Modern Applied Science; Vol. 12, No. 8; 2018
ISSN 1913-1844 E-ISSN 1913-1852
Published by Canadian Center of Science and Education

Travelling Salesman Problem Solution Based-on Grey Wolf Algorithm over Hypercube Interconnection Network

Ameen Shaheen[1], Azzam Sleit[1] & Saleh Al-Sharaeh[1]

[1] Computer Science Department, King Abdullah II School for Information Technology, The University of Jordan, Amman, Jordan

Correspondence: Ameen Shaheen, Computer Science Department, King Abdullah II School for Information Technology, The University of Jordan, Amman 11942, Jordan. E-mail: aminalshahin@gmail.com

Received: June 20, 2018 Accepted: July 12, 2018 Online Published: July 28, 2018

doi:10.5539/mas.v12n8p142 URL: https://doi.org/10.5539/mas.v12n8p142

Abstract

Travelling Salesman Problem (TSP) is one of the most popular NP-complete problems for the researches in the field of computer science which focused on optimization. TSP goal is to find the minimum path between cities with a condition of each city must to visit exactly once by the salesman. Grey Wolf Optimizer (GWO) is a new swarm intelligent optimization mechanism where it success in solving many optimization problems. In this paper, a parallel version of GWO for solving the TSP problem on a Hypercube Interconnection Network is presented. The algorithm has been compared to the alternative algorithms. Algorithms have been evaluated analytically and by simulations in terms of execution time, optimal cost, parallel runtime, speedup and efficiency. The algorithms are tested on a number of benchmark problems and found parallel Gray wolf algorithm is promising in terms of speed-up, efficiency and quality of solution in comparison with the alternative algorithms.

Keywords: grey wolf optimizer, chemical reaction optimization, meta-heuristics, Travelling salesman problem

1. Introduction

Due to the large increases in the number of cities in the world, mobility between cities has become difficult because of there existing many dissimilar roads to reach the same city with different travelling cost (Vukmirović and Pupavac, 2013), where there are several places that are all directly connected to each other by different long roads and the passenger wants to make the shortest trip. Some Algorithms can be used to guide people using one of the transport or movement methods (walking, train, car, and bus) to reach their destination on the shortest route. (Zhan and Noon, 1996).

TSP is arisen in many different practical applications such as School bus routes, Computer wiring, job-shop scheduling and many more (Matai, et al., 2010). While there is many applications of TSP then applied new algorithms and architecture could give the opportunity to inspire new solutions that could be better than existing ones, which means, optimizing for all of these applications.

TSP has received great interest from researchers and mathematicians, as it is easy to describe, difficult to solve. TSP problem has a place in a big class of problems known as NP-complete, as shown in Figure 1. In particular, if an efficient algorithm (polynomial time) can be found for solving TSP, then efficient algorithms could be found for all others. (Karla, et al., 2016).

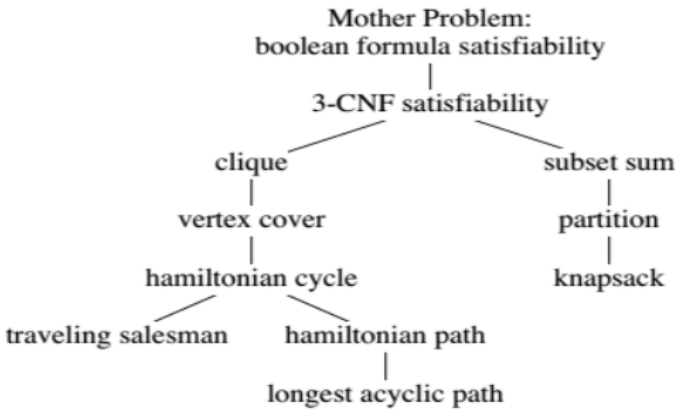

Figure 1. NP – Complete Problems (Al-Shaikh, et al., 2016)

TSP is the problem of finding the shortest route between cities or nodes which classified as minimization problem (Lam and Newman, 1985), the problem is to create the shortest route in the form of aggregation to visit each node exactly once and then return to the initial node (Kan and Shmoys, et al., 1985).

TSP is a permutation problem which required O (n!) as time complexity (Kaempfer and Wolf, 2018). There exist an algorithm called Held and Karp algorithm as in (Chekuri, 2017) based on dynamic programming to solve TSP where they reduce the computational time complexity to O (2nn2) but it's still too high for solving big size of real-world instance.

1.1 TSP Formulation

Miller (as cited in Sawik, 2016) shows that TSP can be defined as an integer linear program as follow:

$X_{ij} =$ { 1 There is rout from city i to city j ; 0 No rout }

Where x is a variable for an n city problem, by defining the Travelling cost between two cities i and j is ci.j and ui is a temporary variable, Miller shows how TSP can be defined as an integer linear program as :

$$\min \sum_{i=0}^{n} \sum_{j\neq i, j=0}^{n} c_{ij} x_{ij} \quad (1)$$

$$\text{Subject to: } 0 \leq x_{ij} \in \{0,1\} \leq 1 \qquad i,j = 0, \dots, n \quad (2)$$

$$u_i \in Z \qquad i = 0, \dots, n \quad (3)$$

$$\sum_{i=0, i\neq j}^{n} x_{ij} = 1 \qquad j = 0, \dots \dots, n \quad (4)$$

$$\sum_{j=0, j\neq 1}^{n} x_{ij} = 1 \qquad i = 0, \dots \dots, n \quad (5)$$

$$u_i - u_i + n x_{ij} \leq n - 1 \qquad 1 \leq i \neq j \leq n \quad (6)$$

Where equations 2 and 3 are used to ensure that each city on the route can only arrive from another city exactly once, Equalities 4 and 5, it is necessary for each city on the route, there is an exit to the another city. Finally, equation 6 is used to ensure that there is only one route covering all cities, which means that they cant be multiple, simultaneous and unconnected paths.

While the importance of the TSP in numerous fields and for its applications, many researchers solve it using different approaches aspiring to get a better solution than existing such as : Bee Colony Optimization (BCO) (Wong, el al., 2008), Ant Colony Optimization (ACO)(Yang, et al., 2008), Firefly algorithm (FA)(Kumbharana and Pandey)(2013) and Heuristic Algorithm (Hernández, el al., 2016). These methodologies do not generally locate the optimal solution. Rather; they will often find the near-optimal solutions for the problem. Most of these algorithms called Swarm optimization (SO) (Poli, el al,2007) or Meta-Heuristic optimization mechanisms (Blum, el al, 2007), SI is an intent of design or distributed problem-solving inspired devices by the collective behavior of colonies of insects social and other sociedades animals (Raja, 2015), which can be used to solve many types of optimization problems. As an examples in natural systems of Swarm optimization are include Bacterial Growth (Zwietering, el al.,1990), whales (Mirjalili and Lewis, 2016), Bird Flocking (Reynolds, 1987), Fish Schooling (Bastos, et al., 2008) and spiders (James and Li, 2015). Swarm intelligence techniques become very popular and commonly used in solving many types of optimization problems due to many advantages, some of these advantages are (Kordon):

- Easy to implement.
- Fewer numbers of parameters to adjust.
- Less memory to save (less space complexity).
- Obtained good results in responsible time.

Grey Wolf Optimizer (GWO) is a recently established SI optimization mechanism where their many researchers used it to solve many optimization problems such as Parameter Estimation in Surface Waves (Song, el al., 2015), Economic Emission Dispatch (Song, et al., 2015) and Scheduling problem (Komaki and Kayvanfar, 2015).

The inspiration for GWO is from a species of wolves called the Grey Wolf (Canis lupus), by imitating its hunting methods and hierarchical pack distribution, which are referred to as; Alpha α, Beta β, Delta δ, and Omega (ω); these are used to imitate the series of commands as shown in Figure 2.

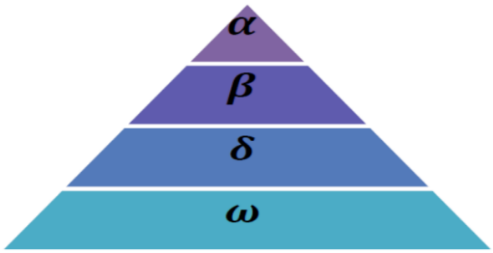

Figure 2. Hierarchy of Grey Wolf (Mirjalili and lewis, 2014)

As seen in figure 2, sovereignty reclines from top to bottom. The first level is Alpha (α), which is the leader, which is not necessary to be the most robust wolf but the superior to other wolves in managing the pack. Thus it is responsible for the decision making. The second level is Beta (β), which helps Alpha in decision making. Thus, it represents as the mentor to Alpha and an educator to the pack. The third level is Delta (δ) which controls Omega (ω). This category could be Scouts, sentinels, elders, hunters, and caretakers. Finally, the fourth level is Omega (ω) that acts as the scapegoat and gives up to all dominant wolves.

In the real world, there are many complicated events that occur simultaneously and in a temporal sequence like weather and galaxy formation, where that far exceeds the capabilities of single-processor architectures (Worboys, 2005).

Therefore, the concept of parallel computing seemed to increase performance and reduce computation time to solve these problems (Barney, 2010) where parallel machines break a single problem in parallel tasks that were performed simultaneously, as TSP problem. Parallel computing is much more suitable for modeling and simulating complex problems (D'Angelo, 2011).

Hypercube is a multi-dimensional mesh of processors with exactly 2 processors in every dimension; this means that a two-dimensional hypercube is made up of processors p = 2 ^ d. For example, a zero-size hypercube is a single processor. In general, a hypercube (d + 1) -dimensional is constructed by connecting the corresponding two-dimensional hypercube processors as shown in Figure 3 (Bhuyan and Agrawal, 1984).

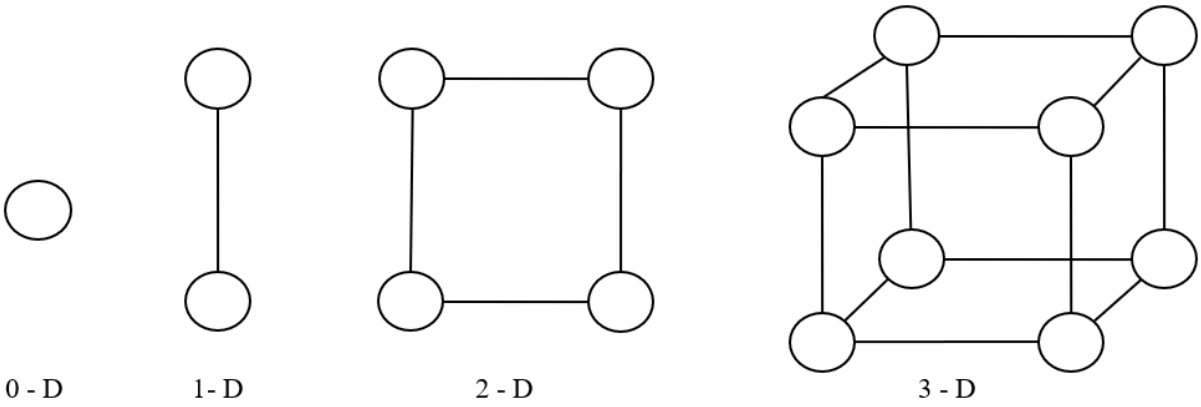

Figure 3. Hypercube Interconnection network

In this study, we decide to use hypercube because of its performance measure proved that it has a better performance compared with static network topologies (Kiasari, et al., 2008) wise of Diameter, Bisection Width, Arc Connectivity and the Cost as shown in Table1. And also, because of the topology of hypercube was implemented in many supercomputers such as Endeavour Supercomputer by NASA (Cathleen, 2011).

The main contributions of this paper are summarized as follows:

- This study adapted GWO to solving the TSP problem, its executed sequentially on different size of World TSP benchmarks and measure the performance in terms of execution time and optimal cost.
- To compare the result of the GWO with other meta-heuristic algorithms, GA (genetic algorithm) and CRO (optimization of the chemical reaction) are chosen and adapted to solve the TSP, the performance metrics are measured in terms of execution times and optimal costs.
- Development of the parallel GWO. The parallel GWO is developed on the basis of both data and computation distribution techniques through the hypercube interconnection network. Data distribution technique is designed based on dividing the dataset map (cities) with a goal of achieving load balancing among the interconnection network. Computation distribution is provided by distributing GWO iterations through the interconnection network to reduce the computing time.
- A comparison between PGOW (Parallel Grey Wolf Optimizer), PCRO (Parallel Chemical reaction optimization) and PGA (Parallel Genetic algorithm) in terms of execution time, parallel runtime, speedup, efficiency and optimal cost. The PGWO shows better performance results than PCRO and PGA.

The remaining of this paper is structured as follows: In Section II, a work related to this study is presented, section III addresses our parallel model for GWO for solving TSP, then in Section IV, the analytical evaluation of the sequential and parallel version of TSP-GWO is presented. Section V shows the discussions of our experimental results. Finally, conclusions and future work are in section VI.

2. Related Works

Several researchers have been conducting research on solving TSP and apply their algorithm on different topologies; below are some of these very recent studies.

A recent meta-heuristic algorithm used to solve TSP in (Kumbharana and Pandey, 2013), where authors used the Firefly Algorithm (FA) to solve TSP problem, the experimental results obtained on different size TSP instances. Authors show that the proposed algorithm provides better results than (ant colony optimization) ACO, genetic algorithm (GA) and simulated annealing (SA) in most of the instances. Also, in (Bhardwaj and Pandey, 2014), they presented a Parallel Ant Colony (ACO) algorithm to solve TSP in heterogeneous platform using the OpenCL framework. All the parameters of the algorithm in ant systemare been investigated to their best values as control parameters, where α and β represent the dependency of probability on the pheromone content or the heuristic and equal to $\alpha = 1$, $\beta = 5$ and $\rho$is the evaporation rate = 0.5. The parallel implementation is done on CPU and GPU using OpenCL, where GPU gives better results. In (AbdulJabbar and Abdullah, 2016), authors proposed a hybrid algorithm based on two metaheuristic methods: simulated annealing (SA) and tabu search (TS). The goal of using tabu search is to resolve the long computation times that take from SA by keeping the best-founded solution in

each SA iteration. By comparing with the basic version of SA, authors found the proposed approach reduces the time complexity by finding the optimal path (best solution) with a few numbers of iterations. In another study (Anjaneyulu, et al., 2014), they used approximation algorithms to find near-optimal solution, the approximation algorithm used for maximization or minimization based on the problem, when it comes to TSP it is minimization. They focused on a special case of TSP, which is Metric TSP (the distance between two cities is the same in each opposite direction), then they proposed a parallel two-approximation algorithm for metric TSP. Finally, they reported that the algorithm found near optimal solution with a significant reduction in runtime. In (Razali and Geraghty, 2011), authors have used the Genetic Algorithm (GA) but with Different Selection Strategies to solve the TSP problem, they firstly use Tournament selection strategy as a popular selection method in genetic algorithm then they try another selection strategy called Proportional Roulette Wheel Selection and as a final selection strategy, they use a Rank-based Roulette Wheel Selection. The algorithms are coded in MATLAB and tested by eight TSP instances. They found that GA with the rank-based roulette wheel selection always gives better solution quality than the other selection strategy.

Because of the importance of TSP and its applications (Applegate, et al., 2007), this study presents a solution to the TSP by using the GWO, where GWO is a recent establish meta-heuristic optimization mechanisms and its success in solving many types of optimization problems with good results. To compare the result of GWO with alternative meta-heuristic algorithms, GWO will be compared with GA and CRO. GA regardless of achieving great success in solving many optimization problems, it is also used for comparison in most meta-heuristic optimization research like in (Shaheen and Sleit, 2016), (Ross and Corne, 1995) and (Ingber and Rosen, 1992). Also, CRO used to compared with GWO because it is one of the newest meta-heuristic algorithm where it also obtained good results in solving NP problems as in (Barham, et al., 2016), (Shaheen, et al.,2018) and (Sun, et al., 1990).

3. Proposed Approach

GWO is a new nature-inspired metaheuristic (Swarm intelligence) where this type of algorithms are inspired by natural systems (Mirjalili, et al., 2014). GWO gets its name from the nature of the social hierarchy of wolves, as well as their hunting behavior. The Hunting behavior of Grey Wolves is split into four procedures: (1) Chasing, (2) Encircling, (3) Hunting and (4) attacking the victim. In Chasing phase, The Algorithm considers that α is the best solution; β is the second best solution and δ is the third best solution. However, ω represents the rest candidate solutions. Thus the hunting is led by the dominant wolves (α, β, and δ). In other words; Grey Wolves could recognize the position of the prey through an iteration process and surround it as in encircling phase, Grey Wolves bounded the victim through the hunt (optimization) by calculating the distance between the location of the prey. In hunting phase, The hunt generally is led be the leader (α). However, sometimes β and δ contribute in hunting. In another hand, there is no idea about the position of the prey that represents the optimum. Therefore, the algorithm assumes that α, β, and δ has preferable knowledge about the position of prey. Thus, the algorithm saves the first three best solutions then update the locations of the rest wolves (ω) depending on the position of the dominant wolves (best search agent) and in attacking phase where it's the final phase, the hunting proceeding obtained the optimization solution the prey stops proceeding.

In this paper, we used the concept of GWO to solve TSP. The proposed solution is implemented in two approaches. In the first approach, the algorithm is implemented sequentially using the standard JAVA programming language where the second approach of the implementation uses Java multithreading and aims to make the most of CPU processing power. The reason for implementing two approaches of the algorithm is to demonstrate the feasibility of the parallel structure. The approaches are implemented in the most logical way possible.

3.1 Sequential ALGORITHM: "GWO-TSP"

In GWO, there are four types of wolfs: Alpha (α), Beta (β), Delta (δ), and Omega (ω) and the prey, wolfs applied the hunting methods to hunt prey, this is being implemented in a hierarchical way until Alpha wolf take the decision of attack. TSP contains number of cities while there is a cost of Travelling between each pair of cities, the objective is to find the shortest path going through all cities, which means a simple cycle tour, which starts and ends at city 1. By applying GWO to find the possible solution for TSP, Figures 4 present the pseudo-code for the proposed "GWO_TSP" sequential algorithm, TABLE 1 shows the main attributes and their meaning related to the proposed algorithm "GWO_TSP", in comparison with wolfs meaning in GWO.

```
Algorithm1: Sequential "GWO-TSP"
Input: TSP problem.
Output : Shortest tour among all cities
1//Initialization phase
2 Population size [].
3 Preys : Initial and next preys selection size = 3;
4 Population [ ].
5 Select 3 random preys from city map and added as initial population.
6   While Population.size < Population [ ] and FullTour(Population [ ]) // Iteration phase
7   {
8            For each Population [i]
9                  Calculate the destination of all wolfs
10       X α = next best wolf from city map.
11       X β = next second wolf from city map.
12       X δ = next third wolf from city map.
13       Update Population as:
14                 Population [i]  =  Population [i] + X α .
15          Population [i+1] + X β  // New population
16          Population [i+2] + X δ  //  New population
17        Calculate fitness for each Population .
18  End for
19            If Population > Population.size
20                 Remove most costly tour.
21  End if
22  }
23  return X alpha // final stage
```

Figure 4. Pseudo code of GWO for solving TSP.

Table 1. Profile for GWO-TSP

GOW meaning	GWO-TSP meaning
Gray wolf population	Candidate solution: Tours
Prey	Strat city
Gray wolfs	The remaining cities
Alpha (α) wolf	Nearest city to the start city.
Beta (β) wolf	Second nearest city to the start city.
Delta (δ) wolf	Third nearest city to the start city.
Number of iterations	Number of solutions.
Fitness function	Current optimal TSP solution.

The GWO-TSP algorithm, as shown in Figure 4 (see lines 1, 6 and 23, present these three stages). First, the initialization stage can be shown in Figure 4 (see lines 1-5) to assign initial values for the algorithm parameters. Each individual in the Population is an array, which represents the maximum number of candidate solutions. While each of them consists a full tour as in Table 1. Next, initialize and assign the value of three preys, this is because as the concept of GWO, its assumes that first three wolves (α, β, and δ) has preferable knowledge about the position of prey and as Table 1 each prey represent a city from the city map. Population (see line 4) is an empty array used to constructing the candidate solutions. In order to start building solutions, three preys (or cities) will be selected randomly from the city map (data-set) and added as an initial population where all towns that surrounding them

are considered the grey wolves which is the final step in the initialization stage. The goal of Iteration stage is to generate and build the candidate solutions (or tours) until reaching the best solution. Iteration stage is shown in Figure 4 (see lines 6-22). After generating the three required population as in the initialization stage, each of them will contain a prey that surrounded by a group of wolves. The function (see line 9) used to calculate the destination of all wolfs from the prey then returns the nearest three wolfs from the prey and added to X_{α}, X_{β}, and X_{δ} respectively. Now, based on the positions of the three best wolves, the Xα will be added to the original population and two new population will be created and added the prey in each one plus Xβ and Xδ respectively. That's mean, each population will generate two new population and each of them will contain two preys. In other words, after the first iteration, the number of population will be equal nine and each of them will contain two preys. Next, the algorithm will calculate the fitness for each population, which means, the cost of Travelling between cities. While increasing of number of population, the function (see line 20) used to remove the most costly tour. This will happen when the number of population is larger than the allowed population size. Iteration stage keeps working until reaches the stopping criteria, two stopping criteria used to stop the iteration stage. Firstly, the number of population must be larger than the allowed population size, and each population should be contained a full solution. Which means, as equal as the number of cities, this is done through FullTour function. In the final stage, the best solution found will be retrieved.

To understand how this study uses GOW to optimize TSP problem, as in figure 5.a. It shows a TSP problem which is an undirected graph of four cities and six edges, each edge has its own Travelling cost, which will be used in this example to represents the possible solutions. Based on TABLE 1, each element in GWO represent what its means in GWO-TSP, for instance, Gray wolf population in GWO represented in GWO-TSP as the candidate solutions of TSP. By adapting the elements in GWO to our proposed algorithm, The example that was displayed in Figure 5.a this will become as shown in Figure 5.b.

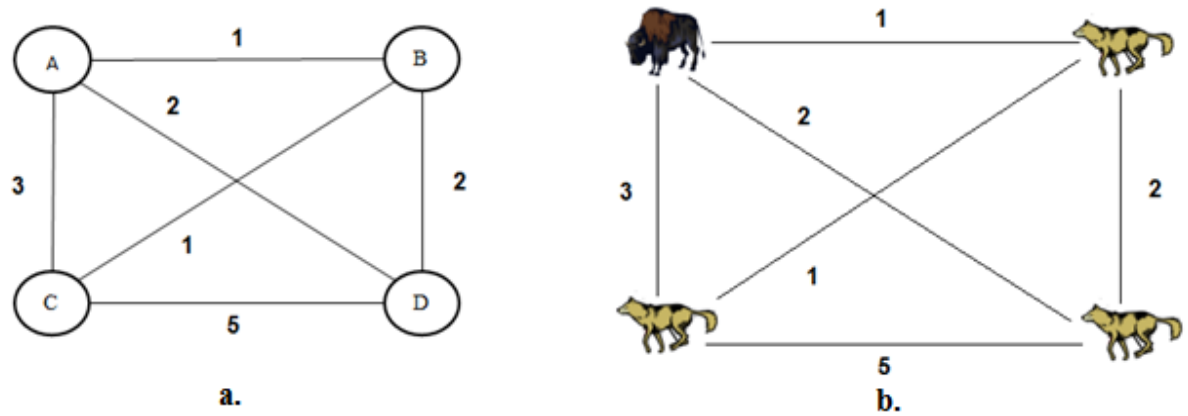

Figure 5. Solving TSP problem using GWO. a) TSP example. b) TSP-GWO example

3.2 Parallel ALGORITHM: "GWO-TSP"

The large number of possible solutions, even with a GWO, opens the possibility of parallelizing the solution for the TSP problem, in which the main objective of this paper is to study the possibility of parallelizing the GWO to solve the TSP problem. In our proposed model, we have created the parallel version of GWO (PGWO) that can be efficiently executed on the Hypercube interconnection network.

For developing PGWO, original TSP map must be divided to achieve load balancing among all processors. In this study, We divide the problem of TSP into sub-problems by creating districts from the original map (dataset) during the partition operation, then we apply the GWO-TSP steps on each part of the district generated by the partition operation. The partition operation consists of two steps:

- Find the highest and lowest values edges of the map.
- Divide the map into multiple districts by subtracting the highest values from the lowest values then dividing the difference by the desired value. Figures 6.1 - 6.3 show how districts are created from a original map.

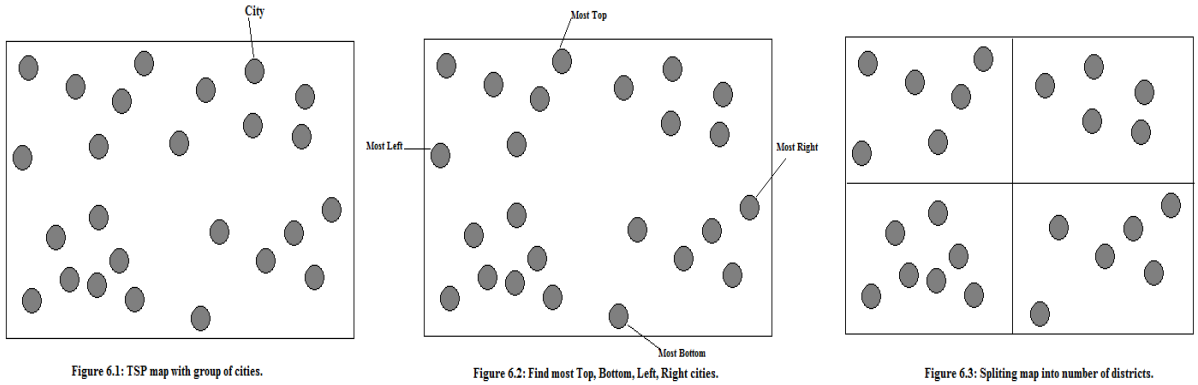

Figure 6. Splitting the TSP map into multiple districts

The number of districts depends on the number of processors we use, for example, if the number of processors is 16, then the number of districts must be at least equal16. The equation 7 was used to find the number of districts.

$$D = N \quad (7)$$

Where D is the number of districts, N is the number of processors. Figure 7 presents the districts algorithm.

Algorithm 2 : Create districts algorithm

Input:

M: map.

N: Number of nodes used to solve TSP using current map M.

Output: D: Array of districts.

```
1. initial_cities = [ ];
2. MostTopCity, MostBottomCity, MostRigthCity, MostLeftCity.
3. for each city in cities map {
5.      if (city in x-axis is the highest value amongs all cities store it MostTopCity)
6.      if (city in x-axis is the lowest value amongs all cities store it in MostBottomCity)
7.      if (city in y-axis is the lowest value amongs all cities store it in MostRigthCity)
8.      if (city in y-axis is the highest value amongs all cities store it in MostLeftCity)
9.  End for}
10. D(Border,CitiesIn) = [ ] where Border: area boundaries, CitiesIn: array of cities in the Border.
11. for each (city in initial_cities) {
12.     for (i = MostTopCity to MostBottomCity) { // increment:= MostBottomCity – MostBottomCity / N)
13.     for (e = MostLeftCity to MostRigthCity) { // increment:= MostRigthCity – MostLeftCity / N)
14. if city x-axis and y-axis between i and y then add city to D with Border i+e .
15. }}
16. return D.
```

Figure 7. Create districts algorithm

As in Figure 7, District algorithm firstly read the city map and add cities on the x-axis and y-axis. Also, find the Border of the map by storing the Most Top City and theMost Bottom City values for x-axis and y-axis. Lines 12 – 15, used to group the cities in districts depends on their location in the map using cities stored in first loop.

As discussed in the introductory section, the hypercube interconnection network contains 2d nodes, where d is the number of hypercube dimension. we can find the number of dimensions of hypercube from the number of nodes using equation 8, where N is the number of nodes.

$$d = \log2(N) \quad (8)$$

To distribute districts over hypercube, label all nodes in the hypercube interconnection network, where the number of bits is required to label all nodes equal to the dimension value. For example, if the dimension is equal to 3, label all the nodes in three bits (000, 001, 010, ..., 111), which provides eight numbers of nodes. To distribute districts through the hypercube network, find d using equation 8, then check all the nodes. If d = 3, the loop will be executed three times, each time the districts are divided in half and send half to the next node along the dimension (d - 1). Each node, the same operation is performed until it reaches the dimension of d = 0 as an indicator to be stopped (see Figures 9).

Figure 8 shows the routing algorithm for hypercube to partition and distribute the data all over nodes.

Algorithm 3: Hypercube routing algorithm

Input:

D: Array of districts.

doh: Dimension of hypercube.

Output: District partition and distrubtion over hypercube.

```
1.for (i:=0 to doh) {
2. nextNode = D / 2.
3. SOA(D, 0, part).
4. D = UpdateD(D, part+1, |D|)
5. }
```

Figure 8. Hypercube routing algorithm

In Figure 8, lines from 1 to 5 a loop equal to the value of hypercube dimension, keeps splitting the districts over all nodes in the hypercube. Line 2 finds the desired value for next node, in Line 3, SOA function responsible to split the original array of districts D, then send that desired value to the next node. Finally, UpdateD function responsible to update the districts array value at Line 4.

Figure 9 shows the communication mechanism for d= 2.

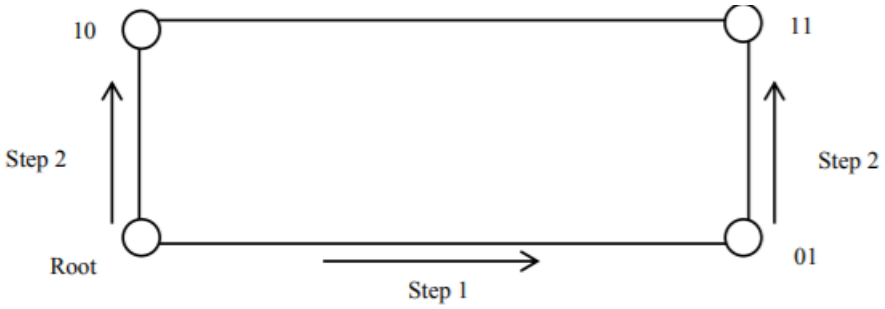

Figure 9. Communication mechanism over hypercube

The computation distribution is provided by distribution the maximum population size from each district in order to reduce the computation time. As equation 9:

$$\text{population size} = a * \text{districtSize} \quad (9)$$

where a is a factor between [0-1], districtSize equals the number of cities in district and the population size is the

maximum number of candidate solutions.

The data combination is performed by overturning the order of the steps in the distribution phase. After each node completes all the planned iterations, sends the best solution to the node that contains a complete route (shorter path) and links the solution with the solution that it provides, in the end, the master node will contain the final solution (full path). Figures 4 present the pseudocode for the proposed parallel algorithm "GWO_TSP".

Algorithm 4:PGWO-TSP on hypercube interconniction network

Input: TSP problem

Output: Shortest tour among all cities

Phase 1: Load Balancing Phase

1. Root nood applies the Create districts algorithm (Fig. 7) to balance the number of cities among all processors in Hypercube;

Phase 2: Data Distribution Phase

2. apply Hypercube routing algorithm(Fig. 8) to partition and distribute the data all over Hypercube nodes

Phase 3: Local Repetitive.

3 for all processors in Hypercube, do in parallel

4. Apply sequential Grey wolf algorithm(Fig. 4) .

Phase 4: Data Combining Phase

5. for each pair of processors differs only in the dth bit position do in parallel

6. sends the best solution

7. Root processor combines the collected group route and finds the minimum route and its cost.

Figure 10. PGWO-TSP algorithm on the hypercube interconnection network

4. Analytical Evaluation

This section provides the analytical evaluation of the sequential GWO-TSP algorithm in term of time complexity and the parallel version on Hypercube interconnection network in terms of parallel time complexity, speedup, efficiency, and cost.

4.1 Analytical Evaluation of Sequential GWO-TSP Algorithm

As it is described in before, GWO-TSP consists of multiple steps, where initially create initial population then create new generation by Calculating the destination between cities.

All the terms that precede (see line 6) are constants. As shown in Figure 4 (lines 1-5), the outer while loop is expected to run until reach the population size where each population must contain a Full solution, as shown in Figure 4 (lines 2-8). In the worst case, the number of population is equal to the number of cities which means O (n). Inside the main loop, another loop runs equal to population size as shown in Figure 4 (lines 8-18), where in each iteration, three cities are picked and updated the population O(n). The function in line 9 which used to Calculate the destination of all wolfs is require O(n) while line 10 to 16 are constants. In line 17, the time complexity for the function of Calculate the fitness value for each Population is O(n). Variables in lines 17 to 19 are constants.

The total time complexity of sequential GWOTSP is shown in Equation 10 where T is the time complexity, N is the number of population and C is constants:

$$T(N) = O(C + N * (N + C + N + C + N) + C \quad (10)$$

Equation 10 can be reduced to Equation 11

$$T(N) = O(2C + 3N^2 + 2NC) \quad (11)$$

The largest term of equation 3 is n^2, Thus, the final time complexity will be O (n^2).

4.2 Analytical Evaluation of GWO-TSP Algorithm on Hypercube

This section provides the analytical evaluation of GWO-TSP algorithm on Hypercube interconnection network in terms of parallel time complexity, speedup, and efficiency.

4.2.1 Parallel Time Complexity

The parallel execution time is equal to the total of computation time plus the total of communication time. The time required to apply the sequential GWO-TSP on a set of cities represents the computation time and the communication time is equal to the number of communication steps required in both phases, distribution and combination.

The analytical evaluation of the complexity of the parallel execution time for all phases of the GWO-TSP algorithm over hypercube interconnection network is demonstrated by tracing the algorithm in Fig. 10, as shown in Table 2.

The overall complexity of the parallel execution time of phases 1-4 is shown in equation (12) where T is the complexity of time, N is the number of cities, P is the number of processors and d is the size of the hypercube.

$$T(N, p) = O(n^2) + O(d) + o(n/p * n^2) + O(d + n) \quad (12)$$

Equation (12) can be reduced to Eq. (13).

$$T(N, p) = O(n^2 + d + n^3/p) \quad (13)$$

Table 2. All phases of GWO-TSP algorithm on Hypercube interconnection network.

Phase 1 (Load balancing phase)	Root processor executes the Create districts algorithm in fig.7. the execution will split the map to a number of districts, from line 1- 3 the time complexity equal to O(5C+1), from line 4-10 the time complexity equal to O(n), finally, the time complexity for grouping cities in districts takes $O(C \times N^2)$, where C is the number of cities in input data and N is the array of districts. The total time complexity is $O(C+n+C\times n^2) \approx O(n^2)$
Phase 2 (Data distribution phase)	In the hypercube interconnection network, the number of steps necessary to distribute the data through all hypercube is required d steps, where d is the size of the hypercube, which is log P, the general execution time is O (d)
Phase 3: Local Repetitive.	All processors run the sequential GWO-TSP on each district. This will require $N/P \times N^2$ time complexity, where N is the number of cities, P is the number of processors and N^2 is the run time complexity of the sequential GWO-TSP.
Phase 4: Data Combining Phase	All processors will send the solution to the root node, this will be performed in log P steps which equals d and root processor required O(n) as time complexity to combining all solutions. The total time complexity is O(d + n)

4.2.2 Speedup

Speedup is an important measure for a parallel algorithm, used to calculate the relation of the sequential computation time and the parallel time as equation 14:

$$S = TS/TP \quad (14)$$

where TS is the time required by the sequential algorithm and TP is the time required by the parallel algorithm.

The sequential time complexity for GWO-TSP is $O(n^2)$ and the parallel version is required which is illustrated in Eq. (3), the speedup of GWO-TSP over hypercube is shown in equation 15:

$$S = n^2 * p / n^2 + d + n^3/p \quad (15)$$

4.1.3 Efficiency

One of the important factors to measure parallel performance is parallel efficiency which measures how much the processors being utilized in the interconnection networks. It is equal the ratio between speedup and the number of processors as equation 16:

$$E = S / p \quad (16)$$

Where E is the Efficiency, S is Speedup as equation 6 and p is the number of processors.

Thus, the efficiency of the GWO-TSP algorithm on Hypercube is shown in equation (17).

$$E = n^2 / \ n^2 + d + n^3/p \quad (17)$$

5. Simulation Results

For our experiments, we used a computer with Intel Core i5-3317U CPU 1.70GHz with 8 GB of RAM. The simulation for GWO, CRO, and GA has implemented in Java JDK 8 programming language. The algorithms were tested by 6 different size TSP problems taken from the World TSP (TSP website, 2009); XQF131, XQG237, PMA343, PKA379, PBL395, and PBN423. The parameters are fixed on follows: number of wolves equals the number of cities in each TSP instance and the maximum number of solutions equals 70% from the number of cities in the dataset, this value is selected to make the algorithm more scalable and to reduce both of computation time and the required space. For fairness, the same specifications and same stopping criteria are used in our simulations for all algorithms.

5.1 Sequential Results

Since GWO, CRO, and GA are meta-heuristic mechanisms, the results obtained in different executions could be different. Because that we repeat the simulation 25 times and record the results as shown in Table 3.

Table 3. The experimental results of sequential GWO, CRO, and GA in terms of fitness value, quality of solution and the execution.

		GWO				CRO				GA			
Instance name	Optimal	Best optimal	Mean optimal	Error rate(%)	Time(Sec)	Best optimal	Mean optimal	Error rate(%)	Time(Sec)	Best optimal	Mean optimal	Error rate(%)	Time(Sec)
XQF131	564	569	575	0.886	22.254	573	580	1.595	17.784	574	591	1.773	18.854
XQG237	1019	1030	1033	1.079	54.985	1033	1037	1.668	44.624	1036	1038	1.766	42.241
PMA343	1368	1385	1387	1.242	68.854	1385	1398	2.119	52.325	1400	1399	2.339	56.745
PKA379	1332	1347	1349	1.126	82.325	1347	1360	1.726	74.365	1358	1361	1.951	72.251
PBL395	1281	1296	1300	1.170	95.254	1296	1312	2.107	83.521	1311	1315	2.341	84.214
PBN423	1365	1383	1386	1.318	112.542	1383	1395	1.831	88.124	1398	1405	2.417	92.248

In Table 3, the first column shows the name of the instance (the numbers in the names indicate the nodes of each instance). The second column shows the best-known solution for each instance taken from the World TSP [42]. For each algorithm there are four columns, the best column shows the best fitness value of the best execution. The mean column shows the average quality of 25 executions of the algorithm. The error rate column shows the fitness function (minimum) of the best individual provided algorithm and the optimal TSPLIB. The error is calculated as in equation 18, finally, the time column shows the time needed to execute the entire program in seconds.

$$Error = \left(\frac{BestSolution - OptimalSolution}{OptimalSolution}\right) * 100 \quad (18)$$

Where Error is the relative value of difference from the optimum tour, Best Solution is the tour length obtained by the experiment and Optimal Solution is the tour length of optimum solution.

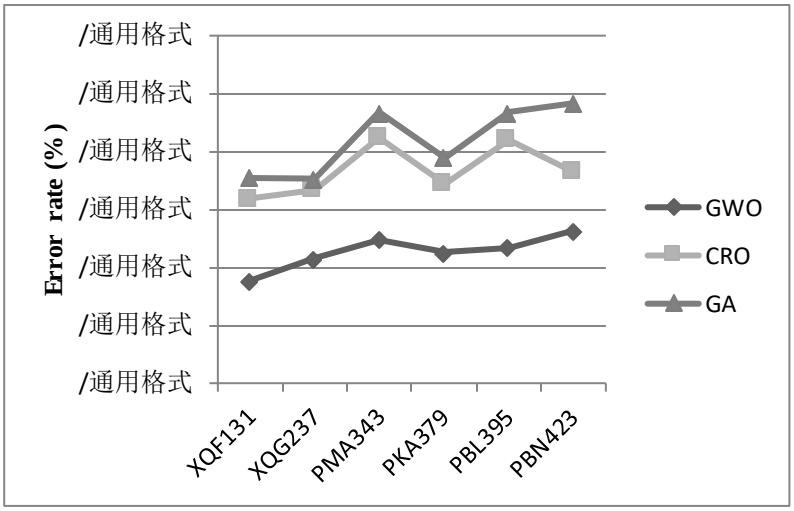

Figure 11. Quality of solutions for GWO, CRO, and GA

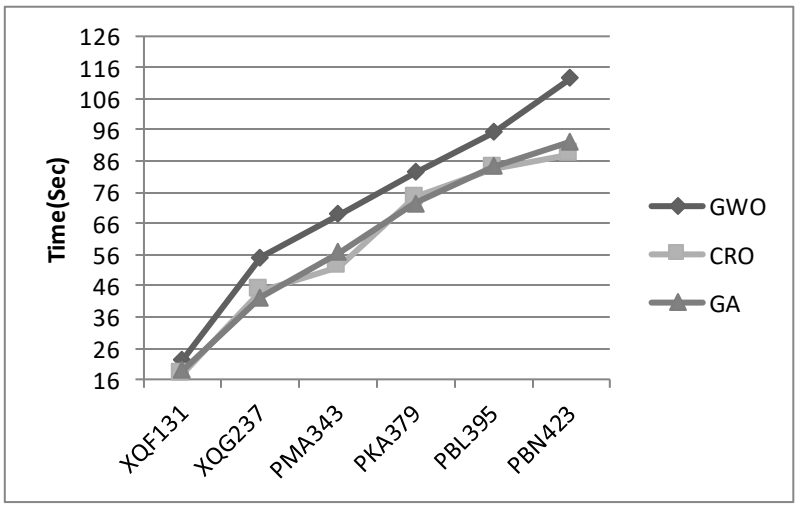

Figure 12. Runtime chart for GWO, CRO and GA

From Figure 11, it is clear that GOW always gives the highest solution quality (minimum traveling cost) for all TSP instances tested. This is followed by CRO and GA algorithms. However, the quality of solution reduces as the size of instance increase with an increase in execution time for all algorithms.

From Figure 12, we can observe that the runtime for all algorithms is almost the same with a slightly different but the best runtime comes from CRO for some data instance such like PMA343 and PBN423. Also, it is clear that the experimental and theoretical time converge.

5.2 Parallel Results

For the Parallel results, we used a big TSP problem (IRW2802) taken from the World TSP, which contains 8423 cities with 5533 lengths as the known optimal solution.

In Table 2, the first column shows the number of processors, the time column shows the parallel time (best time) it takes to run the entire program in seconds on nodes. The third column shows the Speedup a ratio of the computation time of the sequential time and the parallel time as equation 14. The error rate column shows the error value of the fitness function (minimum) of the best individual as equation 18.

Table 4. Parallel time, Speedup and Error rate for PGWO, PCRO, and PGA.

Num of P	PGWO			PCRO			PGA		
	Time(Sec)	Speedup	Error rate	Time(Sec)	Speedup	Error rate	Time(Sec)	Speedup	Error rate
1	223.42	N/A	5.852	191.58	N/A	8.954	182.54	N/A	9.872
32	10.183	21.942	8.954	9.063	21.138	12.625	10.685	17.083	13.521
64	5.486	40.725	9.715	4.901	39.012	13.851	7.661	23.827	14.841
128	3.614	66.812	11.521	2.508	76.387	14.625	3.704	49.281	16.281
256	**2.335**	95.683	11.842	**2.081**	92.061	15.212	2.898	63.162	15.852
512	**2.298**	97.223	12.945	**2.053**	93.317	17.101	2.487	73.397	18.934
1024	2.817	79.311	14.985	2.963	64.657	19.254	3.603	50.663	21.927

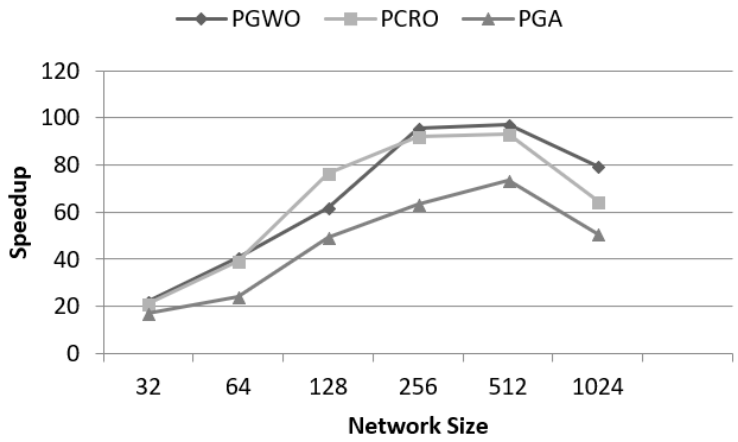

Figure 13. Speedup for PGWO, PCRO, and PGA

Figure 13 shows that speed-up increases in all the algorithms apart from the increase in the number of nodes used until a certain number of nodes begins to decrease. We can see that the speedup is almost linear for PGWO when it uses 32 to 256, while it is sub-linear when there are 512 processors and then begins to decrease. This is due to the communication overload in the 512 and 1024 processors scenario, which is much more than that in 256 processors or less. PCRO got better Speedup compared to PCRO and PGA.

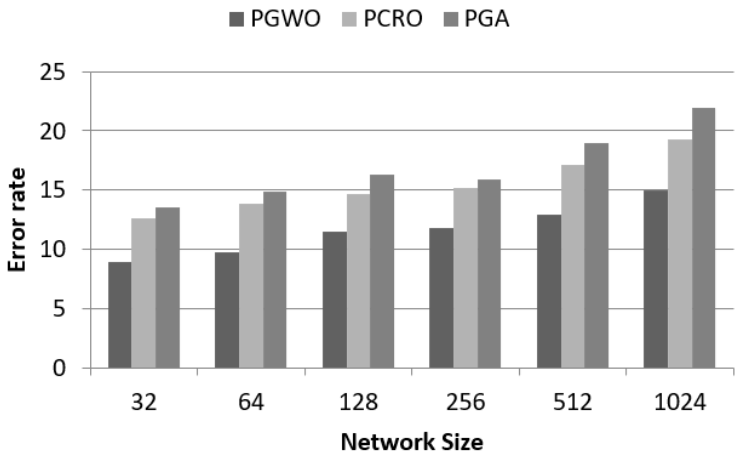

Figure 14. Quality of solutions for PGOW,PCRO, and PGA

Comparing the error rate of fitness values from Figure 14, we find that in most cases the quality of the solutions generated by PGWO is better than PCRO and PGA. This is because in GWO, populations (Solutions) are built from scratch, where the CRO and GA populations are created randomly. The results obtained by CRO are better

than GA, because there are four types of reactions that improve the solutions more than the GA. Moreover, with the increase in the number of processors, the error rate has increased. This is mainly due to the division data and the number of iterations on the processors.

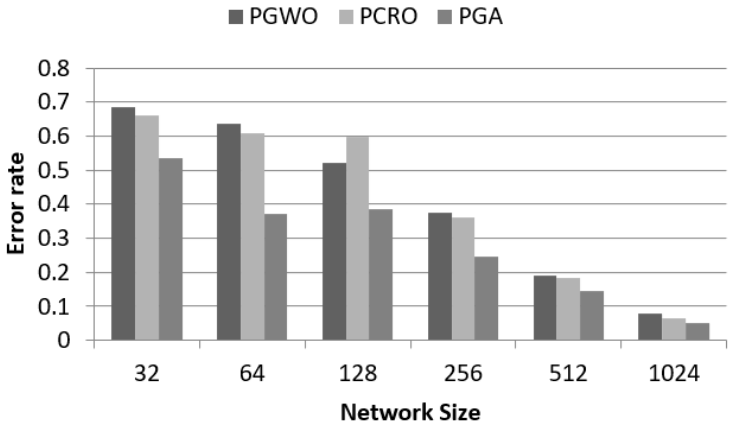

Figure 15. Relative efficiency for GWO, CRO and GA

Figure 15 shows the relative efficiency for PGOW, PCRO, and PGA. As it shows that efficiency decreases in all the algorithms aside with increasing the number of nodes used, since the amount of data shared for each node decreases, so the difference between communication time and computation time is reduced, which affects the speedup. linear with the increase in the number of nodes used, which in turn affects the efficiency, but we can see that the efficiency of GWO is better than PCRO, PGA, where the worst efficiency is obtained from PGA.

6. Conclusions and Future Work

This study introduces a parallel model of GWO algorithm to solve the TSP problem called "GWO-TSP" in a hypercube interconnection network. The analytical evaluation of the sequential and parallel is presented. GWO is compared first with the sequential GWO and then with PCRO and PGA. The simulations are performed by TSP instances of different sizes. To be honest, the same stopping criteria are used in our simulations for all algorithms. The results show that GWO for TSP can improve the fitness value and reduce the computation time with a higher speed-up and better parallel efficiency.

For future work, we intend to compare GWO-TSP with other meta-heuristic algorithms, design and test a deferential interconnection network.

References

Al-Shaikh, A., Khattab, H., Sharieh, A., & Sleit, A. (2016). Resource Utilization in Cloud Computing as an Optimization Problem. *International Journal of Advanced Computer Science and Applications (IJACSA), 7*(6), 336-342, 2016.

Anjaneyulu, G. S. G. N., Dashora, R., Vijayabarathi, A., & Rathore, B. S. (2014). Improving the performance of approximation algorithm to solve travelling salesman problem using parallel algorithm. *International Journal of Scientific Engineering and Technology, 3*(4), 334-337.

Applegate D. L., Bixby R. E., Chvátal V., Cook W. J. (2007). The Traveling Salesman Problem. A Computational Study (2007) (*Princeton University Press*, Princeton, NJ).

Barham, R., & Aljarah, I. (2017, October). Link Prediction Based on Whale Optimization Algorithm. In New Trends in Computing Sciences (ICTCS), 2017 International Conference on (pp. 55-60). IEEE.

Barham, Reham & Sharieh, Ahmad & Sleit, Azzam. (2016). Chemical Reaction Optimization for Max Flow Problem. *International Journal of Advanced Computer Science and Applications, 7*, 189-196. https://doi.org/10.14569/IJACSA.2016.070826.

Barney, B. (2010). Introduction to parallel computing. *Lawrence Livermore National Laboratory, 6*(13), 10.

Bastos-Filho, C., Lima, N., Lins, A., Nascimento, A., & Lima, M. (2008). A Novel Search Algorithm based on Fish School Behavior. *Proceedings of the IEEE International Conference on Systems Man and Cybernetics*, pp. 682-687.

Bhardwaj, G., & Pandey, M. (2014). Parallel implementation of travelling salesman problem using ant colony optimization. *International Journal of Computer Applications Technology and Research, 3*(6), 385-389.

Bhuyan, L. N., & Agrawal, D. P. (1984). Generalized hypercube and hyperbus structures for a computer network. *IEEE Transactions on computers*, (4), 323-333.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptualcomparision. *ACM Comput*. Surv., *35*, 268-308.

Cathleen, L. (2011). Inside a NASA Production Supercomputing Center. *Concept to Reality magazines*, Summer/Fall issue

Chekuri, C., & Quanrud, K. (2017). Approximating the Held-Karp Bound for Metric TSP in Nearly-Linear Time. *arXiv preprint* arXiv:1702.04307.

D'Angelo, G. (2011). Parallel and distributed simulation from many cores to the public cloud. *In High Performance Computing and Simulation (HPCS)*, 2011 International Conference on (pp. 14-23). IEEE.

Gutin, G., Yeo, A., & Zverovich, A. (2002). Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. *Discrete Applied Mathematics, 117*, 81-86.

Hernández, H., Rodríguez, R., & Salazar, J. (2016). A hybrid heuristic approach for the multi-commodity pickup and delivery traveling salesman problem. *European Journal of Operational Research* - ELSEVIER, *251*, pp. 44-52.

Hwaitat, A. K. A., Shaheen, A., Adhim, K., Arkebat, E. N., & Hwiatat, A. A. A. (2018). Computer Hardware Components Ontology. Modern Applied Science, 12(3), 35.

Ingber, L., & Rosen, B. (1992), Genetic Algorithms and Very Fast Simulated Reannealing: A Comparison. *J. of Mathematical and Computer Modeling, 16*(11), 87-100.

James, J. Q., & Li, V. O. (2015). A social spider algorithm for global optimization. *Applied Soft Computing, 30*, 614-627.

Kaempfer, Y., & Wolf, L. (2018). Learning the Multiple Traveling Salesmen Problem with Permutation *Invariant Pooling Networks*. arXiv preprint arXiv:1803.09621.

Kan, A. R., & Shmoys, D. B. (1985). The traveling salesman problem: A guided tour of combinatorial optimization (Vol. 3, pp. 1-463). E. L. *Lawler, & J. K. Lenstra (*Eds.). New York: Wiley.

Karla, L., Hoffman, Manfred, P., & Giovanni, R. (2016). Traveling Salesman Problem. *Encyclopedia of Operations Research and Management Science*, Springer, 1573-1578.

Kiasari, A., & Sarbazi-Azad, H. (2008), Analytic performance comparison of hypercubes and star graphs with implementation constraints*", Journal of Computer and System Sciences*, *74*(6), 1000-1012.

Komaki, G. M., & Kayvanfar. (2015). Grey Wolf Optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time. *Journal of Computational Science, 8*(8), 109-20.

Korayem, L., Khorsid, M., & Kassem, S. S. (2015). Using grey wolf algorithm to solve the capacitated vehicle routing problem. In IOP *Conference Series: Materials Science and Engineering, 83*(1), 012-014. IOP Publishing.

Kordon, A. K. (2010). Swarm intelligence: The benefits of swarms. In Applying Computational Intelligence (pp. 145-174). *Springer Berlin Heidelberg*.

Kumbharana, S. N., & Pandey, G. M. (2013). Solving travelling salesman problem using firefly algorithm. *International Journal for Research in science & advanced Technologies*, 2(2), 53-57.

Lam, F., & Newman, A. (2008). Traveling salesman path problems. *Mathematical Programming, 113*(1), 39-59.

Matai, R., Singh, S., & Lal, M. (2010). Traveling salesman problem: An overview of applications, formulations, and solution approaches. *In D. Davendra (Ed.), Traveling Salesman Problem, Theory and Applications*. InTech.

Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software, 95*, 51-67.

Mohan, A., & Remya, G. (2014). A Parallel Implementation of Ant Colony Optimization for TSP based on MapReduce Framework. *International Journal of Computer Applications, 88*(8), 9-12.

Poli R, Kennedy J, Blackwell T (2007). Particle swarm optimization. An overview. Swarm Intelligence, pp. 33-57.

Raja, H. Q. (2015). Self-Sufficiency of an Autonomous Reconfigurable Modular Robotic Organism. *ebook* , Springer .

Reynolds, C. W. (1987). Flocks herds and schools: A distributed behavioral model. *Computer Graphics*, *21*(4), 25-34.

Ross, P. M., & Corne, D. (1995). Comparing genetic algorithms, stochastic hillclimbing and simulated annealing. In T.C. Fogarty (ed), *Evolutionary computing*, Springer-Verlag, 94-102 (1995).

Shaheen, A., Al-Sayyed, R., & Sleit, A. (2017). Improving visual analyses and communications of ontology by dynamic tree (case study: computer system). *International Journal of Advanced and Applied Sciences, 4*(5), 62-66.

Shaheen, A., Sleit, A., & Al-Sharaeh, S. (2018). An improved chemical reaction optimization algorithm for solving traveling salesman problem, 37-42. https://doi.org/10.1109/IACS.2018.8355438.

Shaheen, A., Sleit, A., & AlSharaeh, S. (2018). Chemical Reaction Optimization for Traveling Salesman Problem Over a Hypercube Interconnection Network, 432- 442. https://doi.org/10.1007/978-3-319-91192-2_43

Shaheen, Ameen & Sleit, Azzam. (2016). Comparing between different approaches to solve the 0/1 Knapsack problem. *International Journal of Network Security*, *16,* 1-10.

Sleit, A. (2008). On using B+-tree for efficient processing for the boundary neighborhood problem. *WSEAS Transactions on Systems, 11*(11), 711-20.

Sleit, A. S., Imad, J., & Rahmeh (2008). *Approximating images using minimum bounding rectangles.* ICADIWT 2008, 394-396. https://doi.org/10.1109/ICADIWT.2008.4664379

Sleit, A., Al-Akhras, M., Juma, I., & Alian, M. (2009). Applying ordinal association rules for cleansing data with missing values. *Journal of American Science, 5*(3), 52-62.

Song, H. M., Sulaiman, M. H., & Mohamed, M. R. (2014). An application of Grey wolf optimizer for solving combined economic emission dispatch problems. *International Review on Modelling and Simulations, 7*(5), 838-44.

Song, X., Tang, L., Zhao, S., Zhang, X., Li, L., Huang, J., & Cai, W. (2015). Grey Wolf Optimizer for parameter estimation in surface waves. *Soil Dynamics and Earthquake Engineering, 75*, 147-157.

Sun, J., Wang, Y., Li, J., & Gao, K. (2011). Hybrid algorithm based on chemical reaction optimization and Lin-Kernighan local search for the traveling salesman problem.

Sun, X. H., & Ni, L. M. (1990, November). Another view on parallel speedup. In Proceedings of the 1990 *ACM/IEEE conference on Supercomputing* (pp. 324-333). IEEE Computer Society Press.

Sun, Y., Albert, Y. S., Lam, V., Li, O. K., Xu, J., James, & Yu, J. Q. (2012). Chemical Reaction Optimization for the optimal power flow problem. Evolutionary Computation (CEC) 2012 *IEEE Congress* on, 1-8, 2012.

TSP website. (2009). *A collection of worldwide benchmark datasets.* Retrieved December 15, 2017, from http://www.math.uwaterloo.ca/tsp/world/countries.html

Vukmirović, S., & Pupavac, D. (2013). The Travelling Salesman Problem in the Function of Transport Network Optimalization, *Osijek: InterdisciplinaryManagemet Research IX, University in Osijek*, Fakulty of Economics.

Wong, L, M., Low, H., & Chong, C. (2008). A bee colony optimization algorithm for traveling salesman problem. *Proceedings of Second Asia International Conference on Modelling & Simulation* (AMS 2008), pp. 818-823.

Worboys, M. (2005). Event-oriented approaches to geographic phenomena. *International Journal of Geographical Information Science, 19*(1), 1-28.

Xu, K., Jiang, M. Y., & Yuan, D. F. (2013). Parallel artificial bee colony algorithm for the traveling salesman problem. *In Advanced Materials Research, 756*, 3254-3259.

Yang, Jinhui, et al. (2008). An ant colony optimization method for generalized TSP problem. *Progress in Natural Science*, *11*(2008), 1417-1422.

Yassien, E., Masadeh, R., Alzaqebah, A., & Shaheen, A. (2017). Grey Wolf Optimization Applied to the 0/1 Knapsack Problem. *International Journal of Computer Applications, 169*(5).

Zhan, F., & Noon, C. (1996). Shortest Path Algorithms: An Evaluation Using Real Road Networks. *Transportation*

Science.

Zwietering, M. H., Jongenburger, I., Rombouts, F. M., & Van 'T Riet, K. (1990). Modeling of the bacterial growth curve. *Appl Environ Microbiol, 56*(6), 1875-81.