www.editada.org

International Journal of Combinatorial Optimization Problems and Informatics, 15(2), May-Aug 2024, 147-159. ISSN: 2007-1558. https://doi.org/10.61467/2007.1558.2024.v15i2.438

An efficient Parallel solution for the CVRP using k-NN, Matlab, and CUDA

José Alberto Hernández-Aguilar [1], *Emanuel Salinas-Carrasco* [1], *Crispín Zavala-Díaz* [1], *and Julio César Ponce-Gallegos*[2]
[1]Autonomous University of Morelos State, Av. Universidad 1001, Col. Chamilpa, 62209, Cuernavaca, Morelos, México
[2]Autonomous University of Aguascalientes, Av. Universidad 940, 20100 Ciudad Universitaria, Aguascalientes, Aguascalientes, México.
E-mails: jose_hernandez@uaem.mx

Abstract. The nearest neighbor algorithm is analyzed to implement a solution to the Capacitated Vehicle Routing Problem (CVRP) to determine one or several routes of a fleet of vehicles with a specific load capacity. It is intended to take advantage of the power of GPUs to reduce the time needed to execute the algorithm significantly. The methodology implies a programming model and development scenarios. Firstly, the sequential nearest neighbor algorithm is analyzed to solve the CVRP, and its parallel implementation is discussed later. Matlab is used as a tool to implement CVRP and perform numerical and matrix calculations because it can interact with GPUs and CUDA. The effectiveness and efficiency of both implementations are tested in different CVRP instances in terms of execution time. The results indicate that parallel implementation is not just suitable but highly beneficial for large instances (greater than or equal to three hundred nodes).

Keywords: CVRP solution, k-NN, CUDA, Matlab, GPU computing.

Article Info
Received May 1, 2024
Accepted Jun 1, 2024

1 Introduction

The VRP – Vehicle Routing Problem is a widely recognized and challenging combinatorial optimization problem, as acknowledged by (Dantzig and Ramser, 1959). Its applications are diverse and impactful, spanning logistics, distribution, material delivery and collection, personnel transportation, computer network routing, relief route planning in natural disasters, and correspondence distribution, as noted by (Ponce et al., 2014).

The VRP problem is an NP-complete problem for which efforts have been made to find an optimal solution with various input data and variants of VRP. However, as the complexity increases in finding a solution, the problem's behavior becomes exponential, not polynomial. In Combinatorial Optimization, VRP is considered NP-hard, and for complexity theory, it is classified as NP-complete.

1.1 What is VRP?

VRP is a widespread problem today, so it is widely used in transportation and logistics. It is exhaustive to solve, but it has countless applications in daily life. It is evident that if the needs of human beings increase, various problems arise, such as this problem in combinatorial optimization is used to improve the travel times and the distance of the routes of a vehicle; it is also known as TSP - Travel Salesman Problem in its basic form. Dantzig and Ramser (1959) mention that the VRP proposes a group of routes drawn or planned so that one or more vehicles, starting from a Depot, begin their route to customers or points of visit. It is intended to reduce the distance and travel time the problem requires. According to (Ponce et al.,2014a), the most common objectives in vehicle routing problems are:

─ Minimize the total transportation cost; These can be the distance or time of the routes.

─ Minimize the number of vehicles and drivers required to satisfy customer demand.
─ Handling a balanced number of clients per vehicle for travel times and loading.
─ Minimize penalties associated with partial deliveries to customers.

1.2 CVRP

The CVRP is a variant of the VRP, according to (Vidal et al., 2013): "The traditional Capacitated Vehicle Routing Problem (CVRP) can be described in its simplest form as a fleet of vehicles with uniform capacities that has to satisfy the demand of a group of customers through a set of routes. That beginning and the end is in a common warehouse (Depot) and represents the lowest possible cost and the identification of the order of visits to them."

On the other hand, according to (Toth and Vigo, 2002), the CVRP is the most general VRP and consists of one or several vehicles with limited and constant capacity to distribute products according to customer demand. This NP-Hard type optimization problem combines the characteristics of a Bin Packing Problem (BPP) to assign loads to capable vehicles and a traveling agent problem (TSP) that aims to find the best route for each vehicle.

The routes. A route corresponds to a path that one or more vehicles must take, representing a cycle with the same point for departure and arrival in the depot. In contrast, a depot must exist because it is taken as the starting point where the routes begin; the nodes symbolize a stopping point or visit to a client.

According to (Rodríguez, 2012), the time cost that a vehicle takes to travel a route and the arcs that make up the designed route (k) must be added; consequently, the objective is to obtain a route with a minimum time concerning the distance traveled, for this the following rules are denoted: a route starts and ends at a depot, a customer or node will only be included in one route, and finally the total demand of a route, that is, all its customers must not exceed the capacity (C) of the vehicle.

The graph. Rodríguez (2012) states that the network is shaped through a series of routes and joined points, graphically formed by arcs created by the path from one point to another; the clients are where a point or node of the graph is located. Mathematically, a graph is a collection of vertices and edges that join these vertices. The vertices are the nodes or clients located in a plane's coordinates; consequently, the edges are the path that determines the route.

When the possible routes are obtained, the arches and the position of the nodes form a graph by themselves; this way, the total cost or demand, which is the most common aspect of this type of problem, can be determined.

The vehicles. In the CVRP, the variables are defined by the vehicle's capacity, but various real-life problems arise where the fleet of vehicles is different with different dimensions; that is, they are different in load or performance. For example, in the case of a logistics company, it is possible to observe that the delivery units have the same capabilities and qualities, which is why it would be called a homogeneous fleet.

For (Rodríguez, 2012), a vehicle has a limited capacity if it is considered to have a specific volume; that is, the vehicle supports a certain weight and volume, as it had previously been said that if all the vehicles of a particular problem are equal in capacity, they are considered homogeneous.

The most common objectives considered in vehicle routing problems are in computational terms; the instances (vehicle capacity) are fixed for the homogeneous fleet, but for the heterogeneous fleet consisting of different vehicles, the constraint and complexity increase; for this, more significant analysis of the solution method must be considered, and an algorithm that adapts to specific changes must be implemented.

Client or node. A node or client in CVRP must not meet a requirement or restriction that implies complexity; it is essential to define the scope because naming a node and its parameters may change when applied in various disciplinary fields. According to Rodríguez (2012), the nodes are the points where a client can be; therefore, a demand that will be delivered by a vehicle must be satisfied, such as in distribution, collection, or passengers. For this, a client must be visited at least once at the claim's request.

It is observed that the CVRP has applications in different areas and was proposed for transportation and planning where there are geographical locations. A node is a point where a vehicle must pass or arrive; its value would be a coordinate. In the case of Internet networks, a node could be a Router or a Host, and the parameter has an IP address; therefore, a node in this area will only have parameters explicitly assigned for a specific objective, in this case, the distance between other nodes and the distance between

the node and the reservoir. For practical and research purposes, it is expected to work with Cartesian coordinates (x, y) for the values of a node.

The depots. A route's starting point is considered a depot if it already exists. In transportation and logistics, the starting node of all designed routes is taken as depots. On the other hand, (Rodríguez, 2012) points out that the problems with various warehouses can have different characteristics, such as the merchandise output capacity, the location, and, therefore, the group of vehicles assigned to that warehouse. Each designed route must begin and end at the said depot.

The parameters of a depot in CVRP are geographic location coordinates (x, y) and the number of vehicles the depot contains. We are dealing with a simple CVRP; only one depot will be used for the problem.

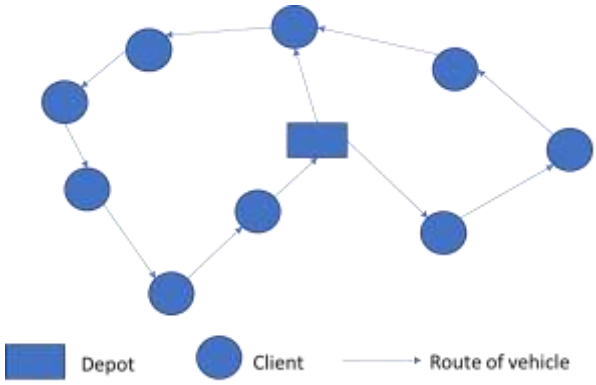

Fig. 1. Representation of a CVRP with a Central Depot and nine clients grouped on two routes with different vehicles (Ponce-Gallegos et al., 2012).

Figure 1 identifies the characteristics of CVRP; it is observed that a graph is drawn with nine nodes and a depot. It is considered a simple problem; the complexity increases when the number of nodes increases.

1.2 Mathematical formulation

The mathematical model for the CVRP proposed by (Ponce-Gallegos et al., 2012) is analyzed below:

$$\sum_{i\in V}\sum_{j\in V} C_{ij}X_{ij}. \quad \textbf{(1)}$$

Subject to:

$$\sum_{i\in V} X_{ij} = 1 \quad j \in V \setminus\{0\}. \quad \textbf{(2)}$$

$$\sum_{j\in V} X_{ij} = 1 \quad i \in V \setminus\{0\}\,. \quad \textbf{(3)}$$

$$\sum_{i\in V} X_{i0} = K\,. \quad \textbf{(4)}$$

$$\sum_{j\in V} X_{0j} = K\,. \quad \textbf{(5)}$$

$$X_{i,j} \in \{0,1\} \quad i,j \in V\,. \quad \textbf{(6)}$$

$$\sum_{i\notin S}\sum_{j\in S} X_{ij} \geq r(S), \quad \forall\, S \subset V \setminus\{0\},\ S \neq 0\,. \quad \textbf{(7)}$$

According to (Ponce-Gallegos et al., 2012), equations 2 to 5 are routing restrictions. Equation 6 is for the two-index model that considers the decision variables. Equation 7 prevents the existence of sub-tours by controlling capacity and cut-off restrictions. Furthermore, r(S) represents the minimum number of vehicles needed to satisfy the demand in S.

The general objectives of CVRP are, according to (Rodríguez, 2012):

a) Minimize the total cost of the vehicles used, assuming all clients must be visited; b) Minimize the distance or total time of the routes, assuming that all clients must be visited; and c) Minimize the number of vehicles used, assuming all customers must be visited. Finally, d) Minimize the total distance traveled.

The related work and the selected algorithm to be parallelized in Matlab to solve the CVRP are presented in the next section.

2 Related work

In (Cossio-Franco et al., 2018), an approach to solving CVRP instances of the Metropolitan Area of Guadalajara is discussed; in this algorithm, 37 nodes with a capacity of 100 are analyzed, for which the Nearest Neighbor Algorithm is used with instances of (NEO, 2024). This work implements a k-NN algorithm in Matlab and observes optimal algorithm performance; however, only small instances are compared to the tests intended to be analyzed in this research.

Abdelatti and Sodhi (2020) propose a hybrid algorithm combining a genetic algorithm and 2-opt to solve CVRP. This algorithm was designed to be executed on an NVIDIA GPU to exploit GPU architecture. For this purpose, they distribute arrays of elements over the GPU grid using kernel functions. Their algorithm provides quality solutions within reasonable computational times and works well for minor benchmark problems. They compared the algorithm performance running in CPU and GPU, and for all the instances compared, the computational times were smaller for the GPU.

(Abdelatti & Sodhi, 2020) used HPC platforms to design a parallel genetic algorithm for solving large-scale VRP problems. They implemented the algorithm over eight GPU Nvidia DGX servers, and the best performance was obtained by mapping all algorithm arrays into block threads. They used VRP benchmark problems of up to 20,000 nodes to compare algorithm performance with different counts of GPUs and multi-CPUs. The main results include a reduction of execution time by a factor of 1,700 and increased performance when raising the number of GPUs.

In (Yelmewad & Talawar, 2020), a feasible initial solution for solving CVRP is constructed using the k Nearest Neighborhood (KNN) Algorithm, later five improvement heuristics are applied in two types and consecutive order: a) inter route: swap and relocate, and b) intra-route: 2-opt, or-opt, and 3-opt. In the intra-route, the heuristics are applied until no further progress exists. In the inter-route process, nodes are exchanged (swapped) or relocated. To reduce the hours of CPU time used to improve the solution, the authors use parallelism at the node level, where one thread is mapped per node. This process consists of collecting information on each node in the m route and selecting the best one. For this purpose, CUDA blocks were used. A large data set of 20,000 customers was used to evaluate the performance of the proposed parallel strategy. The computer used for the experiments was a core i7, eight cores 3.6 GHz, 16GB of RAM, and an Nvidia Tesla P100. The sequential version was coded in C, and the parallel version was coded in CUDA. The parallel strategy shows an average speedup of 82.17x.

(Abbasi, et al., 2020) discuss a method to accelerate the solution for complex vehicle routing problems in the cloud implementation of intelligent transportation systems. The authors proposed the parallelization of a Genetic Algorithm by designing three concurrent kernels, each running some dependent effective operators of the GA: kernel 1 calculates the fitness function; kernel 2 performs crossover, mutation, and fitness function; kernel 3 executes selection. Kernels are synchronized using a low-cost mechanism; the CPU generates the initial population and receives the result after running GA. The results confirm the method's efficiency in parallelizing the GA on GPUs; the maximum speedup of CUDA-based TSP on 1280 cores was 13.73x.

Yelmewad and Talawar (2021) presented two GPU-based strategies for the LSH—Local Search Heuristic algorithm to solve large-scale CVRP instances. The route and customer-level parallel strategies reduce the execution time to improve the solution. The inter-route and the intra-route are implemented in GPUs using CUDA threads. The approach was tested with up to 30,000 customers; compared to the sequential solution, the parallel design obtained a speedup of 147.19x.

2.1 Algorithm used to solve the CVRP

Several algorithms are used to solve complex problems such as CVRP; among the approximate algorithms, there are Metaheuristics and Evolutionary Algorithms. These are considered robust methods capable of dealing with optimization problems, and their objectives commonly include maximizing or minimizing a function. We selected the k-NN as the first approach to providing a feasible solution to the CVRP because it is reportedly simple to implement in sequential and parallel approaches.

K-NN. The Nearest Neighbor Algorithm (k-NN) was one of the first implemented to provide a solution to the VRP; the characteristic is that it generates a short route but is not necessarily ideal. K-NN is classified as a constructive, greedy, or ambitious heuristic algorithm and provides suitable solutions with efficient calculation times. Despite the lack of certainty that the results give an optimal and completely accurate solution, the closeness of the results still needs to be determined. It is worth mentioning that some authors affirm that it increases the efficiency of the solution search process or probable results comparable to the optimal with a satisfactory level of acceptance in a reasonable time in practice (Cossio-Franco et al., 2018).

The k-NN algorithm is straightforward and can be summarized in the following steps (Raschka & Mirjalili, 2017):

1. Choosing the number of k and a measure of distance.
2. Finding the k-nearest neighbors of the desired sample to classify.
3. Assigning the label of the class by a majority of votes.

To analyze how the k-NN algorithm operates in depth, it is necessary to have prior concepts and describe basic terms to understand the premises. (Silverman & Jones, 1989) raised K-nearest neighbors is an inductive learning algorithm that classifies input data sets.

In (Cambronero & Moreno, 2010) is mentioned explicitly one of the characteristics is that input data are considered sample data, which are divided into two: the training set to determine the parameters of the classifier and the test set known as the generalization set to estimate the generalization error, since its objective is for the classifier to achieve a small generalization error, avoiding overfitting. Overfitting or over-training refers to the predictive capacity of the models obtained.

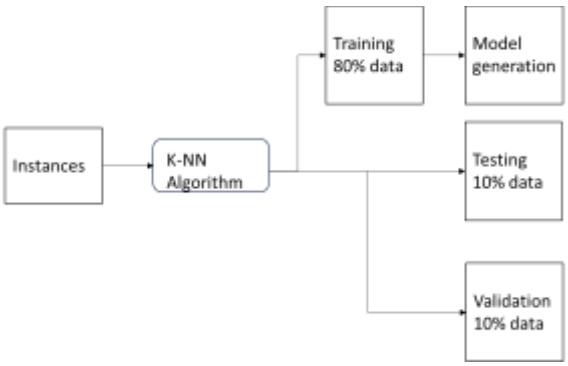

Fig. 2. Classifier representation of the data in k-NN, adapted from (Cambronero and Moreno, 2010).

Figure 3 shows the representation of the k-NN classifier, and according to (Cambronero and Moreno, 2010): “The loss of the ability to generalize leads to undesirable behavior. The training set is usually divided into training sets themselves. Likewise, another validation set is also created to adjust the model”. As machine learning methodology states, eighty percent of the data are usually used to train, ten percent as a validation set, and the remaining ten percent to estimate generalization. Searching for solutions is not optimal if the ability to generalize is lost. In this approach, the algorithm can recognize patterns and classify them.

However, (Silverman and Jones, 1989) mention that lazy learning is considered lazy because the algorithm learns locally. It means that the learning increases in a single set of proto-types or classes; for example, when building a route for the CVRP application case, there can be “n” routes. In contrast, the learning is local and increases in that set or route constructed; subsequently, the algorithm returns to form the following route (class), but the learning is decremented to zero, and so on. That is why it is considered lazy.

To choose a prototype, the input data must be divided into classes. In this case, the classes contain points like those previously mentioned in (Cambronero & Moreno, 2010); for this purpose, Euclides' formula is then described in detail as (Silverman & Jones, 1989) states:

"The space is partitioned into regions by locations and labels of the training examples. A point in space is assigned to class C if this is the most frequent class among the k closest training examples. For this purpose, the Euclidean Distance is used."

Euclidean distance:

$$d(xi , xj) = sqrt\ (sum(xir^2 , xjr^2)) . \quad (8)$$

The contributions of the previous perspectives define and use different terms in the algorithm's rules; integrating diverse points of view and opinions helps to build a broader picture. Next, we discuss the classification rule and the implementation of k-NN by (Cambronero and Moreno, 2010).

KNN rule. The most general neighborhood classification rule is the k-nearest neighbors classification rule or k-NN. It assumes that the closest prototypes have a similar posteriori probability. If ki (X) is the number of samples of the class present in the k nearest neighbors of X, this rule can be expressed as follows:

$$d(X) = Wc \text{ if } Kc\ (X) = max\ i\text{-}1 \rightarrow j\ ki\ (K) . \quad (9)$$

Given that, for the mathematical description, Silverman & Jones (1989) proposed the following for a classification algorithm: Given an exemplar xq that must be classified, let x1,..., xk be the k closest neighbors to xq in the learning examples, return where $\delta(a, b) = 1$ if a = b; and 0 in any other case. The value f (xq) produced by the algorithm as an estimator f(xq) is just the most common value of f among the k closest neighbors of xq. If we choose k=1, the nearest neighbor to xi determines its value.

$$f(x) \leftarrow argmax\ v \in V,\ i=1\ k\delta\ (u, f(xi)) . \quad (10)$$

It is said that when a point is a prospect and is considered close to the data set, it is named Nearest Neighbor Algorithm. On the other hand, to apply the classification rules, it is necessary to keep in mind the restrictions and the increase in computational cost that this entails, in this case, the dimension of the data (distances) and the size of the set (points).

3 Methodology

Currently, there are instances in repositories that can be used freely and accessible, for example, the works of Augerat et al., Breedam, Christofides and Eilon, and for the VRP variants Fisher, Christofides, Mingozzi and Toth, Rinaldi and Yarrow, Taillard, where is possible finding a compilation of the leading cases, described by different authors for the various variants of the vehicle routing problem. In (Cossio-Franco et al., 2018), the structure of a typical CVRP instance is described. It has four columns; in the first, the number of nodes or points is established; in the second column, the coordinate X; in the third column, where the coordinate is located; and the fourth column corresponds to the demand. The data is provided in simple files with a .txt extension to maintain standards and make it easy to handle and use in any program or application.

Figure 3 shows the flow chart of the k-NN Algorithm in one of its simple forms; a solution method for CVRP is considered an inductive, greedy method, which provides fast and feasible solutions close to optimal, according to (Cossio-Franco et al., 2018), that is the approach we used to be programmed in a parallel paradigm.

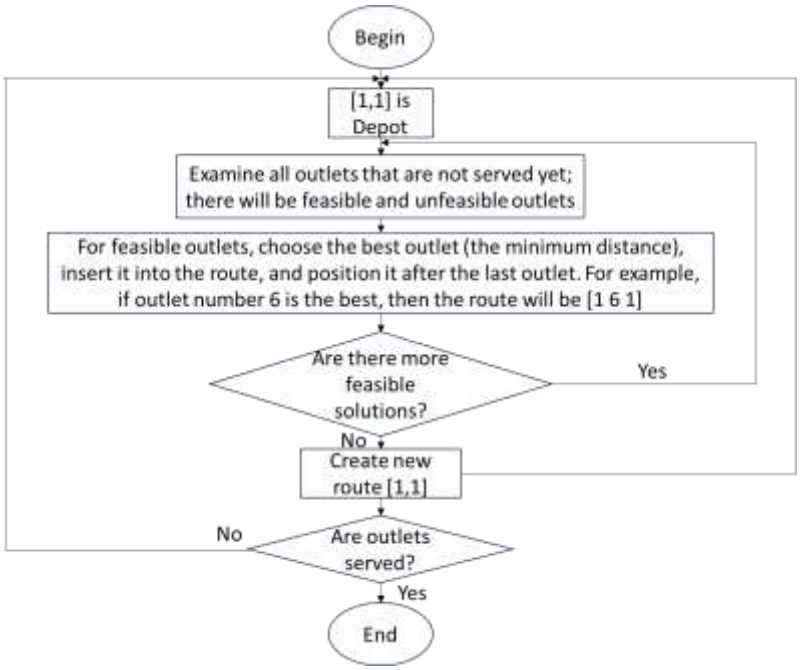

Fig. 3. Flowchart of the Nearest Neighbor Algorithm (Cossio-Franco et al., 2018).

3.1 Analysis of the KNN program in Matlab that solves CVRP instances

The sequential Nearest Neighbors (k-NN) algorithm that solves and graphs the solution of CVRP instances is built into the source code in Matlab and includes three files; this algorithm was proposed and programmed by (Wahid et al., 2011). In the first file, Problem25.m, a function is implemented that obtains the data from the input file; inside it, four columns of data are grouped corresponding to a matrix; the first is the number of points n, in this case, it contains 25 nodes (see the name of the .m file), the second and third column is the location of the x, y coordinates of a point or node that correspond to a Cartesian plane, in the fourth column are the demand parameters.

A second file called NIAlgVRP.m contains the core part. In this function, two parameters are received: the problem and capacity values that determine the number of points from the instance (Problem25.m file) and the vehicle capacity. It is made up of the following steps that match the flowchart shown in Figure 3 (Wahid et al., 2011):

Step 1: Start with [1 1], where node one is the Depot (initialize the route; it starts at the depot and ends at it).
Step 2: Examine all the points that need to be visited; there will be feasible and non-feasible points.
Step 3: For the feasible points, choose the best exit, that is, the minimum distance of the points to the repository of each data class, then insert it into the route and place it after the last exit in that route, for example, if the exit no. 6 is the best, so [1 6 1].
Step 4: Repeat steps 2 and 3 until there is no more feasible exit or the shortest distance on the same route.
Step 5: If no feasible output exists, create a new route [1 1] and repeat steps 1 to 4 for another classification.
Step 6: Instructions are repeated until all points are evaluated.

Three sections are observed in the source code:
─ In the first part, the matrix of distances from the instance and Euclid's formula for calculating them are formulated.
─ In the second part of the source code, it is possible to build a route with short distances in a matrix according to how the coordinates are classified; a repetitive structure is implemented while all the distances are not inserted, that is when there is no longer another route to build. The sentences will be repeated so that the routes are subsequently stored in a vector; then, a structure is initialized that repeats and validates the number of vehicles; that is, only routes are generated for it. A decision structure is identified since if the above is not met, the total sum of the distances of the other vehicles is transferred. Next, after evaluating all the distances, the best ones are inserted into the vector according to the algorithm's classification. Otherwise, if no feasible exits exist, a new Route is created [1 1].

─ In a third section, cycles are used to graph the vectors of both the routes and the data contained in each route. This last section focuses on the graphical visualization of the solution in a Cartesian plane.

The third file, ConstEvalVRP.m, is structured in three sections. It can be seen in the same way as the previous file that a header of variables is established where the distance matrices from the instance and the formula are initialized to calculate distances. In the second part, it is observed that the algorithm evaluates the distances in the routes, and with a repetitive structure, it stores the distances in each of these. Likewise, in the final part, the feasible routes are assigned to each vehicle; in this case, there is only one. It is noted that the parameters can be modified if more significant restrictions are required. This file directly communicates with the core part of the algorithm since parameters are received and sent during execution. Specifically, the function of this file is to classify and evaluate the data coming from the instance file.

3.2 Execution of the Nearest Neighbors Algorithm on Matlab for the CVRP solution

The implementation of the algorithm by (Wahid et al., 2011) contains an example instance with 25 points, a depot, and a vehicle with a capacity of 100. After the installation of Matlab, the mentioned instance is executed (see Figure 4); it should be noted that the program is in its sequential version; it runs on the CPU architecture, and the time taken to process a fragment of code is registered, in this case for the calculation of the matrix of distances, which later, at the end of program execution, will bring the total time of operations used to solve CVRP for this instance.

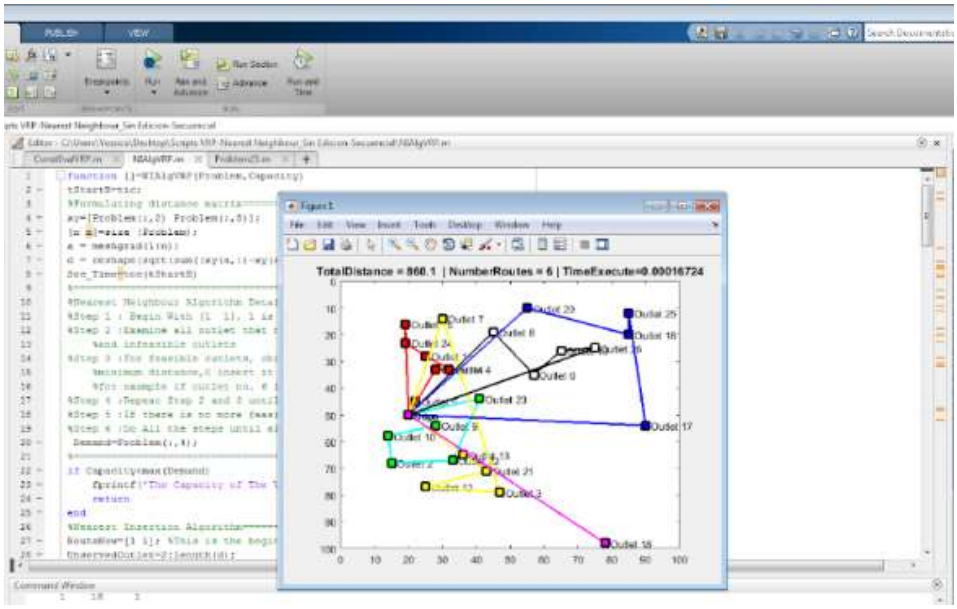

Fig. 4. Sequential Execution of Problem25.m in Matlab.

Figure 4 shows the test Matlab program for solving the CVRP problem using the "Nearest Neighbor" algorithm. Its execution time is .00016 seconds, and the total distance traveled using the sequential solution is 860.1. Likewise, the routes calculated with the algorithm are shown graphically.

2.3 Identification of functions that can be parallelized in Matlab

An analysis was conducted to identify the code fragments and functions to be parallelized. In the sequential version, Global Memory takes a specific time to transfer the operations to the CPU, and once processed, it retakes a particular time to present the results. Therefore, the sum of all operations and transfers increases the execution time. The parallel version aims to parallelize operations such as the calculation of matrices and the transfer of data on the GPUs; it should be noted that parallelism is data-oriented. Table 1 shows the identified functions and their descriptions in the first column, in the second column, the sequential version in Matlab, and in the third column, the parallelized version in Matlab-CUDA.

Table 1. Identification of functions to parallelize.

Description	Sequential version	The parallel version in Matlab - CUDA
File NIAlgVRP.m		
Matrix that stores the (x, y) coordinates of the points of instances.	xy= [Problem(:,2) Problem(:,3)];	xy = gpuArray(xy); AGPU = gpuArray(xy);
The vector that stores all elements of column 1 of the instance matrix, mesh grid, is used to process 2-D Matrices and 3-D Grids.	a = meshgrid(1:n);	a = gpuArray(pre)

Vector that applies the formula to calculate distances; a matrix-type function that converts the assigned data to matrix form.	d=reshape(sqrt(sum((xy(a,:)xy(a',:)).^2,2)),n,n);	d= arrayfun(@GpuFunction,AGPU,a);
		d = reshape(gather(d),n,n);
		
		AGPU = gpuArray(xy);
		RES = sqrt(sum((AGPU(a,:)-AGPU(a',:)).^2,2));
		
		d = reshape(gather(RES),n,n);
		d = reshape(sqrt(sum((xy(a,:)-xy(a',:)).^2,2)),n,n);

File ConstEvalVRP.m		
Matrix x,y that stores the (x, y) coordinates of the points in the instance file.	xy= [Problem(:,2) Problem(:,3)];	xy = gpuArray(xy);
A vector that stores all elements of column 1 of the instance matrix, the mesh grid is used for processing 2-D Matrices and 3-D Grids.	a = meshgrid(1:n);	a = gpuArray(a)
Vector b applies the formula to calculate distances, a matrix-type function that converts the assigned data to matrix form.	d=reshape(sqrt(sum((xy(a,:)xy(a',:)).^2,2)),n,n);	d = sqrt(sum((xy(a,:)-xy(a',:)).^2,2)) d = reshape(p,n,n);

3.4 Hardware and software tools

1. NVIDIA Graphics Card Driver and CUDA SDK. The CUDA toolkit can be downloaded from the official NVIDIA website (NVIDIA, 2024); it was installed manually following the instructions on the website.

2. Matlab R2014a.
3. NVIDIA Tesla C2075 card.

3.5 Design of Experiments

Next, Figures 5 and 6 show the technological platform's representation and the flow diagram for the sequential and parallel implementation, respectively.

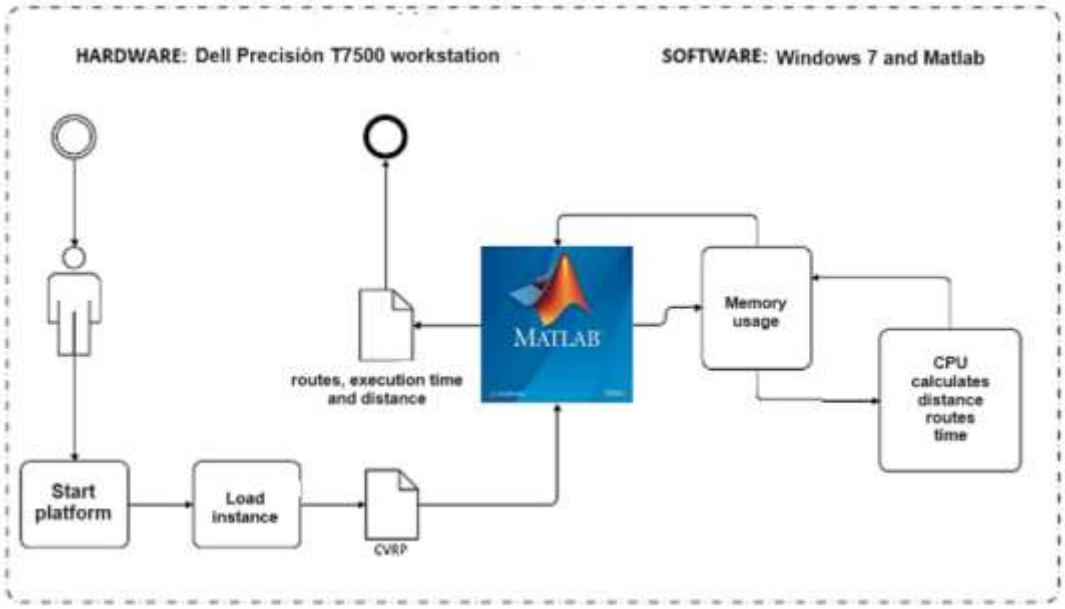

Fig. 5. Sequential algorithm experiment procedure diagram (Salinas-Carrasco, 2018).

The previous diagram shows the procedure conducted for the corresponding tests. For this sequential case, the CPU architecture solves the CVRP.

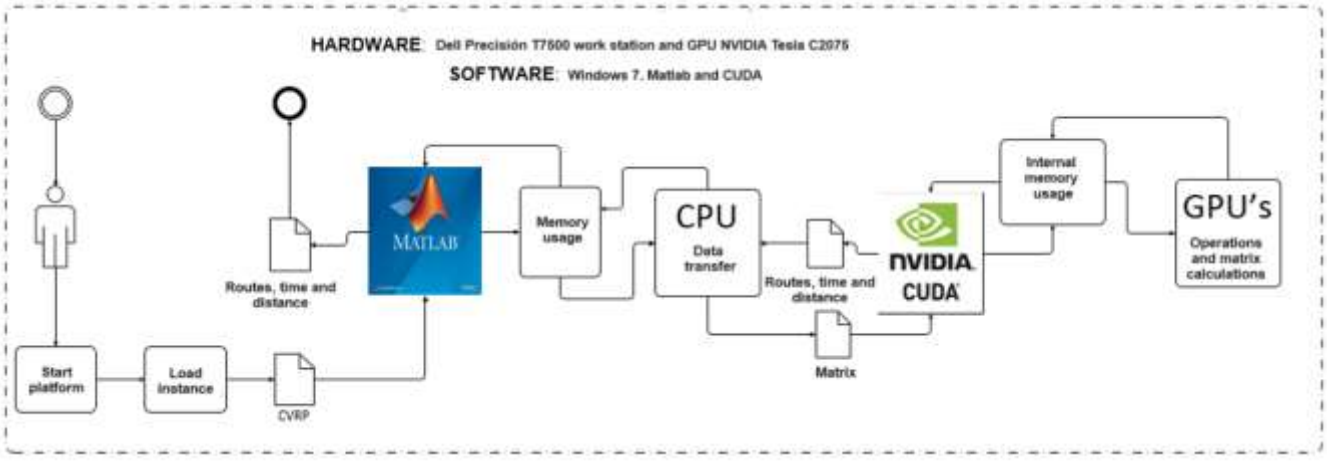

Fig. 6. Parallel algorithm experiment procedure diagram (Salinas-Carrasco, 2018).

The diagram in Figure 6 shows the algorithm testing procedure in the parallel version, for which the CUDA platform and the architecture of the GPUs solve the CVRP.

We tested KNN sequential and parallel implementations for solving the following CVRP instances: A-n33-k5.vrp, A-n101-k4, Kelly-n200.vrp, Kelly-n300. vrp, Kelly-n400.vrp, Kelly-n480.vrp. and X-n1001-k43. These experiments occur in two moments:

the first is when the instances run in the sequential version of the algorithm and the CPU execution time is recorded, and the second is when the instances run in the parallel version to compare execution times. Below are the comparative results for solving test instances between CPU and GPU.

4 Results and discussion

The tests were conducted with the instances shown in the first column of Table 2 to observe the behavior of the KNN algorithm for small and large instances.

Table 2. A comparison of sequential vs. parallel time for solving CVRP using the proposed methodology.

Instance	Sequential (t in ms)	Parallel (t in ms)
A-n33-k5.vrp	0.00012174	0.00100945
A-n101-k4	0.00322051	0.01793554
Kelly-n200.vrp	0.02992213	0.1009305
Kelly-n300.vrp	0.2132614	0.15742606
Kelly-n400.vrp	0.76234279	0.57086785
Kelly-n480.vrp	1.70193	0.71942274
X-n1001-k43	34.8621852	10.8095623

Table 2 compares the time for solving CVRP using sequential and parallel implementations. The sequential time in milliseconds is in the second column, and the parallel time in milliseconds is in the third column. The following figure shows the graphical results.

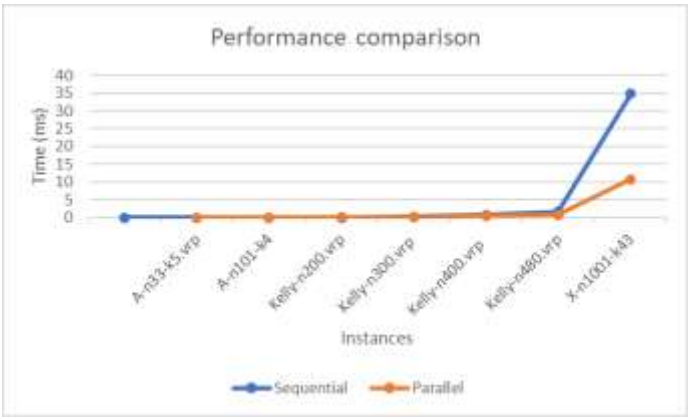

Fig. 7. Comparative graph for solving the CVRP instances, sequential in blue versus parallel in orange.

The results of the A-n33-k5.vrp instance show that the sequential version takes an average of 0.00012174 milliseconds to run on the CPU. In contrast, with the parallel version, there is an average of 0.00100945 milliseconds; therefore, it averages a longer time than the sequential version; that is, there is a time 0.00088771 milliseconds longer. In the parallel version for this instance, it takes a longer time, so we have that the behavior of the algorithm executed on the GPUs for a small instance of thirty-three points does not result in optimal time; therefore, it is appropriate to run said instance on a CPU.

In the results of the A-n101-k4.vrp instance, the sequential version is identified as taking an average of 0.00322051 milliseconds to execute when running on the CPU; however, on the GPU, there is an average of 0.01793554 milliseconds, just as in the previous instance the parallel version shows longer times with a difference of 0.01471503 milliseconds.

For the Kelly-n200.vrp instance, it is observed that the sequential version of the algorithm gives an average of 0.02992213 milliseconds when running on the CPU, while for the parallel version, there is an average of 0.1009305 milliseconds and a difference of 0.07100837 milliseconds. In this 200-point instance, considered medium-sized, it is appropriate to run it on a CPU for optimization purposes.

For Kelly-n300.vrp in the sequential version of the algorithm, it gives an average of 0.2132614 milliseconds when running on the CPU; however, in the parallel version, there is an average of 0.15742606 milliseconds, that is, for the 300-point instance, this version shows better results, since the execution time is reduced using the GPUs. A time gain of 0.05583534 milliseconds below the sequential one is also verified. This same trend can be observed for Kelly-n400.vrp and Kelly-n480.vrp.

In this same sense, for the X-n1001-k43.vrp instance, it is observed that for the sequential version, the tests take an average of 34.8621852 milliseconds when running on the CPU; on the other hand, for the Parallel version, there is an average of 10.8095623 milliseconds, it is possible to verify a considerable and therefore optimal time gain of 24.0526229 milliseconds thanks to the GPUs.

The parallel version behaves non-optimally for small instances of the tests carried out from 25 to 200 points; for a CVRP with instances less than 200, it is not convenient to use the GPUs. This is because there is a transfer latency and access to global memory to transport the data to the GPU Cores. That is why said latency cannot be hidden with operations because the data is small; therefore, running small CVRP instances on a CPU is convenient.

4 Conclusions and future work

The research objectives were achieved. We describe the characteristics of the CVRP and discuss the operation of the Nearest Neighbor Algorithm to obtain a description of the proposed solution and, subsequently, a solution for the problem in a sequential version. The algorithm was programmed in Matlab, analyzing the sequential version of (Wahid et al., 2011), which allows the identification of parallelizable functions implemented in CUDA and Matlab.

It is concluded that the Nearest Neighbors algorithm programmed in a sequential version on Matlab yields optimal times when executed on a CPU with instances of less than three hundred points. The algorithm represents an exponential behavior for instances greater than three hundred up to 1001 points; it is no longer efficient, becomes slow, and consumes many computing resources. This is because there is excessive data saturation, specifically in the transfer of global memory to the CPU, combined with the increase in the calculation of the CPU's operations.

However, for the parallel version, it is concluded that the algorithm yields optimal times with instances from three hundred points up to 1001 points with the GPU.

The lessons of this research were that GPU devices' excellent arithmetic calculation capacity has great potential for application in various computing and computer science areas. One such area is the solution of complex problems such as transportation, which allows obtaining solutions close to optimal and effectively reducing time.

For future work, it intends to conduct tests of various groups of data and algorithms in the R programming language, considered a programming environment and language focused on statistical analysis, well-known in data mining, statistical research, bioinformatics, and financial mathematics, that enables the use of libraries or packages that allow the calculation and visibility of graphs. We will test more significant instances of CVRP and evaluate the performance of the proposed algorithm in Matlab and CUDA languages.

References

Abbasi, M., Rafiee, M., Khosravi, M. R., Jolfaei, A., Menon, V. G., Koushyar, J. M. (2020). An efficient parallel genetic algorithm solution for vehicle routing problem in cloud implementation of the intelligent transportation systems. *Journal of Cloud Computing*, 9, 1-14.

Abdelatti, M. F., & Sodhi, M. S. (2020). *An improved GPU-accelerated heuristic technique applied to the capacitated vehicle routing problem*. In: GECCO 2020 - Proceedings of the 2020 Genetic and Evolutionary Computation Conference. https://doi.org/10.1145/3377930.3390159

Cambronero, C. G., & Moreno, I. G. (2010). *Algoritmos de aprendizaje kNN & KMEANS*. [Inteligencia en Redes de Telecomunicación]. Universidad Carlos III de Madrid.

Cossio-Franco, E. G., Hernández-Aguilar, J. A., Ochoa-Zezzatti, A., & Ponce-Gallegos, J. C. (2018). Comparison between instances to solve the CVRP. *International Journal of Combinatorial Optimization Problems and Informatics*, 9(2), 41-54.

Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, 6(1), 80-91. https://doi.org/10.1287/mnsc.6.1.80.

NEO (2024). *VRP Flavors*. https://Neo.Lcc.Uma.Es/Vrp/Vrp-Flavors/.

NVIDIA (2024). *SDK Manager*. Https://Developer.Nvidia.Com/Sdk-Manager.

Ponce-Gallegos, J. C., Aguilera, F. S. Q., Hernández-Aguilar, J. A., & Correa-Villalón, C. J. (2012). Logistics for the garbage collection through the use of Ant Colony algorithms. In*: Logistics Management and Optimization through Hybrid Artificial Intelligence Systems*. IGI Global https://doi.org/10.4018/978-1-4666-0297-7.ch002

Ponce, J., Recio, A., Ochoa, A., Ornelas, F., Padilla, A., & Torres, A. (2014). Implementación de un algoritmo de colonia de hormigas en CUDA, aplicado al Problema de Ruteo de Vehículos. En: Hernández-Aguilar, J.A., Zavala-Díaz, C., & Vakhania, N. (Eds.), *Aplicaciones modernas de optimización: La experiencia entre cuerpos académicos (1st ed.)*. Universidad Autónoma del Estado de Morelos.

Ponce, J., Recio, A., Ochoa, A., Hernández, A., Ornelas, F., Padilla, A., & Torres, A. (2014a). Algoritmo de colonia de hormigas en CUDA para la optimización de rutas de distribución. *Komputer Sapiens*, 6(3), 25-32.

Raschka, S., & Mirjalili, V. (2017). *Python Machine Learning: Machine Learning and Deep Learning with Python*. In New York.

Rodríguez, J. (2012*). Caracterización, Modelado y Determinación de las Rutas de la Flota en una Empresa de Render* [Master]. Universidad de Sevilla, Sevilla.

Silverman, B. W., & Jones, M. C. (1989). E. Fix and J.L. Hodges, (1951): An Important Contribution to Nonparametric Discriminant Analysis and Density Estimation: Commentary on Fix and Hodges (1951). *International Statistical Review / Revue Internationale de Statistique*, 7(3), 233-238. https://doi.org/10.2307/1403796

Toth, P., & Vigo, D. (2002). The Vehicle Routing Problem. *Society of Industrial and Applied Mathematics (SIAM)*. Monographs on Discrete Mathematics and Applications, (Eds.).

Salinas-Carrasco, E. (2018). *Implementación de un algoritmo para la solución del CRVP sobre unidades gráficas de procesamiento GPUs en Matlab* [Bachelor]. Universidad Autónoma del Estado de Morelos.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1), 1-21, https://doi.org/10.1016/j.ejor.2013.02.053.

Wahid, S., Dan, H., & Santosa, B. (2011). *Artificial immune system untuk penyelesaian vehicle routing problem with time windows*. [Undergraduate Thesis]. ITS.

Yelmewad, P., & Talawar, B. (2020). *GPU-based Parallel Heuristics for Capacited Vehicle Routing Problem*. In: Proceedings of CONECCT 2020 - 6th IEEE International Conference on Electronics, Computing and Communication Technologies. https://doi.org/10.1109/CONECCT50063.2020.9198667

Yelmewad, P., & Talawar, B. (2021). Parallel Version of Local Search Heuristic Algorithm to Solve Capacitated Vehicle Routing Problem. *Cluster Computing*, 24, 3671-3692.