Optimization of Capacitated Vehicle Routing Problem Using Artificial Bee Colony Algorithm

Nikhil Haresh Keswaney
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14586
nhk8621@cs.rit.edu

***Abstract*—Capacitated Vehicle Routing Problem(CVRP) is a combinatorial problem. CVRP is a NP-Hard problem and the goal of the problem is that for a given set of customers with a set of demands and a fleet of vehicles with their respective capacities, find the most optimal route for all the vehicles such that the demands of all the customers are satisfied maintaining the capacity constraint of all the vehicles [1]. This paper presents multiple approaches for solving the capacitated vehicle routing problem. It first explains an exact approach which combines the subset partition problem and the traveling salesman problem, then an approximate algorithm which is the Artificial Bee colony(ABC) algorithm, the paper also presents a parallel architecture of the Artificial Bee Colony algorithm. Finally, a comparative study comparing and contrasting the runtime of all the approaches on datasets of various sizes is presented.**

***Index Terms*—CVRP; ABC; Parallel;**

I. INTRODUCTION

The vehicle routing problem (VRP) is a combinatorial problem, which has a wide spread of applications like transportation, e-commerce etc. It appeared first in 1959 in a paper written by George Dantzig and John Ramser[2]. The goal of Vehicle Routing Problem(VRP) is to serve N number of customers using K number of trucks minimizing a cost function. The cost function can be anything like the distance travelled, fuel utilization, number of left turns etc. There are many variants of the vehicle routing problem such as vehicle routing problem with time windows, dynamic vehicle routing problem, capacitated vehicle routing problem etc[3]. This paper tackles the capacitated vehicle routing problems(CVRP). There has been extensive research carried out on VRP due to the important applications of the problem and there are multiple ways to solve the VRP which include:

1) Exact algorithms that always return the optimal value.
2) Approximate algorithm which doesn't necessarily return the optimal value but return near optimal results.

The problem with using the exact algorithm is the computational complexity. Doing a brute force algorithm on a processor having ∼1GHz of clock cycle and additionally computing the optimal path for ∼50 customers takes ∼9.6 $\times 10^{47}$ years [4], which is not affordable. However, there are certain fast exact algorithms like, branch and bound, cutting plane which reduces the the computation time, but they also have a very high computational complexity especially in large VRP problems. Thus, approximate algorithms have become popular in recent times [5]. Even though the approximate algorithms don't return the most optimal solution, they complete computation in a reasonable amount of time. Some approximate algorithms that have been used to solve CVRP are tabu search, ant colony optimization, genetic algorithm, assembly-based memetic algorithm, simulated annealing etc [6].

A. Problem statement

The capacitated vehicle routing problem is a combinatorial problem which states that there are N customers to whom goods have to be delivered using a fleet of vehicles utilizing the most optimal route and each vehicle in the fleet needs to start and end at the same depot. The state space can be represented using a graph.

$$G \equiv \{V, E\}$$

where V is the number of vertices and E is the number of edges in the graph G. There are N customers that need to be served and can be represented by $S \equiv \{1, 2, 3 \dots N\}$
where S is the set of all the customers. Additionally, since all vehicles start and originate from the same depot assuming it is 0, we get:

$$V \equiv \{0\} \cup S$$

The vehicles are denoted as $K \equiv \{1, 2 \dots k\}$ and the vehicle's capacity will be denoted by q_k which will be the same for all the vehicles.
Each customer has a demand and the demand of i^{th} customer is denoted by d_i. Each edge i.e. the path for going from customer i to customer j is represented by c_{ij}. Additionally, we have to satisfy certain other constraints to solve the problem like:

1) Finding the most optimal route minimizing the cost of travel.
2) All vehicles will start from the depot and end at the depot.
3) One customer will be served once by only one vehicle.
4) The vehicles the visit the customer will also serve the customer.
5) The sum of the demands for the customer that one vehicle visits shouldn't exceed the capacity of the vehicle.

II. Dataset

The dataset used for this paper is the capacitated vehicle routing problem (CVRP) dataset from BranchAndCut.org [7]. It has ∼107 files for CVRP problem. The dataset has problem sets of various sizes. The file of the problem set is divided into 2 parts specification part and the data part. [7].

- Specification part: This part indicates how many nodes are there to be served, the number of trucks used to serve these node, the distance measure that needs to be used (Euclidean, Manhattan, L_∞).
- Data part: This part includes the co-ordinates of the nodes, the demand for each node, the depot.

Each of these files is represented by a .vrp file and each file has a co-responding .opt which has the optimal route for each truck and the optimal total cost to serve all nodes. The range of files varies from 10 nodes to 262 nodes.

III. Exact Algorithm

The exact algorithm for the problem is divided into two parts. The first part will find all possible ways to divide the customers into K subsets where K is equal to the number of vehicles in the fleet. The second part would be to find the best answer using the traveling salesman(TSP) algorithm. The first part returns all the possible ways that the customers can be divided and served by K vehicles and then the best amongst these is chosen by using the traveling salesman algorithm.

A. Subset finder

We have a set of customers or nodes which is denoted by $S \equiv \{1, 2, 3 \dots N\}$, we have to partition S into P in such a way that

$$A \cap B = \emptyset$$

$$A, B \in P$$

The approach used was a dynamic programming approach. The number of partitions the set will be divided into will be the number of vehicles that will be delivering the items. Assuming there are K vehicles with capacity q_k that will be delivering the goods. The recurrence relation for the problem will be as follows [8].

$$U(n, k) = k \times U(n-1, k) + U(n-1, k-1)$$

where $U(n, k)$ denotes all the partitions that are possible while dividing the set n into k sets The above recurrence relation should follow the following constraint. Let C be a candidate subsets in all the possible subsets of $U(n, k)$ and J be the path for a vehicle in C and each customer in J will have a demand d_i then $\sum d_i \leq q_k$.

B. Traveling salesman problem

The traveling salesman problem states that given a set of vertices, we need to minimize the cost for the route such that all the vertices are visited only once. The approach used in this paper for implementing the traveling salesman problem is backtracking. We are assuming that there are N customers and the cost for going from customer i to customer j is c_{ij}. Assuming $T(S, i)$ represents the minimum cost path to visit all the nodes in S starting from the depot and ending at i. So the recurrence relation would be as follows.

$$T(S, i) = \forall i \in S \ \min\{T(S - \{i\}, j) + c_{ij}\}$$

The travelling salesman algorithm will be applied on each path J of a candidate subset C of all the possible subsets of $U(n, k)$ which is defined in the previous section. So for all J in C we apply the travelling salesman and get the cost. We find the total cost for C which is represented as follows.

$$\sum_J T(J, i)$$

. For all C in $U(n, k)$ we find the cost and we find the C with the minimum cost which will be the optimal route for all trucks.

IV. Artificial Bee Colony Algorithm

A. Description

Artificial bee colony algorithm is based on the foraging behaviour of honey bees. It is a swarm intelligence algorithm which is based on three main components namely decentralization, self-organization and collective behaviour[6]. Since there is not one central entity that controls the whole algorithm, and each individual is responsible for its own action, self-organization means that the set of rules for each individual will lead to a neat organization of the entire swarm and collective behaviour is the task of finding the best food source which is done collectively.

The artificial bee colony algorithm is made of three types of bees which are as follows:

1) Scout bees: The job of the scout bees is to go and find new potential neighbourhood for food sources.
2) Employed bees: Approximately 50 % of the swarm bees are employed bees. The job of the employed bees is to explore the neighbourhoods discovered by scout bees.
3) Onlooker Bees: The onlooker bees monitor the work that is performed by the employed bees and exploit food sources in the neighbourhood of the employed bee that has the richest food source.

The food source mentioned in the above introduction are treated as the solution of the problem. The solution with more nectar is considered as a better solution, the nectar of a solution is the fitness value of the solution.

B. Working

In the beginning, the scout bees go out and find possible food sources i.e. neighbourhoods where food with high nectar can be available, then it comes back and informs the employed bees. Then out of these neighbourhoods, the employed bees will select a neighbourhood and start exploiting the food sources in the neighbourhood. Only one employed bee will perform the search in one neighbourhood. While exploiting the food sources the bees will also try and find other food sources in the neighbourhood. If they find a food source, which is better than the current food source, they will start exploiting that food source and abandon the previous food source. Once the employed bees are done exploiting the food sources, the bees will move to the dancing area. In the dancing area, the onlooker bees will look at the dance of the employed bees and based on the dance they will decide which bee's neighbourhood has better food sources and the onlooker bees will go to that neighbourhood and start exploiting the food sources there. If they find a better food source than the employed bees, then the onlooker bee will become an employed bee from there on and the employed bee of that neighbourhood will become a scout bee [1]. If a neighbourhood in which the employed bees are searching for food depletes i.e. the employed bees aren't able to find any more food source in the neighbourhood then the employed bees abandons the neighbourhood and become scout bee to find a new food source.

The above process mentioned is how bees find their food sources. And based on this an algorithm is developed which is called the Artificial Bee Colony algorithm which can be understood using the following pseudocode.

```
Result: The best answer found in all the iterations
Create a queue of all possible neighborhoods
for each ScoutBee do
    Find an initial solution from the search space
end
while MAX_ITERATIONS is reached do
    for each EmployedBee do
        Use the initial solutions found by the scout bees
        while limit is reached do
            Exploit the neighborhood for better solutions
            if new solution found is better then
                move to that solution
            end
        end
    end
    for each onlookerbee do
        Select the employed bee using roulette wheel selection method
        while limit is reached do
            Exploit the neighborhood for better solutions
            if new solution found is better then
                move to that solution
        end
    end
    if A solution cannot be improved then
        Replace it with a solution found by the scout bee.
    Save the best answer
end
return Best ans found
```

Algorithm 1: Bee colony pseudo code

C. Implementation

1) *State Space*
The state of the problem is represented using a string with multiple delimiters which in this case would be the depot. All the numbers between the delimiters represents the a route for a truck.

Customers	1	2	3	4	5	6	7	

Solution	$X = \{0,1,2,0,4,5,3,0,6,7,0,0\}$							Route 1: {0,1,2,0}
								Route 2: {0,4,5,3,0}
								Route 3: {0,6,7,0}
								Route 4: {0,0}

Fig. 1. State Space [6]

The size of the state space is $N + K + 1$ where N is the number of customers K is the number of vehicles.

2) *Fitness Function*
This is one of the most important part of the artificial bee colony algorithm. The fitness function is the measure of how good the food source is in the neighbourhood and the nectar quantity of the food source. The fitness function given below is used for solving the CVRP problem [6].

$$f(x) = o(x) + \beta * p(x)$$

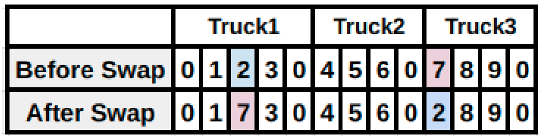
where,
$p(x) = \sum_{i=1}^{N} d_i y_{ik} - q_k$
$o(x) = \sum_{i=1}^{N} c_{i,i+1}$
$\beta = iteration\_index \times no\_of\_iterations$

This fitness function in particular is used because this fitness function allows infeasible solutions in the beginning but as the number of iterations increases. It penalizes the infeasible solution so it prevents the algorithm from getting stuck at the local minima but at the same time insures that the solution keeps on improving.

3) *Scout Bee Phase*
In this section of the algorithm, the scout bees are utilized to find the initial solutions. For this, we randomly shuffle the customers and then divide the customers in trucks according to the truck capacity so there are cases in which sometimes the constraints cannot be satisfied i.e. infeasible solutions which can be used as initial solution, but the rule selected in this paper that changes the performance of the algorithm significantly is that we always use a feasible solution. So if we get an infeasible solution, we discard it and continue to find a solution until we find a feasible solutions in the search space.

4) *Employed Bee Phase*
In this phase, the employed bees go out and explore the food source found by the scout bee. The employed bee then tries to find a better food source in the neighbourhood. So to generate neighbours we use the following operations.

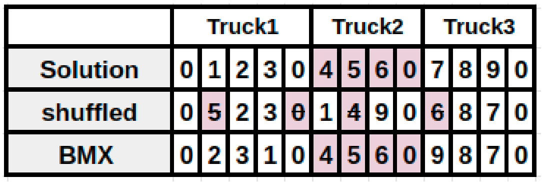
a) Swap Operation[6]: In this operation, we select two random trucks. From those two random trucks, we select a random customer in both the trucks and exchange the customers. Like in the below example, we selected truck 1 and 3 and in truck 1 we selected customer 2 and in truck 3 we selected customer 7. Then we swapped them so now truck 1 will serve customer 7 and truck 3 will serve customer 2.



	Truck1					Truck2				Truck3			
Before Swap	0	1	2	3	0	4	5	6	0	7	8	9	0
After Swap	0	1	7	3	0	4	5	6	0	2	8	9	0

Fig. 2. Swap Operation

b) BMX Operation[6]: In this operation we try to maintain the best part of the candidate solution and shuffle the rest and then divide them into trucks. In the below example, truck 2 has the lowest cost so it is maintained and the rest elements are shuffled.



	Truck1					Truck2				Truck3			
Solution	0	1	2	3	0	4	5	6	0	7	8	9	0
shuffled	0	5	2	3	0	1	4	9	0	6	8	7	0
BMX	0	2	3	1	0	4	5	6	0	9	8	7	0

Fig. 3. BMX Operation

We find the neighbourhood using these operators and update them, if the neighbourhood is better or if we are not able to find a better neighbourhood after the iteration limit is exceeded. Then the neighbourhood is abandoned by the employed bees [6].

$$limit = SN \times K$$

where,
SN = Number of bees
K = Number of trucks

5) *Onlooker Bee Phase*
In this phase, the employed bees return with the best solution in their neighbourhood. The onlooker bees use the roulette wheel selection method to select an employed bee's neighbourhood and start exploiting that neighbourhood. If they find a better solution the onlooker bee becomes the employed bee and participate in the employed bee phase.

Following is the flow chart of the implementation.

V. Parallel Artificial Bee Colony Architecture

In the previous section, the sequential algorithm for solving the artificial bee colony was explained. In this section, the parallel architecture of solving the same problem will be explained.

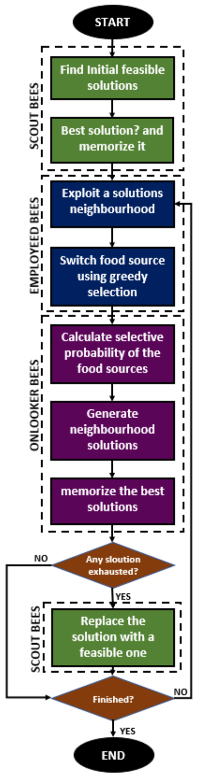

START
SCOUT BEES
Find Initial feasible solutions
Best solution? and memorize it
EMPLOYEED BEES
Exploit a solutions neighbourhood
Switch food source using greedy selection
ONLOOKER BEES
Calculate selective probability of the food sources
Generate neighbourhood solutions
memorize the best solutions
NO
Any sloution exhausted?
YES
SCOUT BEES
Replace the solution with a feasible one
Finished?
NO
YES
END

Fig. 4. Sequential Algorithm

A. Sequential dependencies

Sequential dependencies in a program are parts of the program that are not parallelizable.

The artificial bee colony algorithm is tricky to parallelize as there are a lot of sequential dependencies in the algorithm [9], which are as follows

- In the previous algorithm, there are multiple iterations of the algorithm which cannot be parallelized as the state of each round is dependant on the state of the previous round.
- The onlooker bees in the previous algorithm used to individually go to the employed bees neighbourhood and update their best answer which now would be a problem as multiple threads can try to update the same answer [9].
- The selection of the best answer after a round has completed.

B. Scaling

1) Strong Scaling [10]: Strong scaling is basically increasing the number of cores while the problem size remains the same. So in an ideal world if we are running a program on K cores it should take $1/K$ amount of time to run the problem of the same size.

$$Speedup(N, K) = \frac{T_{seq}(N, 1)}{T_{par}(N, K)}$$

$$Efficiency(N, K) = \frac{Speedup(N, K)}{K}$$

where,
N = Problem set size.
K = Number of cores.

We understand that the speedup is ideal if the efficiency is 1. In fig 5. we see that in strong scaling with the

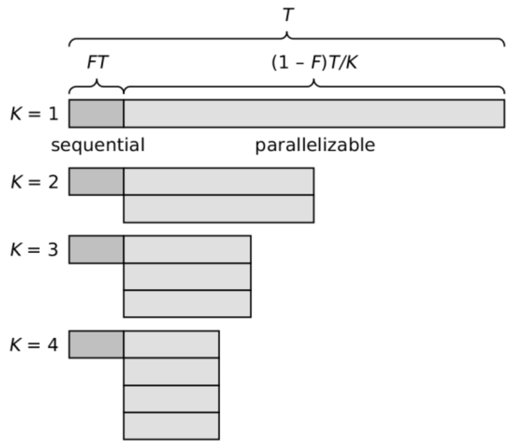


Fig. 5. Strong scaling with sequential fraction.[10]

increase in the number of cores the parallel part of the program is divided but the sequential part of the program still remains the same and after a point it becomes dominant over the parallel part. Hence the performance of strong scaling decreases as the number of cores increases in case of sequential dependencies.

2) Weak Scaling [10]: The most basic idea on which week scaling is based on is that instead of solving the same problem quicker we often want to solve problem of larger size in the same time. [10]. So in this type of scaling we increase the problem set size along with the number of cores.

$$Sizeup(N,K) = \frac{N(K)}{N(1)} \times \frac{T_{seq}(N(1),1)}{T_{par}(N(K),K)}$$

$$Efficiency(N,K) = \frac{Sizeup(N,K)}{K}$$

where,
K: Number of cores.
N(1): Problem set size to run on 1 core.
N(K): Problem set size to run on K cores.
We understand that the sizeup is ideal if the efficiency is 1. In Fig 6. we can see that weak scaling gives near to ideal efficiency even with sequential parts in the program.

C. Parallelizable parts

Even though there are parts that are sequentially dependant, there are parts that are parallelizable, some of which are as follows:

- Scout bee phase: This can easily be parallelized in which each thread can be responsible for creating a feasible solution and all the threads will work towards populating the queue.
- Employed bee phase: This part of the algorithm can also be easily parallelized as each thread will behave like a bee and take one neighbourhood from the scout-bee queue and work on finding the best solution in the neighbourhood.
- Onlooker Bee phase: This part is very tricky to parallelize since the job of the onlooker bee is to go to the employed bee's neighbourhood and exploit it. If they find a better answer, they update the best answer in the employed bees neighbourhood but now if we make each thread an onlooker bee then there will be a conflict of multiple onlooker bees deciding to exploit the same employed bee neighbourhood. So the following techniques were used:
    1) Each thread will represent a single onlooker bee.
    2) First the onlooker bees create a local copy of the neighbourhoods of their respective employed bees.
    3) Each onlooker bee will then update their own local copy of their employed bees neighbourhood.
    4) Now each onlooker bee's local copy has the best answer for their respective employed bee and then their best answer is reduced to their employed bees global copy.

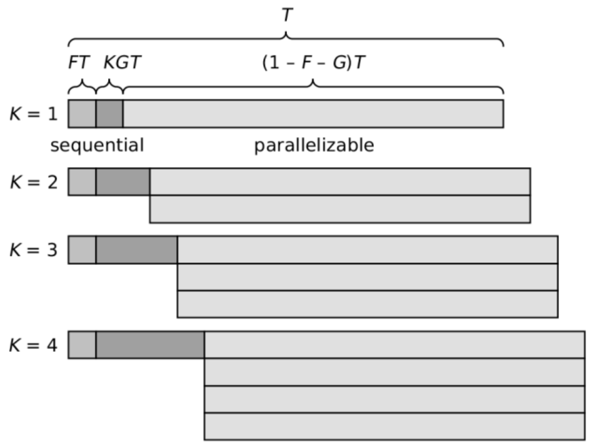


Fig. 6. Weak scaling with sequential fraction.[10]

The algorithm for the parallel architecture is shown in Fig 7.

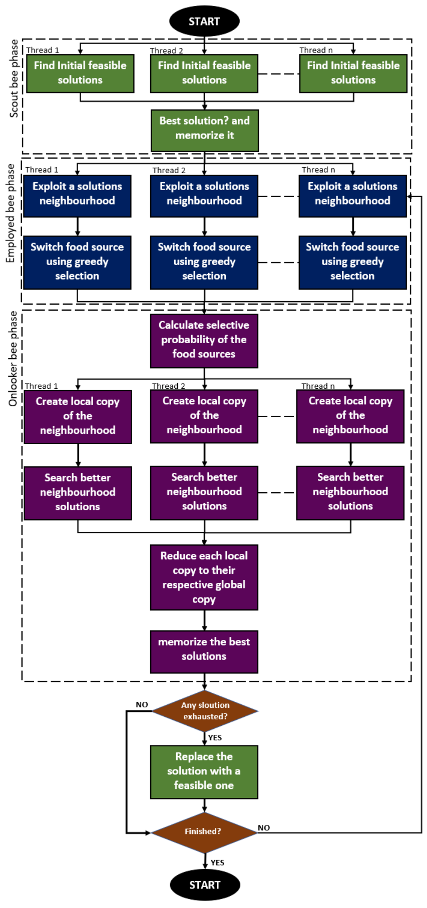

START
Scout bee phase
Thread 1
Thread 2
Thread n
Find Initial feasible solutions
Find Initial feasible solutions
Find Initial feasible solutions
Best solution? and memorize it
Employed bee phase
Thread 1
Thread 2
Thread n
Exploit a solutions neighbourhood
Exploit a solutions neighbourhood
Exploit a solutions neighbourhood
Switch food source using greedy selection
Switch food source using greedy selection
Switch food source using greedy selection
Onlooker bee phase
Calculate selective probability of the food sources
Thread 1
Thread 2
Thread n
Create local copy of the neighbourhood
Create local copy of the neighbourhood
Create local copy of the neighbourhood
Search better neighbourhood solutions
Search better neighbourhood solutions
Search better neighbourhood solutions
Reduce each local copy to their respective global copy
memorize the best solutions
NO
Any sloution exhausted?
YES
Replace the solution with a feasible one
Finished?
NO
YES
START

Fig. 7. Parallel Algorithm

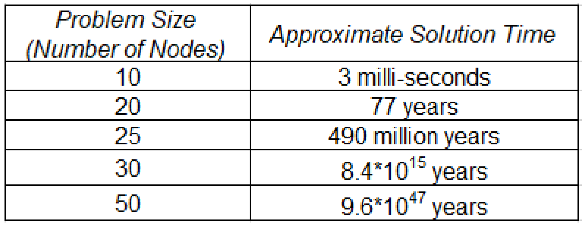
VI. Results

The Naive approach run-time and the Bee Colony algorithm run-time are based on an Intel Core i7-7500U @ $2.70GHz \times 4$ CPU. The parallel architecture of the bee colony algorithm is run on tardis.cs.rit.edu cluster which has Intel Xeon E3-1220 @ $3.0GHz \times 12$ processor.

A. Exact Algorithm

There results are going to be discussed for two types of exact algorithms which are as follows.

- **Complete Brute force:** This technique is not implemented in this paper but is inspired from this source [4]. Following figure depicts the run-times for complete brute force algorithm on varied sizes of N.

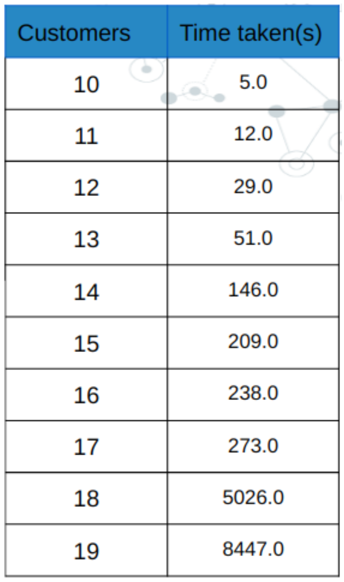


Problem Size (Number of Nodes)	*Approximate Solution Time*
10	3 milli-seconds
20	77 years
25	490 million years
30	$8.4*10^{15}$ years
50	$9.6*10^{47}$ years

Fig. 8. Complete brute force run-times[4]

Fig 8. shows the run-times for running a complete brute-force algorithm on a 1GHz processor. [4]

- **Optimized Brute Force:** The algorithm explained in the first section of this paper were ran for $N = 10, 11, \ldots, 19$ where N is the number of customers. The run times for these are shown in the below table. After this the time



Customers	Time taken(s)
10	5.0
11	12.0
12	29.0
13	51.0
14	146.0
15	209.0
16	238.0
17	273.0
18	5026.0
19	8447.0

Fig. 9. Time Taken For Optimized Brute force

taken increases exponentially. A 5-degree polynomial was used to fit the given data. The polynomial is as follows.

$$\begin{aligned} y = & -3.1 \times 10^7 x^0 + 1.1 \times 10^7 x^1 \\ & -1.7 \times 10^6 x^2 + 1.3 \times 10^5 x^3 \\ & -4.8 \times 10^3 x^4 + 7.0 \times 10^1 x^5 \end{aligned}$$

In the above polynomial x is the number of nodes and y is the time taken to find the optimal result for that node. So this was done from x = 20 until x = 50 with an increment of 1 and the following result was obtained.

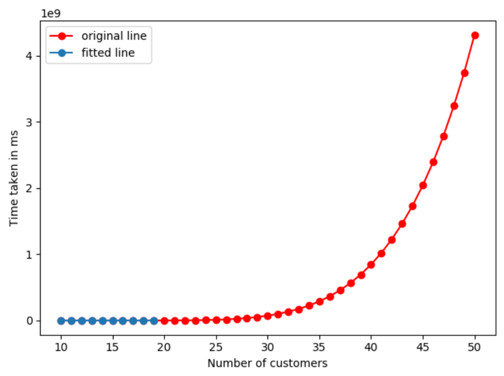
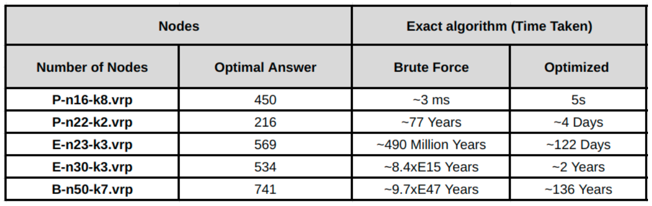


Fig. 10. polynomial fit to predict for N=50

The time comparison for the complete brute force and optimized brute force is given in Fig 11.



Nodes		Exact algorithm (Time Taken)	
Number of Nodes	**Optimal Answer**	**Brute Force**	**Optimized**
P-n16-k8.vrp	450	~3 ms	5s
P-n22-k2.vrp	216	~77 Years	~4 Days
E-n23-k3.vrp	569	~490 Million Years	~122 Days
E-n30-k3.vrp	534	~8.4xE15 Years	~2 Years
B-n50-k7.vrp	741	~9.7xE47 Years	~136 Years

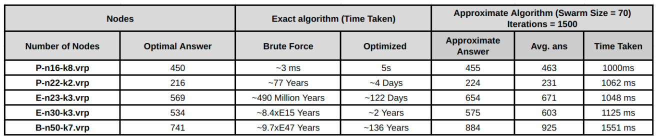
Fig. 11. comparison for the complete brute force and optimized brute force

B. Sequential Bee Colony approach

In this approach the criteria for understanding that the neighbourhood is extinct is to set a limit. The limit in this case is:

$$limit = 0.5 \times CS \times K$$

where CS=Colony size, K=Constant. Following is the comparative time for the the exact algorithm and bee colony algorithm. in Fig 12. we can see that the results obtained were



Nodes		Exact algorithm (Time Taken)		Approximate Algorithm (Swarm Size = 70) Iterations = 1500		
Number of Nodes	**Optimal Answer**	**Brute Force**	**Optimized**	**Approximate Answer**	**Avg. ans**	**Time Taken**
P-n16-k8.vrp	450	~3 ms	5s	455	463	1000ms
P-n22-k2.vrp	216	~77 Years	~4 Days	224	231	1062 ms
E-n23-k3.vrp	569	~490 Million Years	~122 Days	654	671	1048 ms
E-n30-k3.vrp	534	~8.4xE15 Years	~2 Years	575	603	1125 ms
B-n50-k7.vrp	741	~9.7xE47 Years	~136 Years	884	925	1551 ms

Fig. 12. Comparison for the complete brute force, optimized brute force and Bee Colony approach

not the most optimal result but were very close but the time taken was substantially low.

C. Parallel Bee Colony approach

The outputs of the parallel program in terms of correctness were same as the sequential bee colony algorithm. Following specifications were used to perform the scaling test of the parallel architecture.

- Dataset: E-n101-k14.vrp
- Number of iterations: 1500

Two scaling tests were done on this algorithm with the following specifications

- **Strong Scaling:** Swarm size taken was 5000 bees. Following are the graphs of efficiency and speedup for the parallel architecture of the artificial bee colony algorithm.

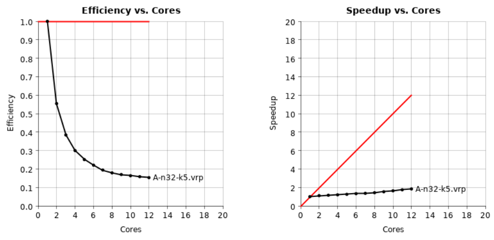


Fig. 13. Efficiency and speed up of strong scaling

The strong scaling is not optimal due to the huge sequential dependencies in the program as explained in the previous section.

- **Weak scaling:** Swarm size taken were 1 Core = 300, 2 Core = 600, 3 Core = 900 . . . 12 core = 3600. Following are the graphs of efficiency and speedup for the parallel architecture of the artificial bee colony algorithm.

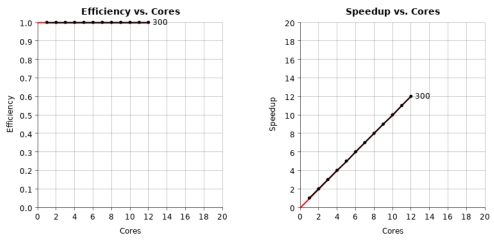


Fig. 14. Efficiency and speed up of week scaling

Now since the weak scaling is used to balance out the sequential dependencies we can see that the program has perfect weak scaling.

D. Answers

In the previous results we tested the run times and scaling of the parallel algorithms. In this section we increased the number of cores along with number of bees to check if the answer increases. The dataset on which this test was run was: E-n101-k14.vrp

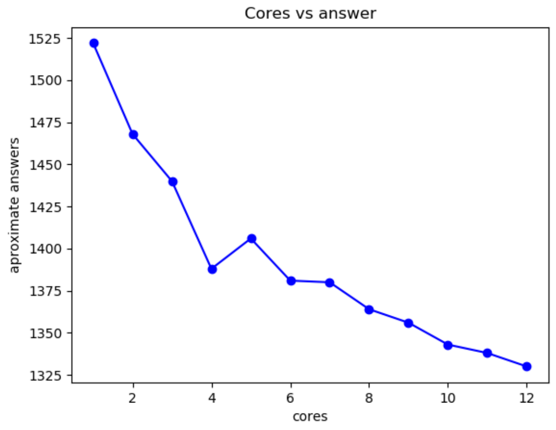


Fig. 15. Solution improvement with increase in number of cores a

As we can see in figure 12, as we increase the number of bees and number of cores on which the algorithm runs, it improves the results of the algorithm.

VII. Conclusion

To conclude the paper we discussed multiple approaches to solve the Constraint Vehicle Routing Problem (CVRP). Their runtime were compared and contrasted and their outputs were compared and contrasted. In the end this paper successfully implements all the above mentioned algorithms.

Acknowledgment

I would like to thank Dr. Zack Butler for guiding me through the project and helping me whenever I needed it.

References

[1] Pjvpjv, "Vehicle routing problem," 2006, [Online; accessed 29-September-2012]. [Online]. Available: https://en.wikipedia.org/wiki/Vehicle_routing_problem

[2] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," vol. 6, pp. 80–91, 1959.

[3] S. Nanda Kumar and R. Panneerselvam, "A survey on the vehicle routing problem and its variants," *Intelligent Information Management*, vol. 04, 01 2012.

[4] A. Seshadri, "Vehicle routing problems 101," 2015. [Online]. Available: https://medium.com/opex-analytics/opex-101-vehicle-routing-problems-262a173f4214

[5] G. L. J.-Y. P. J-F Cordeau, M Gendreau and F. Semet, "A guide to vehicle routing heuristics," *Journal of the Operational Research*, vol. 53, pp. 512–522, 2002.

[6] S. Z. . C. Lee, "An improved artificial bee colony algorithm for the capacitated vehicle routing problem," *2015 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 53, pp. 512–522, 2015.

[7] "Branchandcut.org," 2006. [Online]. Available: https://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/index.htm.old

[8] geeksforgeeks, "Count number of ways to partition a set into k subsets." [Online]. Available: https://www.geeksforgeeks.org/count-number-of-ways-to-partition-a-set-into-k-subsets/

[9] H. Narasimhan, "Parallel artificial bee colony (pabc) algorithm," *2009 World Congress on Nature Biologically Inspired Computing (NaBIC)*, pp. 512–522, 2009.

[10] A. Kaminsky, *BIG CPU, BIG DATA Solving the World's Toughest Computational Problems with Parallel Computing Second Edition*, 2006.