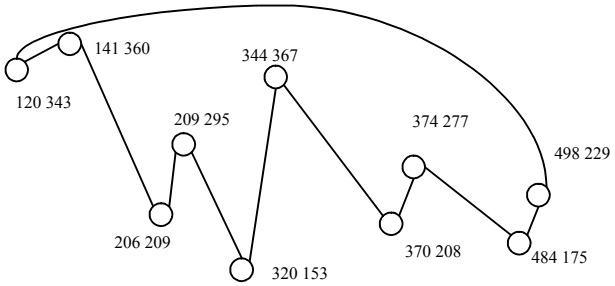

Fig. 7: Initial island connection.

It is noted that, normally the lines are straight, but for a better view some are used as curved. Figure 8 shows the connection upon the first island is detached from its position (141 360). Thus, the order of the island positions will be as follows: (206 209), (209 295), (320 153), (344 367), (370 208), (374 277), (484 175), (498 229), (141 360), (120 343).

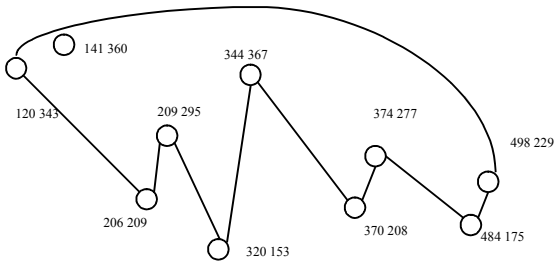

Fig. 8: The order after detaching the island.

After detaching the first island, the best place is sought through searching and the connection is provided afterward. In this case, the best place is found between the islands (120 343) and (498 229).

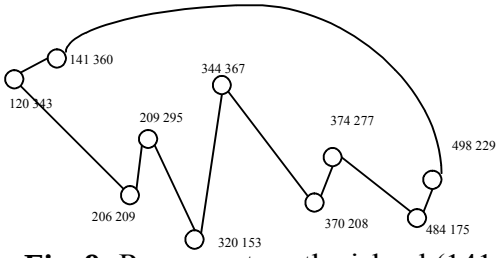

Fig. 9: Reconnecting the island (141 360).

Then, detaching the island (209 295) (see Fig. 10) rearranges the order as follows: (206 209), (320 153), (344 367), (370 208), (374 277), (484 175), (498 229), (209 295), (141 360), (120 343). Figure 11 shows the reconnection after this detachment.

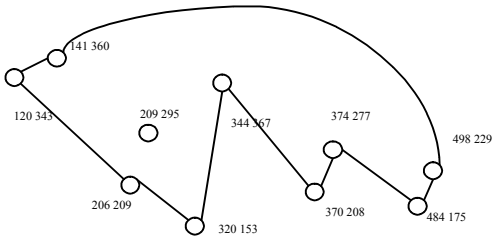

Fig. 10: The order after detaching the island (209 295).

Upon this detachment, the best place is searched for and the reconnection is provided. In this case, the best place for it is found between the islands (141 360) and (498 229).

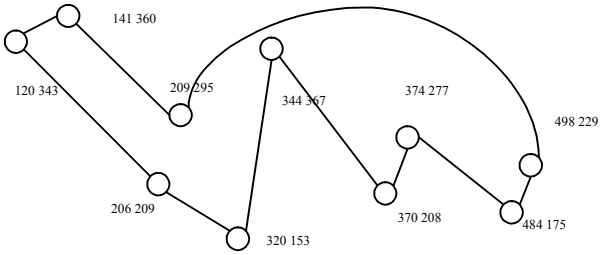

Fig. 11: Reconnecting the island (209 295).

This process is repeated for every island in the ring. After then, group size is increased. The islands are processed in groups of 1, 2, 3, and finally n-1. In the following, it is explained in detail how the case of processing in a group of 3 is formed. Suppose that an initial island connection is given as in Figure 12,

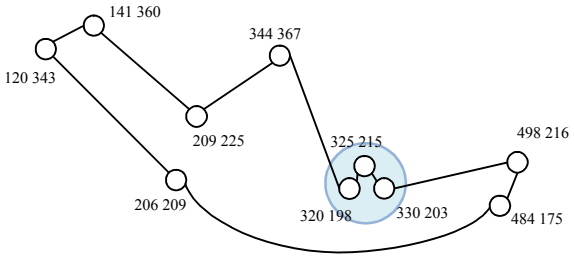

Fig. 12: Processing islands in a group of 3.

Figure 13 shows the connection after *Group A* elements are disconnected from the chain.

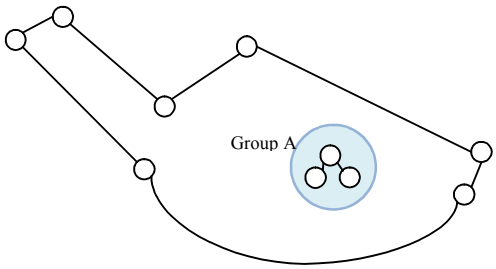

Fig. 13: Disconnecting *Group A* elements.

Here, the islands in *Group A* are treated as if it was a single island. Thus, a better place for it is searched and they are moved or placed there.

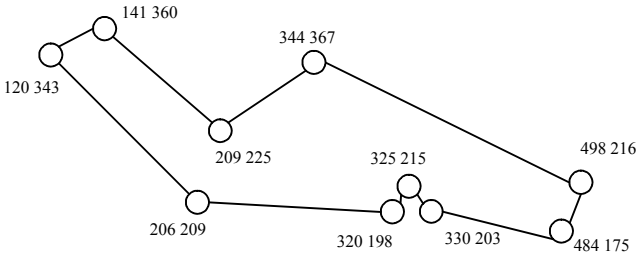

Fig. 14: Placing *Group A* elements back to the chain.

3.3. Step 2 – Randomization

The nature of this problem is completely random. There may be groups of any size starting from any point. Therefore, this group process is handled from a group of 1 until *n*-1. This helps us to catch these groups and place them to their best place. However, this is not enough to catch the best tour. When placing a group, there may be many places giving the same minimum result for this current moment. Normally, any of these minimum results may lead us to the exact solution. But, we either select the first minimum or the last minimum.

The best catch might be any other one. In linear search, we can catch either the first or the last occurrence, and there is no difference between them. If we try all these possibilities, it will be no different from *backtracking*. For this purpose, we change the start position to catch the other possibilities. As we use a circular linked list, while we change the start position, we will be able to catch the other skipped options.

In our case, as in Figure 15-a, when we start from position 0 in a circular linked list, we will catch the minimum value in position 10. After checking the possible paths in the first, second or in third trials, by moving the start position, we can catch the item in position 0 (Fig. 15-b).

While we continue to move our start position, the element in position 4 can be caught as the minimum (Figure 15-c).

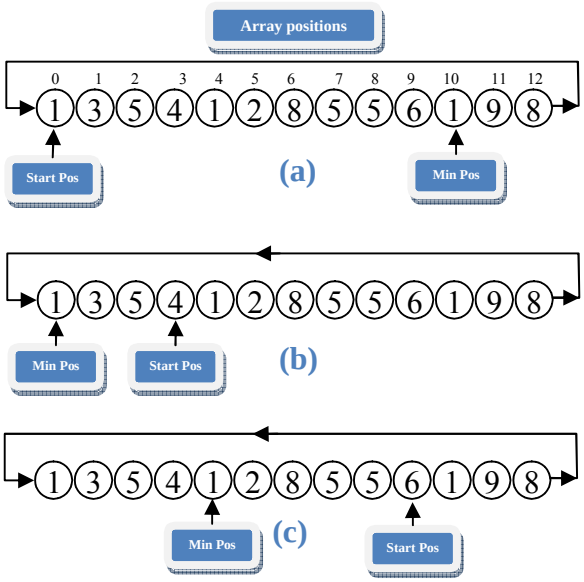

Fig. 15: Catching more possibilities by changing the start point.

For sure, this is not a constant situation and all the values are instantly changing. But in some other trials we will have similar situations and changing the start position will help us to catch these possibilities.

3.4. Step 3 – Refining the steps

Changing only the start position will not be able to catch all the possibilities. Thanks to the random nature of the problem, there could be many places that give us the same result for any group. Because of this, many people used random functions. However, using random functions have presented many drawbacks. Instead of randomization, we have developed a different method. After each iteration for the groups, we search for the longest edge; because any incorrect connection will cause an edge to be longer than its normal form. We search and then find a better place for it. One might think that this edge might be the real long edge. If so, the program will not be able to find any better position. Thus, we repeat this longest edge operation from 1 to the square root of *n*. Therefore, on the first pass, we search and replace only one longest edge, and on the second pass, we search and replace two longest edges, etc.

3.5. Complexity

iii. The inner most procedure *WalkAround* goes through the islands for the given group size. For each group it starts from the beginning until the end to search for a better position; therefore the procedure is called *n* times. In the beginning we have the group size 1, thus overall in the example we will have 100 groups. This process is executed for each group giving us the complexity as $n*n/2$.

iv. Procedure *CheckOne* calls *WalkAround* for all group size starting from an initial island. After processing each group, it also processes for the *n*-longest edges. The group size is incremented from 1 to *n*-1. This means that *WalkAround* procedure is called *n*-1 times.

WalkAround is a linked list application. Thus, when searching a better position for a group of islands, we start and search from the initial island until the final one. But after the first island, group positioning of the initial and final islands change place. This is a drawback. Thus *WalkAround* procedure may not check for all the groups. As a result, we are not able to check *n* times. But, running the *CheckOne* procedure *i*+5 times for this group size enables us to run *WalkAround* (*i*+5) times, so that for this group size starting from the beginning, we check better places for all island groups. By this way, this procedure *CheckOne* calls *WalkAround* $(i+5)*(n-1)$ times where *i* is a counter increasing from 1 to the square root of *n* and is explained in the 4th step. (*i*+5) gets the numbers from 6 to 15 but it cannot be calculated as an extra multiplier because it is a complementary of the *WalkAround* n complexity.

iii. *CheckAll* calls procedure *CheckOne* by changing the start position *n* times.

iv. And finally, *CheckAll* is called, by the main function, square root of *n* times for changing the longest *n* edges.

As a result, the complexity will be $4n\sqrt[4]{n}$. This is an exponentially growing complexity and is still huge. But compared to $n!$, it is very small.

4. COMPARISON OF THE ALGORITHMS

For comparison, we have had twenty island sets with different number of islands. After each run, we have noted the length of their best path. Then, we have checked for the winner as based on the integer part of the result by skipping the floating point parts. Winner of every check got one point. When they have had the same integer length, we have assumed both as the winner. In the end, we have taken the sum of the total paths and counted the number of wins. Here are the results presented in Table 1:

Table 1: Comparison of GP and RL programs.

#Trials	Best Distance			Who is winner?	
	#Islands	RLS	GP	RLS	GP
1	40	1810	1782	0	1
2	40	1648	1648	1	1
3	40	1887	1862	0	1
4	40	1724	1724	1	1
5	30	1527	1527	1	1
6	30	1560	1560	1	1
7	70	2173	2137	0	1
8	70	2203	2137	0	1
9	80	2480	2407	0	1
10	80	2495	2413	0	1
11	80	2500	2430	0	1
12	80	2396	2347	0	1
13	80	2497	2354	0	1
14	100	3617	3588	0	1
15	100	3992	3891	0	1
16	100	2700	2723	1	0
17	100	2778	2743	0	1
18	100	2697	2702	1	0
19	100	2796	2752	0	1
20	100	3887	3602	0	1
Total Distance		49367	48329	Winning Percentage	
Percentage?		1.021		30	90

where,

#Trials: Number of trials

Islands: Number of islands

RLS: Random logic solution using inversion (2-opt) method

GP: Group Processing Solution

As seen from Table 1, out of the 20% of the cases, GP and RLS have produced similar results. But in the 70% of the cases GP has found better results. In other words, GP has found very good paths in 90% of the total cases. Moreover, GP has produced 2.1% shorter paths in the total.

To be able to compare our program with the Artificial Neural Network based TSP programs, we have done a search and found several programs. Out of this search, the one from *sourceforge.net* [12] seemed the most serious and up-to-date work in this field

(August, 2008). It uses *NeuronDotNet 3.0* [13] (Release Date: August, 2008) Kohonen's SOM and itself have also produced better paths when compared to other ANN programs available to us such as the one by Lalena in [14].

Thus, we again produced twenty different input sets containing different number of islands and checked the results as follows:

Table 2: GP vs. ANN program from *sourceforge.net*.

#Trials	Best Distance			Who is winner?	
	# Islands	ANN	GP	ANN	GP
1	40	568	540	0	1
2	40	541	538	0	1
3	40	508	508	1	1
4	40	513	513	1	1
5	40	482	470	0	1
6	60	614	614	1	1
7	60	637	615	0	1
8	60	584	587	1	0
9	60	664	608	0	1
10	80	711	706	0	1
11	80	753	709	0	1
12	80	748	740	0	1
13	80	697	683	0	1
14	80	740	712	0	1
15	80	730	719	0	1
16	100	797	754	0	1
17	100	815	802	0	1
18	100	838	820	0	1
19	100	791	772	0	1
20	100	789	759	0	1
Total Distance		13520	13169	Winning Percentage	
Percentage?		1.027		20	95

where,

#Trials: Number of trials
Islands: Number of islands
ANN: Artificial Neural Network Solution
GP: Group Processing Solution

This time, we had better results. In total, GP has produced 2.7% shorter paths. Out of 95% of the whole cases, it was able to find very good paths. In 15% they found the same length path. Finally, the ANN was able to produce shorter path in just one case (5%).

5. CONCLUSION

Our main purpose was to find a solution to the question: "Since it is impossible to check all the possibilities one by one, can we develop an algorithm which grows geometrically, and thus, we check all the possibilities?" Finally, we developed such an algorithm which worked better than some famous algorithms.

In random logic one will always have a doubt, "if I run again, will it produce a better path?" Thus, one can never be sure if this path is good or if there is a better path.

Backtracking cannot be used for the questions that have more than twenty islands, because it takes a lot of time. Kohonen's SOM algorithm, like many others, does not claim to solve for the best solution. Instead, it aims for producing an acceptable solution in a meaningful time. It processes the islands according to their closest neighbours. To our opinion, Kohonen's SOM lacks the idea of processing island in groups. Moreover, ANN solutions also use random selection.

However, our algorithm does not use $n!$, because the islands are being processed in groups, and we search for better positions for the island groups. This takes less time to process as compared to the backtracking and exhaustive search algorithms. On the other hand, since it is not a random logic, one might know that in the next run, it will not produce a different path. Therefore, we have the advantage of backtracking and random logic in one program.

As it can be seen from the tables 1 and 2, the GP approach can find better paths in the 90% of the cases, and we achieve this in an iterative way.

6. REFERENCES

[1] J.J. Hopfield and D. Tank, Neural Computation of Decisions in Optimization Problem, *Biological Cybernetics,* **5**, 1994

[2] T.H. Cormen, C.E. Leiserson., R.L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd Edition, The MIT Press, McGraw-Hill, 2001

[3] Y. Li, Traveling Salesman Problem Based on DNA Computing, *Third IEEE International Conference on Natural Computation*, **4**, 28-34, 2007

[4] R. Takahashi, A Hybrid Method of Genetic Algorithms and Ant Colony Optimization, *International Conference on Machine Learning and Applications*, 81-88, 2009

[5] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Pub. Co., 1989

[6] K. Helsgaun, An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic, *European Journal of Operational Research*, **126**, 106–130, 2000

[7] I. Mavroidis, I. Papaefstathiou, D. Pnevmatikatos, Hardware Implementation of 2-opt Local Search Algorithm for the Traveling Salesman Problem, *18th IEEE/IFIP International Workshop on Rapid System Prototyping*, 41-47 , 2007

[8] K. Fujimura, H. Tokutaka, S. Tanaka, T. Maenou, and S. Kishida, *The optimization for TSP using SOM method of many cities (e.g., 532 cities in USA)*, Tottori University-Japan, 1997

[9] B. Angeniol, G. De-La-Croix, and J.-Y. Le-Texier, Self-organizing feature maps and the Traveling Salesman Problem, *Neural Networks*, **1**. 289-293. 1988

[10] L. Brocki, *Kohonen Self-Organizing Map for the Traveling Salesperson Problem*, Polish–Japanese Institute of Information Technology, 2007

[11] T. Kohonen, *Self-Organizing Maps*, Berlin: Springer, 2001

[12] http://sourceforge.net/projects/neurondotnet/files/
http://neurondotnet.freehostia.com/download.html Program last updated: Aug, 2008. Last Viewed April, 2010

[13] http://neurondotnet.freehostia.com/manual/ Program last updated: Aug, 2008 Last Viewed on April 2010

[14] M. Lalena, http://www.lalena.com/AI/Tsp/ release date: 2006. Last viewed Apr, 2010