Solving the Vehicle Routing Problem using Differential Evolution

232886*

ABSTRACT

The Differential Evolution (DE) algorithm is a powerful metaheuristic algorithm that utilises a population of solutions in order to find an optimal solution. DE is metaheuristic as it does not need the gradient of the function it's optimising. In this paper, I will be introducing a variation of DE designed for the Vehicle Routing Problem (VRP). This problem is similar to the TSP but requires a number of vehicles with limited capacity to visit a set of customers with specific demands and then return to the start node, all using the shortest path. The DE will be run on a set of different CVRP datasets and then compared to the random search and stochastic hill-climbing algorithms.

ACM Reference Format:
232886. . Solving the Vehicle Routing Problem using Differential Evolution. In . ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION

Vehicle Routing Problems (VRPs) are some of the most widely researched optimisation problems. They can be classified as a combinatorial or integer programming problem depending on their formulation. While the TSP could be seen as a very basic instance of a VRP where the number of vehicles is 1, the most common and basic VRP is the Capacitated VRP (CVRP). The CVRP was originally introduced by Dantzig et al.[7], in 1959. In the CVRP there are a set of vehicles, each with the same given capacity, a set of customers (nodes in a graph) each with a certain demand, and a depot (the starting node). The goal of the CVRP is to calculate the total minimum distance travelled so that all the trucks are used to satisfy the needs of the customers, where each truck cannot carry more than its capacity. While there are a number of different VRPs, for example, the Periodic Routing Problem, Inventory Routing Problem and Multi-trip Vehicle Routing Problems, this paper will be focusing on solving the CVRP. Since the VRP is such a major area of optimisation research many different algorithms have been developed to try and solve it[11]. Among these algorithms include single solution-based algorithms like Simulated Annealing and Tabu search and population-based algorithms. In the category of population-based algorithms, there are swarm-based algorithms and evolutionary-based algorithms. Differential Evolution, which lies in the evolutionary-based category, was introduced in 1997[22]. After being introduced DE gained a large amount of popularity due to its simple nature and impressive ability to solve problems [19]. One such problem being the VRP.

The DE as introduced by Storn et al.[22], is a heuristic approach capable of finding global optima with very few control variables, ease of use and lends itself well to parallel computation. In its original formulation, it is used to minimize nonlinear, non-differentiable continuous space objective functions. The part of DE which makes it unique is its implementation of mutation, which uses the difference between solution vectors and a factor F to generate a new solution vector. This has proven to work very well for solving problems where the gradient of the problem isn't known beforehand. DE also uses a crossover method with swaps the values of the mutated solution with an initial solution based on a crossover probability CR. These are the only 2 control variables required for DE, which makes it a lot simpler to optimise.

One of the difficulties of solving the VRP using DE is that the general formulation of the VRP uses a permutation-based encoding scheme[7], whereas DE requires a continuous objective function. In this paper, I look at implementing a commonly used encoding for the VRP for evolutionary algorithms[13] and then implementing a version of the DE to solve it using an updated mutation function.

2 LITERATURE REVIEW

In this section, I hope to look at the literature surrounding the CVRP, DE algorithm and methods for solving the CVRP using DE.

The CVRP as mentioned previously was introduced in 1959 by Dantzig et al.[7] as the "Truck Dispatching Problem". Then 5 years later it was generalised to a linear optimisation problem by Clarke et al. [6]. This generalisation is often encountered in the domain of transportation and logistics. The problem aims to find the shortest path a group of trucks, with set capacities, could take to serve a set of customers and then return to the starting position or depot. Since its inception, the VRP has become one of the most widely researched problems in the field of optimisation.

While a very basic instance of the VRP may incorporate the TSP, the addition of further real-world parameters and scenarios to the VRP has led to a multitude of different versions of the VRP, many of which are immensely different from the original problem. With the addition of these variations comes an increase in complexity. The VRP has already been proven to be NP-Hard[17] so any new variations will only be harder to solve.

Since the VRP is NP-Hard, only small problems can be solved via exact solvers. A solution to this is the use of heuristic or metaheuristic algorithms which aim to approximate an optimisation problem in a relatively shorter period of time. Since real-world applications of VRPs are often very large[14][9], this method of finding solutions is the most popular.

As stated in the review of VRP variations and classifications by Braekers et al.[5], the rate of literature on the according to Eksioglu et al. [10], the rate that literature has been growing exponentially at a rate of 6% each year. This is most likely due to its number of industrial applications and possible variations. Due to this increase in the amount of research though, the number of surveys covering the different variations is diminishing. The review by Braekers et al.[5] classifies a large variety of variations, however, it is starting to get relatively outdated. While literature reviews like Sar et. al[20] offer more recent coverage of the problem for a specific application,

*

, ,
.

a more up-to-date general survey would help to determine what variations are currently most widely used and what papers there are for each variation.

As mentioned before heuristics and metaheuristics that solve the VRP fall into 2 categories, single-solution-based methods and population-based methods. Within the category of single-solution-based methods are algorithms like Simulated Annealing and Tabu Search, which take a single solution and then traverse the search space trying to find a better solution. These algorithms also employ methods that vary whether a better solution is accepted based on a number of factors to avoid getting trapped in local optima [3][23].

The population-based methods use a population of solutions to try and cover a greater area of the search space in each iteration so that there is a higher chance of finding a solution close to the minimum. Within the population-based methods are swarm-based algorithms[4] and evolutionary algorithms[1]. Swarm-based algorithms try to use the collective behaviour of agents in the population to converge upon an optimal solution. Evolutionary algorithms take a population of potential solutions and try to update the population so that good solutions in the initial population are used to generate potentially better solutions in the new population.

Differential Evolution (DE) is one of the algorithms in the evolutionary category. Developed by Storn et al. [22] in 1997, it offers a simple method for solving optimisation problems with continuous search spaces. DE also benefits from having a low number of control parameters (only 2 as stated previously). A survey by Das et al.[8] great extent of the literature on the DE. In the survey, a number of different variations of the DE are discussed, with different implementations of the mutation and crossover functions in order to solve a range of problems.

A number of different methods have been proposed for solving the VRP using DE. As mentioned before the standard formulation of the VRP [7] uses a permutation-based encoding, whereas the DE requires a continuous search space[22]. The first method uses mixed integer programming to formulate the VRP with backhauls and time windows[16]. This method was shown to solve the VRP and was compared to 4 discrete DE algorithms on CVRP in [21].

In 2012 Liu et al.[18] solved VRP with time windows using a memetic DE algorithm. In this algorithm, a source vector was translated to a solution vector by a number of modifications, one of which was the insertion of the sub-tour separator "0". This solution was then optimised by a set of 3 local search algorithms. In the paper, the author performed a number of experiments which showed the variation to the algorithm improved the quality of the solutions. The new algorithm was also shown to be especially suitable for instances with clustered locations.

A considerably different approach, introduced by Xu et al.[24], used a non-traditional multi-objective variation of the DE algorithm to solve the unidirectional logistics distribution VRP with no time windows. In this approach, they introduced an encoding scheme that mapped a real-valued candidate vector to a routing of k vehicles.

Kromer et al. [15] introduces a version of the DE that simultaneously looks for an optimal set of routes and minimizes the number of vehicles needed. In this paper, the algorithm proposed is used to solve the SVRPSP[2] variation of the VRP. They also benchmark the algorithm on different CVRP instances as well.

Lastly, in 2013 Huo et al. [12] proposed a new discrete variation of the DE algorithm for stochastic VRPs with simultaneous pickups and deliveries. The algorithm used a relatively standard encoding for the VRP, with integer values and "0" separating each route. The fitness function in this paper incorporates the objectives of the problem and constraints. Apart from the fitness function the paper also uses a different mutation and crossover function that applies a bitwise operator to mutate the solution. A "revise" operator is then used to eliminate any illegal chromosomes in the trial solution.

3 IMPLEMENTATION

I would like to note that no matter how hard I tried Latex refused to display my algorithm pseudo-code. I, therefore, apologise for the lack of it and I've tried to comment and describe the code as thoroughly as possible in the Jupyter notebook to account for this.

3.1 Solution Encoding

For my implementation of the DE algorithm for solving the VRP, I decided to use the common direct encoding scheme referenced here [12]. This encoding scheme starts the solution with a "0" and then the route for each vehicle is separated by a "0" with the solution finally ending with a "0". An example of a solution using this encoding is given below:

- depot→customer(1)→customer(4)→depot = [0,1,4,0]
- depot→customer(1)→customer(4)→depot → customer(2)→customer(3)→customer(5)→ depot = [0,1,4,0,2,3,5,0]

3.2 Check Solution Validity

While the majority of constraints are handled by the encoding of the solution it is still possible for a mutated solution to break some constraints. I decided that it would be more beneficial to the algorithm to apply penalties instead of hard coding the constraints so that the algorithm is able to search more of the solution space in the hopes of finding a better solution. I, therefore, implemented the check valid constraints function to count the number of constraints broken in a solution. These constraints include whether the vehicle is not being used and whether a route hasn't ended.

3.3 Fitness

The fitness function in my implementation calculates the total distance travelled over all routes and then adds a Lagrange parameter to the fitness for each constraint that's been broken. The Lagrange parameter helps to keep the search space smoother. The equation for the Lagrange function is given below, I've recently been using this value for solving the TSP and found it to be a good value for routing problems.

$$\text{Lagrange} = \frac{\text{sum of distances} * \text{total nodes}}{\text{total edges}}$$

3.4 Differential Evolution

The main principles of my DE algorithm work the same as the standard DE algorithms. It generates a population, performs a mutation on each member of that population, performs crossover between each member and their mutated child solution, and then it uses a

greedy population update method where only better solutions are accepted.

3.5 Mutation and Crossover

Firstly I decided to implement the crossover and mutation functions in one. This is due to the crossover and mutation only being applied to individual elements meaning that only if the element is selected for crossover does it need to be mutated.

To maintain the integrity of the solution encoding I have implemented a swap function as the mutation method. Firstly this function removes the 0 from each side of the solution so they aren't changed. This function then uses the method usually used for generating the solution element value to instead select an index from the solution. The formula used to generate this index is as follows:

$$\text{int}(v_{i,G+1} = x_{r1,G} + F \cdot (x_{r2,G} - x_{r3,G}) \mod \text{solution\_length})$$

The int function just rounds the value to an integer so that it's a valid index. Using this method each element in the solution is iterated over and if the element is crossed over it swaps the elements and then performs a crossover. This allows the trial solution to be some neighbour of the initial solution.

4 TESTING

In this section, I performed 4 tests. The first 2 are to determine the effect of the CR and F values on the DE algorithm. The third one uses a parameter tuner to find the best value of CR and F. Finally the fourth one runs the problem on a random search and stochastic hill climber algorithm to compare the results of the DE too.

After looking through the literature I found two datasets often used for CVRP instances when solving using DE-type algorithms. These can both be found in CVRPLIB. The first dataset is "Set A (Augerat, 1995)" and the second dataset is "Set P (Augerat, 1995)".

To download and interpret these datasets I used the vrplib python module. I've included all instances used alongside this notebook so the following function can be used to load them in. I should also note that not all CVRP problems on CVRPLIB have coordinates so if that is not the case the coordinates are randomly generated. Doing this won't affect the solution but the graph edge weights won't directly correspond to the length of the edge.

Also since external libraries are not allowed, I used the vrplib python library to download and convert the instances to dictionaries which I'm now reading in using pythons standard pickle library.

4.1 Test 1

The first test involved below involves running the algorithm for different values of CR. CR must be between 0 and 1 and based on a few preliminary tests I found it to work best the smaller the value. I, therefore, decided to test the values 0, 0.1, 0.2, 0.25, 0.5, 0.75 and 1. Each of these tests was conducted 4 times and the average was calculated. The population size was equal to dimension. The number of generations was 3000.

The results for this test are given below:

Table 1: Testing CR Value

Instance	CR	Time(s)	Best Solution	Fitness	Fitness Error(%)
	0.000000	24.682669	450.000000	528.250000	17.388889
	0.100000	27.311374	450.000000	538.250000	19.611111
	0.200000	29.654758	450.000000	620.250000	37.833333
P-n16-k8	0.250000	30.169260	450.000000	716.000000	59.111111
	0.500000	34.323817	450.000000	806.000000	79.111111
	0.750000	38.161891	450.000000	815.750000	81.277778
	1.000000	37.595295	450.000000	1088.750000	141.944444

Table 2: Testing F Value

F		Time(s)	Best Solution	Fitness	Fitness Error(Instance
	0.100000	25.103940	450.000000	608.000000	35.111111
	0.200000	24.387844	450.000000	684.250000	52.055556
	0.300000	24.396876	450.000000	683.500000	51.888889
	0.400000	24.432986	450.000000	680.250000	51.166667
	0.500000	24.395875	450.000000	685.500000	52.333333
	0.600000	24.287070	450.000000	607.000000	34.888889
	0.700000	24.313432	450.000000	609.000000	35.333333
P-n16-k8	0.800000	24.471735	450.000000	679.500000	51.000000
	0.900000	24.385294	450.000000	684.750000	52.166667
	1.000000	23.941138	450.000000	606.000000	34.666667
	1.100000	24.161986	450.000000	755.250000	67.833333
	1.200000	24.375428	450.000000	680.250000	51.166667
	1.300000	24.532908	450.000000	605.250000	34.500000
	1.400000	24.675777	450.000000	603.500000	34.111111
	1.500000	24.938721	450.000000	685.000000	52.222222

4.2 Test 2

The second experiment hopes to look at the effect of different F values on the DE. In the previous test, I showed that a CR value of 0 worked best, so that is the value we will be using in this experiment. The other values are all identical to the previous experiment as well. In this experiment, I'll use F values between 0.1 and 1.5 with intervals of 0.1. This is due to the mutation function wrapping round, which makes having a large value of F relatively pointless and F must be greater than 0.

The results for the test are given in Table 2.

4.3 Test 3

In the previous experiments, I used rough values of CR and F to gauge the effect on the DE. To narrow these parameters down even further I will use a hyperparameter tuner. Hyperparameter tuners use a variety of different methods to help find the best parameters for an algorithm. In this test, I will be using Optuna a well-known hyperparameter tuner, which uses machine learning to optimise the parameters. Apart from CR and F the other parameters will stay the same as the previous experiments.

The results for this test are as follows:

- *CR:* 0.04217962751716662
- *F:* 0.5378005850476446

4.4 Test 4

Using the parameters from the previous experiment I will now test my DE implementation on 5 different CRVP instances. These will be the first 3 instances from "Set A (Augerat, 1995)" and the first 2 from "Set P (Augerat, 1995)". Doing so allows me to compare the

Table 3: Testing Different Instances

	Time(s)	Best Solution	Fitness	Fitness Error(Instance
A-n32-k5	57.362062	784.000000	1104.000000	40.816327
A-n33-k5	59.998449	661.000000	990.750000	49.886536
A-n33-k6	63.175768	742.000000	1079.000000	45.417790
P-n16-k8	27.539934	450.000000	681.750000	51.500000
P-n19-k2	22.443003	212.000000	226.750000	6.957547

Table 4: Testing Greedy Algorithms

Instance		Time(s)	Best Solution	Fitness	Fitness Error(Algorithm
	A-n32-k5	0.605577	784.000000	1673.750000	113.488520
	A-n33-k5	0.641061	661.000000	1399.000000	111.649017
Random Search	A-n33-k6	0.648487	742.000000	1451.250000	95.586253
	P-n16-k8	0.626924	450.000000	492.000000	9.333333
	P-n19-k2	0.385771	212.000000	366.000000	72.641509
	A-n32-k5	0.608620	784.000000	3603.483871	359.628045
	A-n33-k5	0.648799	661.000000	5539.656250	738.072050
Stochastic Hill Climber	A-n33-k6	0.671159	742.000000	6118.937500	724.654650
	P-n16-k8	0.645485	450.000000	1339.750000	197.722222
	P-n19-k2	0.397854	212.000000	1067.138889	403.367400

algorithm with a couple of different papers. As with the previous tests the maximum number of generations will be 3000.

The results for the test are given in Table 3.

4.5 Test 5

For experiment 5, I've created a random search and stochastic hill climbing algorithm to benchmark the solutions found by the DE algorithm. These algorithms utilize the population generation function from the DE and the fitness function from the DE so that they're searching the same search space. The stochastic hill climber algorithm uses the notion of swapping 2 values in the solution to act as a step.

The results for the test are given in Table 4.

5 ANALYSIS OF RESULTS

5.1 Test 1

In test 1 we can clearly see that the value of CR has a significant impact on the ability of the algorithm to find solutions. We can conclude from this test that the value of CR should be around 0 to 0.1.

5.2 Test 2

In test 1 we can clearly see that the value of F appears to have little impact on the final ability of the algorithm to find a solution. This is probably due to the implementation of the mutation and crossover function.

5.3 Test 3

Here we again found that the lower value of CR helps the algorithm to find better solutions. In this test, we found optimal values of CR and F for use in test 4.

5.4 Test 4

In test 4 we can see that on average the algorithm was able to find a solution within between 6.9% to 51.5% of the optimal solution in only 3000 iterations.

5.5 Test 5

In test 5 we can see that in all but 1 case the random search and hill climber algorithms were not able to find a solution as close to the optimal as the DE algorithm. This proves that my implementation of the DE algorithm was in fact able to beat both Stochastic Hill Climber and Random Search.

6 CONCLUSION

In conclusion the project showed an implementation of the DE algorithm which was able to solve the VRP better than both the Random Search and the Stochastic Hill Climber Algorithm.

REFERENCES

[1] Thomas Back. 1996. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press.

[2] Eshetie Berhan, Pavel Krömer, Daniel Kitaw, Ajith Abraham, and Václav Snášel. 2014. Solving stochastic vehicle routing problem with real simultaneous pickup and delivery using differential evolution. In *Innovations in Bio-inspired Computing and Applications: Proceedings of the 4th International Conference on Innovations in Bio-Inspired Computing and Applications, IBICA 2013, August 22-24, 2013-Ostrava, Czech Republic*. Springer, 187–200.

[3] Christian Blum and Andrea Roli. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)* 35, 3 (2003), 268–308.

[4] Eric Bonabeau, Marco Dorigo, Guy Theraulaz, and Guy Theraulaz. 1999. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press.

[5] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. 2016. The vehicle routing problem: State of the art classification and review. *Computers & industrial engineering* 99 (2016), 300–313.

[6] Geoff Clarke and John W Wright. 1964. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research* 12, 4 (1964), 568–581.

[7] George B Dantzig and John H Ramser. 1959. The truck dispatching problem. *Management science* 6, 1 (1959), 80–91.

[8] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N Suganthan. 2016. Recent advances in differential evolution–an updated survey. *Swarm and evolutionary computation* 27 (2016), 1–30.

[9] Michael Drexl. 2012. Rich vehicle routing in theory and practice. *Logistics Research* 5 (2012), 47–63.

[10] Burak Eksioglu, Arif Volkan Vural, and Arnold Reisman. 2009. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering* 57, 4 (2009), 1472–1483.

[11] Raafat Elshaer and Hadeer Awad. 2020. A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers & Industrial Engineering* 140 (2020), 106242.

[12] Lingjuan Hou, Zhijiang Hou, and Hong Zhou. 2012. Application of a novel discrete differential evolution algorithm to svrp. In *2012 Fifth International Joint Conference on Computational Sciences and Optimization*. IEEE, 141–145.

[13] Lingjuan Hou and Hong Zhou. 2010. Stochastic vehicle routing problem with uncertain demand and travel time and simultaneous pickups and deliveries. In *2010 Third International Joint Conference on Computational Science and Optimization*, Vol. 1. IEEE, 32–35.

[14] DGND Jayarathna, GHJ Lanel, and ZAMS Juman. 2022. Industrial vehicle routing problem: a case study. *Journal of Shipping and Trade* 7, 1 (2022), 6.

[15] Pavel Krömer, Ajith Abraham, Václav Snášel, Eshetie Berhan, and Daniel Kitaw. 2013. On the differential evolution for vehicle routing problem. In *2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR)*. IEEE, 384–389.

[16] İlker Küçükoğlu and Nursel Öztürk. 2014. A differential evolution approach for the vehicle routing problem with backhauls and time windows. *Journal of Advanced Transportation* 48, 8 (2014), 942–956.

[17] Jan Karel Lenstra and AHG Rinnooy Kan. 1981. Complexity of vehicle routing and scheduling problems. *Networks* 11, 2 (1981), 221–227.

[18] Wanfeng Liu, Xu Wang, and Xia Li. 2012. Memetic differential evolution for vehicle routing problem with time windows. In *Advances in Swarm Intelligence: Third International Conference, ICSI 2012, Shenzhen, China, June 17-20, 2012 Proceedings, Part I 3*. Springer, 358–365.

[19] VP Plagianakos, DK Tasoulis, and Michael N Vrahatis. 2008. A review of major application areas of differential evolution. *Advances in differential evolution* (2008), 197–238.

[20] Kubra Sar and Pezhman Ghadimi. 2023. A Systematic Literature Review of the Vehicle Routing Problem in Reverse Logistics Operations. *Computers & Industrial Engineering* (2023), 109011.

[21] Andre Luis Silva, Jaime Arturo Ramírez, and Felipe Campelo. 2013. A statistical study of discrete differential evolution approaches for the capacitated vehicle routing problem.. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. 77–78.

[22] Rainer Storn and Kenneth Price. 1997. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization* 11, 4 (1997), 341.

[23] El-Ghazali Talbi. 2009. *Metaheuristics: from design to implementation*. John Wiley & Sons.

[24] Huan Xu and Jiechang Wen. 2012. Differential evolution algorithm for the optimization of the vehicle routing problem in logistics. In *2012 Eighth International Conference on Computational Intelligence and Security*. IEEE, 48–51.