Center for Research and Advanced Studies
of the National Polytechnic Institute

Zacatenco Unit
Mathematics Department

Optimization problems by routing swarms of drones

Thesis presented by:

Daniel Gutiérrez Espinoza

To obtain the degree of

Master in Science

In the Specialty of Mathematics

Thesis advisor:
Dr. Isidoro Gitler Goldwain

Mexico City. July 2023.

Centro de Investigación y de Estudios
Avanzados del Instituto Politécnico Nacional

Unidad Zacatenco
Departamento de Matemáticas

Problemas de optimización al rutear enjambres de drones

Tesis que presenta:

Daniel Gutiérrez Espinoza

Para obtener el grado de

Maestro en Ciencias

En la especialidad de Matemáticas

Director de Tesis:
Dr. Isidoro Gitler Goldwain

Ciudad de México. Julio 2023.

Datos del jurado

Dr. Isidoro Gitler Goldwain
Departamento de Matemáticas
Cinvestav, México.

Dr. Ruy Fabila Monroy
Departamento de Matemáticas
Cinvestav, México.

Dr. Feliú Davino Sagols Troncoso
Departamento de Matemáticas
Cinvestav, México.

Abstract

This work is an exposition of the mathematical problems present in routing drones in relation to the package delivery problem. Two main optimization problems are addressed: firstly, the traveling salesman problem, which is the essence of any vehicle routing model (of any kind), and secondly, the clustering problem. For each of these problems, the classic strategy for solving the problem is presented, as well as an alternative proposal. Finally, a modern and realistic package delivery model of a truck that utilizes drones to deliver packages. In all cases, a worst-case analysis is performed.

Resumen

El presente trabajo es una exposición de los problemas matemáticos presentes en el de ruteo de drones con respecto al problema de entrega de paquetes. Se abordan dos problemas principales de optimización: primero, el problema del agente viajero, el cual es la esencia de cualquier modelo de ruteo de vehículos (de cualquier tipo) y segundo, el problema de agrupamiento (también llamado clusterización). Para cada uno de estos problemas se presenta la estrategia clásica para resolver el problema, así como una propuesta alternativa. Finalmente, se presenta un modelo moderno y realista de una camioneta que utiliza drones para realizar las entregas de paquetes. En todos los casos se realiza un análisis del peor caso.

Para los tesistas perpetuos,

que mi trabajo sirva de inspiración

para que terminen pronto.

Agradezco al Consejo Nacional de Ciencia y Tecnología (Conacyt) por haber financiado este trabajo.

Agradecimientos

Primeramente, debo de agradecer al Dr. Isidoro Gitler por todo el apoyo que me brindó para la realización de este trabajo y por todo el tiempo que dedico a la revisión del mismo. Es gracias a ello que ahora este trabajo puede ver la luz. También debo agradecerle por toda la asistencia que me proporcionó durante estos dos años que realice mis estudios de maestría.

A mis sinodales, el Dr. Ruy y el Dr. Feliú, por la revisión que hicieron del presente trabajo, así como por sus enseñanzas durante los cursos que tome con ellos. Curiosamente son también los dos primeros profesores que tuve durante mis estudios de maestría y quizás no por coincidencia, la escuela de combinatoria en el Cinvestav es pequeña pero muy fuerte.

También debo agradecer a aquellos profesores que me apoyaron desde mis inicios en este camino de las matemáticas: al Mtro. Loreto Cruz quien finalmente me convenció de ser matemático, al Dr. Alejandro Illanes por todas las enseñanzas y sobre todo por la gran inspiración que pude tomar de él y finalmente al Dr. Vinicio Gómez por todo el apoyo que me proporcionó desde mis estudios de licenciatura y que se ha extendido hasta la maestría.

A mi mamá por todo el apoyo que me dio para que yo pudiera realizar este proyecto, a mi tío Israel por siempre haber estado al pendiente de nosotros y a mi Tita que nos mira desde arriba.

A mi amigo José Alberto Serrano por seguir aquí conmigo y a Lizbeth Durán por seguir dándome problemas.

Contents

Introduction

Unmanned Aerial Vehicles (UAV), now better known as drones, were a technology previously only available for military applications. This trend has changed in the last decade and currently the interest in civilian applications has surpassed the interest in military applications, at least in academia.

Probably this trend started because of the massive introduction to the market of low-cost drones. We identified at least six civilian applications of drones: package delivery, surveillance, agriculture, topography, telecommunications, and cinematography. Surveillance and package delivery are the most researched areas covering 90% of the published articles [27].

In this work we address the drone package delivery problem. We selected that application, mainly, because it is the first one that people will notice on a daily basis, faster deliveries are more common every day, now with some companies even offering one-hour delivery [35].

Despite the many economic, social and political impacts of drone package delivery, the present work will be purely mathematical. We will study the mathematics that are behind the drone-truck package delivery problem, we will give a deep insight into the hidden mathematics that lay behind, and we will exhibit the mathematical techniques used in this type of problems. Which are, fundamentally, two NP-hard optimization problems: a TSP optimization problem and a clustering optimization problem.

The structure will be to first present the classical, most widely used, technique to solve a given problem and later propose an alternative technique for the same problem.

Chapter 1: The Traveling Salesman Problem

In the first chapter we will study our first optimization problem, the Traveling Salesman Problem. Although being a several centuries old problem, it is still a research problem as it is at the core of any vehicle routing problem, in particular, at the core of any drone package delivery problem.

Although there are even complete books about the TSP [9], no work can be as complete as including everything about the TSP. Still, we try to give a comprehensive introduction by tackling a bit about the history of the problem from the early beginnings, the different types of TSP problems (all of which are commonly referred as the TSP, which has been a source of great confusion).

We will provide the first proposed optimization formulation of the problem, which is still one of the most popular, and introduce the classical nearest neighbor method to solve the problem, which is still the most popular method (at least as a benchmark to compare new methods).

The alternative we will present will not be Christofides algorithm (or any of its new improved variants [8])

That is because their complexity is bigger than $O(n^2)$, (initially $O(n^3)$, but it has been reduced). This justification can be understood because delivery companies need a fast algorithm rather than a low bound algorithm that takes hours to run (think of the same day or one-hour deliveries). Hence, in practice, Christofides is not very popular [9].

For this reason, we present a Greedy algorithm with a $O(n^2)$ running time, which in addition, works for many variants of the TSP such as the asymmetrical version and the TSP without triangle inequality. Working with a TSP without triangle inequality is important, especially in relation to trucks (or cars), because of the car traffic and other circumstances found in cities (car accidents, etc.) that make the triangle inequality not valid in real life. We will show that there is no ρ-approximation algorithm, $\rho \in \mathbb{R}$, for the TSP without triangle inequality which supports the argument that the greedy algorithm is a fast and versatile algorithm.

Chapter 2: Clustering

In this chapter we address another NP-hard problem [12]. Clustering is considered an unsupervised machine learning method, that means that its purpose is to find some unknown hidden features among some given data objects. A clustering task aims to partition the data so that closely related data objects are partitioned together and in a different partition to unrelated objects.

The data objects can be of any type and no assumption can be made, such as supposing they live in some Euclidean space or that they have a metric. Therefore, the partition that is being sought can have many forms, for instance, it could happen that it is not possible to give a visual representation. The study in this work will not be as general as to consider those cases, even if there is not a metric (for example, if there is no triangle inequality) we will assume there is some type of proximity function between data objects, or at least between clusters. Ultimately the purpose of the first part is to show that there are alternatives to the sum of the squared error optimization function that we will present (and with which we will work for the rest of the work).

Later, we will present the k-means algorithm which is the most widely used clustering algorithm in the literature for vehicle routing problems [1]. We will prove that the algorithm constructs some local optimal solutions, but its random initialization nature can generate clusterings as bad as wanted.

We will present an algorithm that has gained popularity just recently [32], the k-means++ algorithm, and maybe the presented worst-case analysis will clarify why we say that the k-means is easy to understand.

It can be proved that the k-means++ algorithm has an expected worst case of $\Theta(\ln k)$, however for this work we will not delve so deeply and instead we will just prove the main bound, that is, show that the expected worst case is $O(\ln k)$. However, for the reader eager for more worst-case analysis, we prepared many interesting results for the next chapter.

Chapter 3: Drone package delivery

In this last chapter we present a modern drone-truck delivery model, but before presenting the model, we discuss several worst-case results of drone-truck package delivery.

We highlight the lack of mathematical theoretical articles about drone-truck package delivery since most of the published articles just present some particular models and computer assisted heuristics to solve those particular problems which, probably, cannot be applied to other types of problems.

On the other hand, the worst case results we present are as general as possible in a very simple model with as few assumptions as possible. Perhaps one of the most interesting results, that arises as a consequence of one of the proved theorems, is that even if we know that a single truck making deliveries should be slower than m trucks making deliveries simultaneously to the same set of clients, the reverse bound is not that obvious, one can imagine that since the trucks are working simultaneously they may do the work at least m times faster, but that is not the case and we prove that they actually do the same job at most m times faster. Finally, we prove that a fleet of m trucks, each with k drones, delivers to a set of clients at most $k+1$ times faster and that the bound is tight.

In the last part we give a brief discussion about the current constraints that drones encounter in today's world and present a modern and realistic model which considers most of those current known constraints, the Parking Location and TSP with Homogeneous Drones (PLTSPHD).

The problem is solved in two parts: first a clustering problem is solved and later an optimization TSP is solved by using a Deep Reinforcement Learning approach (which we omit), however we analyze the experimental results which further support our assertion about the great potential that drones represent.

Overall, we have made our best effort to keep this work self-contained, for which we have included some long forgotten results, which explain the inclusion of some old bibliography. Also, we have avoided the current trend of quoting old results by referencing new articles which quote the results, thus making the results appear more modern. Instead, we always reference the original work. No prerequisite is assumed to understand this work.

1. The Traveling Salesman Problem

TSP

Probably the most famous problem in combinatorial optimization is the Traveling Salesman Problem (TSP). The problem is simple to understand: a salesman who needs to travel through a set of n cities, what is the shortest route he can take in terms of distance if at the end he needs to return to the starting position, and he must visit each city?

The origin of the problem is still a mystery. It can be tracked to a German book from 1832 which was indeed written by a salesman! In the book the author provides some routes to be used by a salesman willing to travel through regions of Germany and Switzerland [4]. However, people have faced the problem in some way or another since ancient times.

Before mathematicians studied the problem in the early XX century, the usual way to solve the problem was to put pegs on a map for each of the locations to visit, then with a string create routes through the pegs manually. Incredibly, that was the first method to be used by the first mathematicians who attacked the problem (Dantzing, Fulkerson, and Johnson) however they did not admit it until later.

Before presenting the formulation of the TSP, it is convenient to review some topics about complexity theory.

Complexity theory:

deterministic

In the literature there are many versions of the TSP, different authors use the same term for quite different problems. Before we talk about the different combinatorial versions of the TSP, we need to understand what we mean by an algorithm. Formally, in the theory of computation, an algorithm is a systematic method for solving a problem that can be implemented by a Turing machine [29]. Informally, we will refer to an algorithm as a sequence of instructions that can be carried out deterministically to solve a problem. And by *deterministic* we mean that every time the algorithm receives the same input it will provide the same output.

Turing machines are theoretical models and thus assumed to have unlimited power, therefore, memory and computing time are of no concern. In the real world we do need to take them into account.

P
NP
NP-hard
NP-complete

A common method to classify problems is to say that a problem is in *P* if we know an algorithm to solve the problem in polynomial time. We say that a problem is *NP* if given a solution candidate, we can verify it in polynomial time. In addition, we say that a problem is *NP-hard* if any problem in NP can be reduced to it in polynomial time, for this reason, problems in this category are said to be at least as hard as any NP problem. Finally, we say that a problem is *NP-complete* if the problem is NP and NP-hard.

In general, if a problem is classified as NP-hard that does not mean that there is no algorithm that can solve the problem in polynomial time. Instead, it just means currently we do not know any such algorithm, but if for a NP-hard problem we discovered any such algorithm, then we would be able to solve every NP problem in polynomial time (but not necessarily every NP-hard problem).

In summary, finding an algorithm to solve a NP-hard problem in polynomial time would be tremendously difficult because that would solve, automatically, any NP problem in polynomial time.

A different classification

There are many ways to classify problems, we already presented one. A different type of classification is related to the type of output that is expected as a solution. Given some input a problem may ask for a binary solution, that is, for a yes or a no. Such problems are known as *decision problems*. Other problems may ask for the solution to be an instance that holds some properties, those problems are known as *search problems*. Finally, another class of problems might ask to find a best solution from a collection of feasible solutions, such problems are known as *optimization problems*.

We already defined, informally, what the TSP problem is, however among different authors the TSP problem means completely different things. For some authors the TSP problem is a decision problem, that is, given a distance k the problem asks if there is a route that the salesman can take (that visits every city and returns to the origin) of distance k. Hence, the expected output of the problem is simply a yes or a no. For other authors, the TSP is a search problem that given k asks to provide a route of distance k that the salesman can take or indicate that no such route exists.

It is easy to see that both the decision problem and the search problem of the TSP are NP, if we are given a candidate solution we can verify, in polynomial time, that it is a solution of the expected length. In fact, it is a well-known fact that both problems are NP-complete.

Other authors consider that the TSP problem is an optimization problem, to find the minimum possible length of a route that visits all the cities only once and returns to the origin. Some other authors refer to this problem as the TSP-O or the TSP-opt. In this work, unless otherwise stated, the TSP will be a search problem that looks for a route of optimal length. We will follow this convention for historical reasons, since it is the way it was formulated originally almost two hundred years ago.

We should make a final remark, this optimization version of the TSP, with which we are going to work, is not NP. That is, given a solution we do not have any (known) way to determine in polynomial time if it is an optimal solution. The only way we could ensure that it is an optimal solution is if we compute all the feasible solutions and compare them. We do not know how to do that in polynomial time.

However, this can serve as an example to show that if a problem is not NP that does not mean that it is harder than the NP problems. We can bound the maximum length of any route that the salesman can take by multiplying the maximum length between any two cities α (from a list of n cities) by n. Then, the maximum length of any route that the salesman can take is less or equal than αn. Now, if we had an algorithm that could solve the search version of the TSP in polynomial time $\mathcal{T}(n)$, we could do a binary search to find the optimal solution. A binary search costs $O(\log_2(\alpha n)) = O(\log_2 n)$, hence the optimization problem would cost in total $O(\log_2 n \cdot \mathcal{T}(n))$ which is also polynomial. In summary, the search version of the TSP, which is a NP-complete problem, is at least as hard as the optimization version of the TSP (which is not even NP), or in other words, even though the optimization version of the TSP is not NP, it is not harder than the search version which is NP-complete.

That was just an example to show that not being NP does not mean that it is a harder problem than the NP problems, there might be problems for which this occurs, but it is not a general rule.

Since NP problems are difficult to solve, we will rely on methods that do not solve them but provide solutions that are close to the optimal solution. Such methods are known as *approximation algorithms,* and we will say that any such method is *k-optimal*, or that it is a *k-approximation*, if the value of the provided solution is not more than *k* times the value of the optimal solution (with *k* being a real number or a polynomial). Finally, there will be problems for which the presented methods will not even be able to provide a (polynomial) approximation guarantee, for example, the problem may ask for a minimum structure of some type, and the method will just provide a solution with the structure, but we will not know how far the solution is from the sought minimum solution. We will call such methods *heuristics*. These distinctions between the algorithms, approximation algorithms, and heuristics are important because only the algorithms solve the problems. As we have said, we will not try to present algorithms to solve the NP-hard problems that we will present. Instead, we will rely on approximation algorithms and heuristics. Once that is understood, we will abuse the notation and refer to algorithms as *exact algorithms* and use the term algorithms broadly (and informally) to refer to heuristics, approximation algorithms, and exact algorithms as is common in literature.

k-optimal
approximation

heuristic

exact algorithm

The Assignment Problem

We will present the optimization formulation of the TSP proposed by Dantzig, Fulkerson, and Johnson constructively. The first step is to present a preceding model known as the *LP relaxation of the TSP* or simply as the *Assignment problem*. Then, we will explain the drawback of such a model.

We will represent the cities that the salesman must visit as the vertices of a complete graph $\mathcal{G} = (V, E)$, and use the following definition:

tour

Definition 1.1. *Given a graph $\mathcal{G}$ we call a closed walk $\mathcal{T}$ a tour or a Hamiltonian cycle if $\mathcal{T}$ contains every vertex of $\mathcal{G}$ exactly once.*

The Assignment Problem

Let $\mathcal{V} = \{1, 2, \ldots, n\}$ be a set of vertices, let $i, j \in \mathcal{V}$, and let $x_{i,j}$ be the indicator function defined by:

$$x_{i,j} = \begin{cases} 1 \text{ if the saleman travels from } i \text{ to } j \text{ for } i \neq j. \\ 0 \text{ otherwise.} \end{cases}$$

For $i, j \in \mathcal{V}$ let $d(i, j)$ denote the distance from i to j. Consider the following objective function:

$$\min\ Z = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{i,j} d(i, j)$$

$$\text{s.t.}$$

$$\sum_{j=1}^{n} x_{i,j} = 1 \text{ for all } i \in \mathcal{V}.$$

$$\sum_{i=1}^{n} x_{i,j} = 1 \text{ for all } j \in \mathcal{V}.$$

Solving the assignment problem is a challenging task, in principle there are $(n-1)!$ feasible solutions. Further, solving the assignment problem does not solve the TSP because the solution might be composed of several independent cycles instead of exactly one cycle. Hence, some additional constraints must be enforced to the problem.

There are many optimization formulations of TSP, one of the most famous is the *Dantzig–Fulkerson–Johnson* formulation *(DFJ)*: *DFJ*

Dantzig–Fulkerson–Johnson (DFJ)

The formulation uses the same objective function and constraints as the assignment problem, but it adds an additional constraint:

$$\min\ Z = \sum_{i=1}^{n}\sum_{j=1}^{n} x_{i,j} d(i,j)$$

$$\text{s.t.}$$

$$\sum_{j=1}^{n} x_{i,j} = 1 \text{ for all } i \in \mathcal{V}.$$

$$\sum_{i=1}^{n} x_{i,j} = 1 \text{ for all } j \in \mathcal{V},$$

$$\sum_{i\in S}\sum_{j\in S} x_{i,j} \leq |S| - 1 \text{ for all } S \subsetneq \mathcal{V}.$$

On closer inspection we can notice that it is not just one additional constraint but $2^n - 1$ new constraints, one per each proper subset of $\mathcal{V}$, which guarantee that there is no cycle with any proper subset of S, so that the only cycle is the one with all the vertices of S. These new $2^n - 1$ constraints make it infeasible to find exact solutions for long values of n.

Now we will study some of the earliest methods developed to handle the TSP. Despite the methods being several decades old, they are still being used today, mainly because of their simplicity and because of the good approximations they provide.

The classical TSP

We discussed previously that the meaning of the TSP problem changes from author to author, for some it is a decision problem, for others it is a search problem, and for others it is an optimization problem. It might appear that the confusion ends there, but that is not the case. Even after we have decided the type of problems with which we are going to be working, the TSP divides further depending on the type of distance that we use. There is a metric version of the TSP, an asymmetric version of the TSP, a Euclidean version of the TSP, a non-Euclidean version of the TSP, a TSP with triangle inequality, a TSP without triangle inequality, and so on. In general, there is no one classical form of the TSP, and we should always be aware of the version that a particular author is using. First, we will present the metric version.

It is natural for many mathematicians who work with the TSP to want to calculate distances by using the Euclidean distance (or some other metric). Given a space V and a function

metric

$d : V \times V \to \mathbb{R}$, we say that d is a *metric* if the following properties hold:

For any $u, v, w \in V$:

1. $d(u, v) \geq 0$.
2. $d(u, v) = 0$ iff $u = v$.
3. $d(u, v) = d(v, u)$.
4. $d(u, v) + d(v, w) \geq d(u, w)$.

ATSP

When in a TSP model (3) does not hold we say that it is an *asymmetric TSP (ATSP)*, when (4) does not hold we say that it is a *TSP without triangle inequality*. We will say that a TSP is

Euclidean

Euclidean if we can calculate the distance between any two vertices.

In this work we will work with the Euclidean version of the TSP with triangle inequality.

Since the aim of this work is to present results related to Unnamed Aerial Vehicles (UAV) and because UAV travel (fly) in a straight line it is reasonable to assume that the length of the shortest path between any two vertices is the Euclidean distance between them. That is, we assume that the problem of finding the shortest path between any two pairs of vertices is trivial so we can assume that the distance matrix of the vertices is already given. Nevertheless, it is worth noting that in cases where such an assumption is not possible the shortest path problem should be solved to construct the distance matrix. A common strategy is to use Dijkstra's algorithm which can be implemented in $O(m + n \lg n)$ with n the number of vertices and m the number of edges [30]. We consider the distance matrix to have the form:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$$

where a_{ij} is the distance from vertex i to j for $i, j \in \mathcal{V}$. Since we want to avoid loops, we write $a_{ii} = \infty$ for all $i \in \mathcal{V}$, i.e.,

$$a_{i,j} = \begin{cases} d(i, j) \text{ for } i, j \in \mathcal{V}, i \neq j. \\ \infty, \text{ otherwise.} \end{cases}$$

From these assumptions, it should be clear that any algorithm that aims to solve the TSP and provide a near optimal solution should have, at least, a $O(n^2)$ complexity which is just the time required to read all the matrix entries. The algorithms that we will present have complexities of $O(n^2)$ which is very convenient for the applications we are considering in this work, since fast solutions are a mandatory requirement.

Algorithms to solve the TSP

Nearest Neighbor

The *Nearest Neighbor (NN)* is one of the oldest methods known to solve the TSP [28]. It was developed in the mid 50s to solve ecology problems [3]. The algorithm is still in use because it is easy to understand, easy to implement, and provides a good approximation. Here we present the original version [5]: *NN*

```
Algorithm 1. Nearest Neighbor($D$):

Input: A distance matrix $D$ of a set of vertices $V = \{v_1, ..., v_n\}$.
Output: A list which corresponds to the TSP route.

1. Pick an arbitrary vertex $x \in V$ and insert it into a new path $P$.
2. Let $u$ be the last vertex of $P$, find  $w = \arg \min_{w \in V \setminus P} d(u, w)$ and insert it to $P$.
3. When $|P| = n$ insert the initial vertex $x$ to $P$ to form a cycle.
```

To be convinced that the algorithm has complexity $O(n^2)$, first we observe that steps 1 and 2 take $O(1)$ each. For step 2, if P has length k, then we need to look for the other $n - k$ vertices not in P to find w, hence it takes $n - 1 + n - 2 + \ldots + 1 = O(n^2)$.

When the algorithm first appeared, it was not known whether it provided good results or not, yet mathematicians used it anyway. Fortunately, Rosenkrantz [28] proved the following results:

Theorem 1.1. *The Nearest Neighbor algorithm is a* $\frac{1}{2} + \frac{\lceil \log_2(n) \rceil}{2}$ *approximation algorithm for the TSP.*

Given a complete graph G, let's denote by C^* the length of the optimal tour of G. Before proving the theorem, we will present two lemmas: *C^**

Lemma 1.2. *Suppose that* $\mathcal{G} = (V, E)$ *is a graph of order* n *and* l *is a function* $l : V \to \mathbb{R}$ *such that for any* $p, q \in V$*:*

(a) $d(p, q) \geq \min(l(p), l(q))$.

(b) $l(p) \leq \frac{1}{2} C^*$,

Then,

$$\sum_{v \in V} l(v) \leq \frac{C^*}{2}(\lceil \log_2(n) \rceil + 1). \tag{1}$$

Note 1.3. *We can rename the elements of the set V such that $V = \{1, 2, ..., n\}$ and such that if $1 \leq j \leq i \leq n$ then $l(j) \geq l(i)$. In the following proof we will use that assumption.*

Proof (lemma 1.2). First, we will prove that for any k such that $1 \leq k \leq n$ the following holds:

$$C^* \geq 2 \sum_{i=k+1}^{\min(2k,n)} l(i) \tag{2}$$

Let k be such that $1 \leq k \leq n$, and let $H = (V', E')$ be an induced subgraph of G with $V' = \{i \mid 1 \leq i \leq \min(2k, n)\}$.

Let T be the tour in H such that the vertices in H are visited in the same order as in the optimal tour of C^* (in case there are several tours with distance C^* we pick one of them and use it as the representative of C^*). Let's denote by L the length of T, then:

$$C^* \geq L \tag{3}$$

Let α_i be the number of edges $(i, j) = (j, i) \in T$ such that $i > j$, by Note 1.3 we have than for any such edge, $l(i) = \min(l(i), l(j))$ so α_i is actually the number of times that $l(i) = \min(l(i), l(j))$ for $(i, j) \in T$. By part (a) of the lemma, for any $(i, j) \in T$, it holds that $d(i, j) \geq \min(l(i), l(j))$, hence:

$$\sum_{(i,j) \in T} d(i, j) = L \geq \sum_{(i,j) \in T} \min(l(i), l(j)) = \sum_{i \in H} \alpha_i l(i) \tag{4}$$

We note that, since T is a tour, for any $i \in H$ it holds that $\alpha_i \in \{0, 1, 2\}$. Then we can give a lower bound to the equation in (3) if we assume than the associated α_i's of the biggest $l(i)$'s are zero, in particular, since there are at most $2k$ vertices in H we can assume that the associated α_i's of the k biggest $l(i)$'s are zero, which in turns means that the associated α_i's of the $\min(2k, n) - k$ smallest $l(i)$'s are two (note that we are not interested in determining whether this case is achievable, we are just giving a lower bound). By Note 1.3 the smallest $l(i)$'s are $\{k + 1, \ldots, \min(2k, n)\}$, then we have the following boundary to equation (4):

$$\sum_{i \in H} \alpha_i l(i) \geq 2 \sum_{k+1}^{\min(2k,n)} l(i) \tag{5}$$

Then (3), (4), and (5) together prove (2).

We proved (2) is valid for any k such that $1 \leq k \leq n$, in particular it is valid for $k = 2^j$ for any j such that $0 \leq 2^j \leq n$, that is, for $0 \leq j \leq \lceil \log_2(n) \rceil - 1$ at least.

Then by using (2) to sum through all those values:

$$\lceil \log_2(n) \rceil C^* = \sum_{j=0}^{\lceil \log_2(n) \rceil - 1} C^* \geq \sum_{j=0}^{\lceil \log_2(n) \rceil - 1} 2 \sum_{i=2^j+1}^{\min(2^{j+1}, n)} l(i)$$

$$= 2 \sum_{j=0}^{\lceil \log_2(n) \rceil - 1} \sum_{i=2^j+1}^{\min(2^{j+1}, n)} l(i)$$

since j starts in zero, and $2^0 + 1 = 2$,

$$= 2 \sum_{i=2}^{n} l(i)$$

in summary,

$$\frac{\lceil \log_2(n) \rceil C^*}{2} \geq \sum_{i=2}^{n} l(i)$$

and, by part (b) of the lemma hypothesis we know $\frac{1}{2} C^* \geq l(1)$, then,

$$\frac{(\lceil \log_2(n) \rceil + 1) C^*}{2} \geq \sum_{i=1}^{n} l(i)$$

which is the inequality (1) we wanted to prove. $\square$

Now we are ready to prove the theorem.

Proof (theorem 1.1). Let l be the function $l : V \to \mathbb{R}$ that assigns to each $v \in V$ the distance $d(v, w)$ such that w is the next vertex that the Nearest Neighbor adds to the tour construction after v. We will prove that l satisfies the conditions of Lemma 1.2.

First, to prove (a), let's take $p, q \in V$, $p \neq q$, and assume without loss of generality that p appears before q in the tour constructed by the algorithm. Then, $l(p) \leq d(p, q)$ and also $l(p) \leq l(q)$. Then $d(p, q) \geq l(p) = \min(l(p), l(q))$.

To prove that l also satisfies (b) let's take $p \in V$ and assume that q is the next vertex that the algorithm takes after v, then $l(p) = d(p, q)$.
In the tour that represents C^*, there are two independent paths P_1, P_2 that go from p to q.

By the triangle inequality

$$d(p, q) \leq d(P_1)$$

and also,

$$d(p, q) \leq d(P_2)$$

then,

$$2d(p, q) \leq d(P_1) + d(P_2) = C^*$$

so,

$$l(p) = d(p, q) \leq \frac{1}{2}C^*$$

Hence, l fulfills the requirements of Lemma 1.2. Finally, if we denote by C the length of the tour constructed by the algorithm,

$$C = \sum_{i=1}^{n} l(i)$$

by using Lemma 1.2,

$$C = \sum_{i=1}^{n} l(i) \leq \frac{C^*}{2}(\lceil \log_2(n) \rceil + 1)$$

so,

$$\frac{C}{C^*} \leq \frac{1}{2} + \frac{\lceil \log_2(n) \rceil}{2}$$

□

As the saying goes: sometimes simple is better. Previously we proved that the Nearest Neighbor is fast, now we proved that it gives a good approximation. For example, for $n = 2^{32}$ it gives a 16.5 approximation, and for the applications considered in this work $n \leq 2^{10}$ which will give 5.5 approximations. This is why these algorithms are preferred to others that give slightly better approximations but with higher complexity such as $O(n^3)$ [8].

Several variants of the Nearest Neighbor have being developed since the 60s [5], and despite the Nearest Neighbor is a greedy algorithm itself, the greedy name is reserved for a more general algorithm that works by joining at each step the smallest cost possible edge, even if we are left with independent paths (but no cycles), until we are left with a Hamiltonian path that we close to generate the final tour [9]. This algorithm is currently the most popular algorithm for the asymmetric TSP.

Greedy Algorithm (GR)

The greedy algorithm is simple to state, we present the version of [14], which as we mentioned before, is general and does not assume symmetry which means we will consider a complete bidirected graph instead.

```
Algorithm 2. Greedy ($D$):

Input: A distance matrix D of a set of vertices $V = \{v_1, ..., v_n\}$.
Output: A list which corresponds to the TSP route.

1. Pick the smallest edge available, contract it and update the weights in D accordingly.

2. Repeat (1) until there are only two edges available $\overrightarrow{ab}$, and $\overrightarrow{cd}$.

3. The contracted paths of the cycle $\overline{abcda}$ form the sought solution.
```

Obviously, the contraction operation should be defined, and indeed it is the hard part:

```
Algorithm 3. Contraction($D, V, v_i, v_j$):

Input: A set of vertices $V = \{v_1, ..., v_n\}$, the corresponding distance matrix $D$ (which defines a distance $d$), and an edge $e = \overrightarrow{v_i v_j}$ with $v_i, v_j \in V, i \neq j$ to contract.

Output: An updated vertex set $V$ of $n-1$ vertices, with $w$ added, $v_i$, $v_j$ removed, and an updated weight matrix $D$.

1. Let $w$ be a new vertex and update $V = V \cup \{w\}$.
2. For $a, b \in V$, reassign $d(a,b) = \begin{cases} \infty \text{ if } a = b. \\ d(v_j, b) \text{ if } a = w. \\ d(a, v_i) \text{ if } b = w. \\ d(a,b), \text{ otherwise.} \end{cases}$

3. Update $V = V \cup \{w\} - \{v_i, v_j\}$ and recalculate the weight matrix $D$ of $V$.
```

Notice that this implementation of the contraction only removes unimportant information from the matrix, that is, given the edge $\overrightarrow{v_i v_j}$ to be contracted, the algorithm will only remove information about edges of the form $\overrightarrow{u v_j}$ for $u \neq v_i$ and of the form $\overrightarrow{v_i u}$ for $u \neq v_j$ which is useless information because we are trying to form a cycle, and we already selected the edge $\overrightarrow{v_i v_j}$. Although simple to state, the implementation should run in $O(n^2)$ which is not trivial. For the present work an implementation was programmed and it is shown to run in $O(n^2)$ in the Appendix.

The following theorem was first presented by Ong and Moore [22].

Theorem 1.4. The *Greedy algorithm is a* $\frac{1}{2}+\frac{\lceil \log_2(n) \rceil}{2}$ *approximation algorithm for the TSP.*

Proof. Identical to the proof of Theorem 1.1.

□

As we can see, both proofs depend on the triangle inequality, so the results are not true for the TSP without triangle inequality. However, we can give a classical result for the TSP without triangle inequality from [10]:

Theorem 1.5. *If P* $\neq$ *NP, then for any constant* $\rho \geq 1$ *there is no* ρ*-approximation algorithm for the TSP without triangle inequality.*

Proof. Let's suppose $\mathcal{A}$ is an ρ-approximation algorithm for the TSP with $\rho \geq 1$.

Let's consider a graph $G=(V,E)$, let $K=(V,E')$ be the complete graph of the vertex set V and consider a cost function $c:V \to \mathbb{R}$ defined as:

$$c(v)=\begin{cases} 1 \text{ if } v \in E. \\ \rho|V|+1 \text{ if } v \notin E. \end{cases}$$

If G has a tour, such a tour has cost $|V|$, on the other hand, if we construct a tour that has an edge not in E, such a tour has a cost of at least $|V|-1+\rho|V|+1=|V|+\rho|V|$.

Now let's consider the TSP in K. We note that the minimum cost possible for any tour in K is $|V|$, by hypothesis $\mathcal{A}$ can find a tour with approximation ratio ρ. If $\mathcal{A}$ finds a tour with, at least, an edge not in E the ratio would be at least:

$$\frac{|V|+\rho|V|}{|V|}=\rho+1>\rho$$

Which contradicts the hypothesis that $\mathcal{A}$ is a ρ-approximation algorithm. Hence, the tour found should only have edges contained in E, that is, $\mathcal{A}$ found a tour in G. Therefore, $\mathcal{A}$ solved the Hamiltonian-tour problem for G which is well-known to be NP-Complete. And to transform the problem we only had to construct the complete graph K and the cost function c which we did in linear time (i.e., polynomial time). Then P = NP. The theorem follows by contrapositive.

□

A complete proof that the TSP is NP-complete can be found in [7], in which the author proves all the way SAT $\preceq$ CLIQUE $\preceq$ VC $\preceq$ HC $\preceq$ TSP.

2. Clustering

Clustering (also called data segmentation) refers to the methods used in data analysis to partition some given objects (which we will call points) into sets called *clusters*, the objective is to partition them in a way such that objects in the same cluster are similar, whereas objects in different clusters are different. We will also call *clustering* the partition of the given space into the clusters. Obviously, we need to define what we mean by saying similar or different, as we will see, that is not an easy task and in most cases, it depends on the application. In this work we will assume that two clusters never intersect, but there might happen that some objects are not assigned to any cluster.

cluster

clustering

We will use the notation $C = \{c_1, ..., c_k\}$ to denote a clustering C of k clusters $C_1, \ldots, C_k$ whose centroids are $c_1, \ldots, c_k$. The *centroids* are points in the space that act as representatives of the clusters, later we will ask them to fulfill more conditions. In the next page we will introduce the concept of proximity, but for the moment suppose we have defined some distance function d, we will say that a point x belongs to a cluster C_i, $x \in C_i$, if $i = \underset{i \in \{1,...,K\}}{\operatorname{argmin}} d(x, c_i)$. Finally, we will say that a clustering C is a *k-clustering* if C has k clusters.

C

centroid

k-clustering

The most famous clustering method is known as the k-means method (or Lloyd's algorithm) [1], it is easy to implement and can provide local optimums as we will see next. The following section is based on [32].

Clustering validation

We would like to have methods to compare, or rank, different clusterings either generated by different executions of the same algorithm or generated by different algorithms. Ideally, we would like to have an optimization function associated to every clustering. In general, there are clustering algorithms that do not have any associated objective function. Optimization problems do not always have objective functions, hence, not even the task of evaluating different clusterings generated by the same algorithm is easy.

No single evaluation metric is widely accepted to compare different clusterings, even more, no single evaluation metric can be applied in general for any type of data [32]. However, in most cases we can define a *proximity* function to compare the points in some way. As we will see in the next section, the k-means algorithm uses as a proximity function the square of the Euclidean distances, but in general the proximity can take many forms such as the cosine similarity $c(x, y) = \dfrac{x \cdot y}{|x||y|}$ which is not even a metric. For the purposes of this work, we will use the square of the Euclidean distance as the proximity function, that is, for any two points x, and y:

proximity

$$\text{proximity}(x, y) = d(x, y)^2$$

Once we have established that in our case the proximity function is the square of the distances, we can introduce some key concepts, fundamental to clustering validation study that are generally presented in terms of proximity, but given our assumptions, we will present them in terms of distances.

Definition 2.1. *Given a cluster C we define the cohesion of C as:*

cohesion

$$\text{cohesion}(C) = \sum_{x,y \in C} d(x, y)^2$$

Definition 2.2. *Given two clusters C_i, C_j we define the separation between C_i and C_j as:*

separation

$$\text{separation}(C_i, C_j) = \sum_{\substack{x \in C_i \\ y \in C_j}} d(x, y)^2$$

Note that those calculations are $\Theta(n^2)$. Many times it is preferred to calculate a prototype-based cohesion and a prototype-based separation based just on the centroids of the clusters.

Definition 2.3. *Given a cluster C with centroid c, we define the p-cohesion of C as:*

p-cohesion

$$\text{p-cohesion}(C) = \sum_{x \in C} d(x, c)^2$$

Definition 2.4. *Given two clusters C_i, C_j with c_i, c_j their respective centroids, we define the p-separation between C_i and C_j as:*

p-separation

$$\text{p-separation}(C_i, C_j) = d(c_i, c_j)^2$$

p-centroid

We will also define the p-separation of a single cluster. A *p-centroid* is constructed by taking the average coordinates of all the points in the space, i.e., it is the overall mean of all the points in the space.

Definition 2.5. *Given a cluster C_i with centroid c_i, and c the p-centroid of the space, we define the p-separation of C_i as:*

p-separation

$$\text{p-separation}(C_i) = d(c_i, c)^2$$

Definition 2.6. *Given k clusters C_i, with centroids c_i, for $i \in \{1, ..., k\}$ with c the p-centroid, we define the total separation between clusters* or the *between sum group of squares (SSB)* as:

$$\text{SSB} = \sum_{i=1}^{K} |C_i| \text{p-separation}(C_i) = \sum_{i=1}^{K} |C_i| d(c_i, c)^2$$

SSB

Definition 2.7. *Given k clusters C_i with $i \in \{1, ..., k\}$, with c the p-centroid. We define the total sum of squares (TSS) as:*

$$\text{TSS} = \sum_{i=1}^{K} \sum_{x \in C_i} d(x, c)^2$$

TSS

Some important observations are necessary: unlike the centroids of the clusters that can be updated as we iterate some clustering algorithm, the p-centroid is always fixed. In the same way, the TSS is always the same number.

As can be intuited, if the cohesion is minimized then the separation is maximized. We will prove this fact, but first we observe that:

$$\sum_{i=1}^{K} \text{p-cohesion}(C_i) = \sum_{i=1}^{K} \sum_{x \in C_i} d(x, c_i)^2$$

We will call this last sum $\sum_{i=1}^{K} \sum_{x \in C_i} d(x, c_i)^2$ the *sum of the squared error (SSE)* and it will be a fundamental concept in this work because it will be the objective function of the clustering algorithms that we will present. In the last equation, we saw that the sum of the squared error is analogous to the total cohesion between clusters.

SSE

Proposition 2.8. *The sum of the total cohesion and the total separation is a fixed constant, in fact, it is the total sum of squares:*

$$\text{SSE} + \text{SSB} = \text{TSS}$$

Before presenting the proof of the proposition we need an important identity.

Lemma 2.9. *Given k clusters C_i, with centroids c_i, for $i \in \{1, ..., k\}$, and c the p-centroid, it holds that:*

$$\sum_{i=1}^{K} \sum_{x \in C_i} \Big((x - c_i) \cdot (c_i - c) \Big) = 0$$

Proof (lemma 2.9): Let $i \in \{1, \dots k\}$,

$$\sum_{x \in C_i} \Big((x - c_i) \cdot (c_i - c) \Big) = \sum_{x \in C_i} \Big(x \cdot c_i - x \cdot c - ||c_i||^2 + c_i \cdot c \Big)$$

$$= -|C_i| ||c_i||^2 + |C_i| c_i \cdot c + \sum_{x \in C_i} x \cdot (c_i - c)$$

$$= -|C_i| ||c_i||^2 + |C_i| c_i \cdot c + (c_i - c) \cdot \sum_{x \in C_i} x$$

$$= -|C_i| ||c_i||^2 + |C_i| c_i \cdot c + (c_i - c) \cdot |C_i| c_i$$

$$= |C_i| \Big(- ||c_i||^2 + c_i \cdot c + ||c_i||^2 - c \cdot c_i \Big)$$

$$= 0$$

hence,

$$\sum_{i=1}^{K} \sum_{x \in C_i} \Big((x - c_i) \cdot (c_i - c) \Big) = 0$$

□

Proof (proposition 2.8): By definition of TSS,

$$\text{TSS} = \sum_{i=1}^{K} \sum_{x \in C_i} d(x, c)^2 = \sum_{i=1}^{K} \sum_{x \in C_i} ||x - c||^2$$

$$= \sum_{i=1}^{K} \sum_{x \in C_i} (x - c) \cdot (x - c)$$

$$= \sum_{i=1}^{K} \sum_{x \in C_i} \Big([(x - c_i) + (c_i - c)] \cdot [(x - c_i) + (c_i - c)] \Big)$$

$$= \sum_{i=1}^{K} \sum_{x \in C_i} ||x - c_i||^2 + 2 \sum_{i=1}^{K} \sum_{x \in C_i} \Big((x - c_i) \cdot (c_i - c) \Big) + \sum_{i=1}^{K} \sum_{x \in C_i} ||c_i - c||^2$$

by Lemma 2.9,

$$= \sum_{i=1}^{K} \sum_{x \in C_i} d(x, c_i)^2 + 0 + \sum_{i=1}^{K} \sum_{x \in C_i} d(c_i, c)^2$$

$$= \text{SSE} + \text{SSB}$$

□

We need to analyze the implications of the previous result. We proved that there is a duality between cohesion and separation, if we minimize cohesion we maximize separation.

That is a strong result because it means that we do not need to be concerned about minimizing the cohesion and maximizing the separation, if we can optimize any of those values then the other will be optimized as well. The last clustering measure that we will present, the silhouette coefficient, follows a similar approach of only considering one value.

Silhouette Coefficient

Given k clusters $C_1, ..., C_k$ that partition some space Ω, the *silhouette coefficient* s is a function $s : \Omega \to [-1, 1]$ that summarizes the cohesion and the separation in only one value, it is a tool to rank different clusterings with a single number. It is an effective tool to compare different clusterings obtained by similar algorithms or by different iterations of the same algorithm. *silhouette*

The objective is to get values close to 1. Then, the goodness of a clustering can be calculated by taking the average silhouette coefficient of all the points in the space.

Algorithm 4. Silhouette Coefficient (V, C)

```
Input: A set of points $V = \{v_1, ..., v_n\}$ and its clustering $C = (C_1, ..., C_k)$.
Output: A number $s$ in the range [-1, 1].

1. Let $s = 0$.
2. For $v \in V$:
3. Let $a_v = \frac{1}{|C_v|} \sum_{x \in C_v} d(x, v)^2$, (we are denoting by $C_v$ the cluster of $v$).
4. Let $b_v = \min_{C_t \neq C_v} \left\{ \frac{1}{|C_t|} \sum_{x \in C_t} d(v, x)^2 \right\}$
5. Let $s_v = \frac{b_v - a_v}{\max(a_v, b_v)}$.
6. $s = s + \frac{s_v}{|V|}$.
```

K-means

k-means

center of mass

The *k-means* algorithm works by taking a number k as a parameter which is the number of clusters to construct, then it constructs k clusters by initializing k centroids at random. Then the algorithm assigns the given points to their closest centroid. After that, the centroids are recalculated as the *center of mass* of their associated points, that is, for a cluster C the updated centroid becomes $c = \frac{1}{|C|} \sum_{x \in C} x$. The formulation of the following algorithm is based in [32].

```
Algorithm 5. K-means($V, k$)

Input: A set of vertices $V = \{v_1, ..., v_n\}$, and the required number of clusters $k$.

Output: A cluster assignment function $\alpha : V \longrightarrow \{1, 2, ..., k\}$.

1. Select $k$ initial centroids uniformly at random from the space and name them $c_1, ..., c_k$.

2. Let $\alpha(v) = \underset{i \in \{1,...,k\}}{\text{argmin}}\ d(v, c_i)^2$ for each $v \in V$.

3. Recompute each centroid, $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$ .

4. If the centroids changed in (3) repeat from (2).
```

One of the main advantages of the K-means algorithm is that it can be assigned an objective function, the sum of the squared error. As we have seen, the SSE provides a measure related to cohesion and separation of points, and the objective is to minimize it.

With the SSE as the objective function, we can prove that by taking the centers of mass in step (3) of the previous algorithm we are getting the optimum solution for each cluster.

The next result can be found on [32].

Proposition 2.10. *Given a cluster C from a clustering $C_1, \ldots, C_k$, the centroid c of C that minimizes the SSE is the center of mass of C.*

Proof. Let $C_1, \ldots, C_k$ be the clusters of a clustering C, with $c_1, \ldots, c_k$ their respective centroids. Let's pick c_t with $t \in \{1, \ldots, K\}$ and assume $c_t = (c_{t_1}, ..., c_{t_n}) \in \mathbb{R}^n$. We will calculate the optimal value for the coordinate c_{t_j} for $j \in \{1, \ldots, n\}$ and the reasoning is the same for the rest of the coordinates. Also, let's denote any point x as $x = (x_1, ..., x_n)$.

By taking the derivative of the SSE,

$$0 = \frac{\partial}{\partial c_{t_j}} \text{SSE} = \frac{\partial}{\partial c_{t_j}} \sum_{i=1}^{K} \sum_{x \in C_i} d(c_i, x)^2$$

since c_{t_j} only appears when $i = t$,

$$= \frac{\partial}{\partial c_{t_j}} \sum_{x \in C_t} d(c_t, x)^2$$

$$= \frac{\partial}{\partial c_{t_j}} \sum_{x \in C_t} \sum_{r=1}^{n} (c_{t_r} - x_r)^2$$

since c_{t_j} only appears when $r = j$,

$$= \frac{\partial}{\partial c_{t_j}} \sum_{x \in C_t} (c_{t_j} - x_j)^2$$

$$= 2 \sum_{x \in C_t} (c_{t_j} - x_j)$$

then,

$$\sum_{x \in C_t} x_j = \sum_{x \in C_t} c_{t_j} = |C_t| c_{t_j}$$

which means that,

$$c_{t_j} = \frac{1}{|C_t|} \sum_{x \in C_t} x_j$$

since this was for any $j \in \{1, ..., n\}$,

$$c_t = \frac{1}{|C_t|} \sum_{x \in C_t} x$$

$\square$

It is important to emphasize that we are only getting local optimums, that is, c_t is the optimum centroid for C_t given the fixed points that C_t has, but if we wanted to find the optimal clustering, we would need to consider all the possible points that C_t can have, and C_t can have any subset of V which makes the task of finding global optimums unfeasible.

There are many drawbacks of the K-means algorithm, most of them are related to the random nature of the initialization. There might not be any convergence at all if, for example, the generated clusters are empty. Several variants of the algorithm have been developed to fix the random initialization drawback, one popular strategy is to run the algorithm several times and choose the clustering with the minimum SSE from those results.

K-means++

As we have mentioned, the random initialization of the k-means algorithm can lead to low quality clusterings. The algorithm picks the initial centroids uniformly at random, but we do not have any information about the points to be clustered, they might all be concentrated in a small section, or some other strange behavior might occur. Another disadvantage of the k-means algorithm is that it can produce empty clusters when a centroid fails to gain any points. However, the major drawback of the algorithm is that it does not provide any approximation guarantee, that is, the approximation ratio can be arbitrarily large [20].

An algorithm that has gained popularity recently [32] is known as the k-means++ algorithm. In essence, the algorithm is almost the same as the k-means algorithm but without the uniformly random initialization of the centroids of the k-means. The content in this section is based on the work of [2] (unless otherwise stated).

Before presenting the algorithm, we need some definitions.

D

Definition 2.11. *Given a clustering $C = \{c_1, ..., c_k\}$ of a set of points V, we define the shortest distance function $D : V \to \mathbb{R}$ as $D(v) = \min\limits_{c \in C} d(v, c)$. When working with multiple clusterings we may use the notation D_C to denote the clustering that the function is considering.*

D^2-probability

Definition 2.12. *Given a clustering $C = \{c_1, ..., c_k\}$ of a set of points V, we define the D^2-probability as $P_{D^2}(v) = P(v) = \dfrac{D(v)^2}{\sum_{x \in V} D(x)^2}$.*

$\phi(A)$

Definition 2.13. *Given a clustering C of a set of points V, and a subset $A \subseteq V$, we define the potential function $\phi_C(A) = \sum\limits_{a \in A} D(a)^2$. When $A = V$, we will abbreviate the notation $\phi_C(V) = \phi_C$.*

Now, we are ready to present the algorithm,

```
Algorithm 6. K-means++($V,k$)

Input: A set of vertices $V = \{v_1, ..., v_n\}$, and the required number of clusters $k$.
Output: A cluster assignment function $\alpha : V \longrightarrow \{1, 2, ..., k\}$.
1a. Select an initial centroid $c_1$ uniformly at random from $V$.
1b. For $i = 2$ to $k$:
      1c. Choose a new centroid $c_i$, by selecting $c_i = x' \in V$ with $D^2$-probability.
2. Let $\alpha(v) = \underset{i\in\{1,...,K\}}{\text{argmin}}\ d(v, c_i)^2$ for each $v \in V$.
3. Recompute each centroid, $c_i = \frac{1}{|C_i|} \sum_{x\in C_i} x$.
4. If the centroids changed in (3) repeat from (2).
```

We will say that the three steps 1a, 1b, and 1c together are *step 1*. Then, we can see that the only difference between the k-means algorithm and the k-means++ is just step 1, all other steps are exactly the same. However, in a moment, we will see that because of that small difference we can provide good guarantees for the expected approximation ratio of the k-means++ algorithm. *step 1*

Lemma 2.14. *If $C' = C \cup \{v\}$ then $\phi_{C'}(A) \le \phi_C(A)$, for any $v \in V$ and any $A \subseteq V$.*

Proof.

$$\phi_{C'}(A) = \sum_{a\in A} \min\big(D(a), d(a,v)\big)^2 \le \sum_{a\in A} D(a)^2 = \phi_C(A)$$

□

Now a lemma by Kanungo [20],

Lemma 2.15. *Let V be a set of points which have as centroid c the center of mass of V, and let z be an arbitrary point. Then,*

$$\sum_{x\in V} d(x,z)^2 = \sum_{x\in V} d(x,c)^2 + |V| d(c,z)^2$$

Proof.

$$\sum_{x\in V} d(x,z)^2 = \sum_{x\in V} ||x - z||^2$$

$$= \sum_{x\in V} ||(x - c) + (c - z)||^2$$

$$= \sum_{x \in V} \Big([(x-c)+(c-z)] \cdot [(x-c)+(c-z)] \Big)$$

$$= \sum_{x \in V} ||x-c||^2 + 2\sum_{x \in V}(x-c)\cdot(c-z) + \sum_{x \in V} ||c-z||^2$$

$$= \sum_{x \in V} d(x,c)^2 + 2(c-z)\cdot \sum_{x \in V}(x-c) + |V|d(c,z)^2$$

$$= \sum_{x \in V} d(x,c)^2 + 2(c-z)\cdot \sum_{x \in V}(x-c) + |V|d(c,z)^2$$

and because c is the center of mass,

$$= \sum_{x \in V} d(x,c)^2 + |V|d(c,z)^2$$

□

Lemma 2.16. *Let A be a cluster of an optimal clustering O^* with centroid c_A. Let C be a clustering generated with just one centroid chosen uniformly at random from the points of A. Then $E[\phi_C(A)] = 2\phi_{O^*}(A)$.*

Proof. Let's calculate $\phi_{O^*}(A)$,

$$\phi_{O^*}(A) = \sum_{a \in A} d(c_A, a)^2$$

Now, let's calculate the expected value of $\phi_C(A)$, which by construction has only a as centroid,

$$E[\phi_C(A)] = \sum_{a \in A} \frac{1}{|A|} \sum_{x \in A} d(x,a)^2$$

Since O^* is an optimal clustering, and as a consequence of Proposition 2.10 c_A is the center of mass of A, then by Lemma 2.15,

$$= \sum_{a \in A} \frac{1}{|A|} \Big(\sum_{x \in A} \big(d(x,c_A)^2\big) + |A|d(c_A,a)^2 \Big)$$

$$= \sum_{a \in A} \frac{1}{|A|} \sum_{x \in A} \big(d(x,c_A)^2\big) + \sum_{a \in A} \frac{1}{|A|}|A|d(c_A,a)^2$$

$$= |A|\frac{1}{|A|}\sum_{x\in A}\left(d(x,c_A)^2\right) + \sum_{a\in A}\frac{1}{|A|}|A|d(c_A,a)^2$$

$$= 2\sum_{a\in A} d(c_A,a)^2$$

$$= 2\phi_{O^*}(A)$$

□

We will state a result known as the power-mean inequality.

Lemma 2.17. *Given $a_1,\dots,a_n \in \mathbb{R}$, it holds that $\sum_{i=1}^{n} a_i^2 \geq \frac{1}{n}\Big(\sum_{i=1}^{n} a_i\Big)^2$*

Proof. Let's consider the vectors $v = (1,...,1), w = (a_1,...,a_n)$ by the Cauchy-Schwarz inequality:

$$n\sum_{i=1}^{n} a_i^2 = ||v||^2||w||^2 \geq |v\cdot w|^2 = \Big(\sum_{i=1}^{n} a_i\Big)^2$$

hence,

$$\sum_{i=1}^{n} a_i^2 \geq \frac{1}{n}\Big(\sum_{i=1}^{n} a_i\Big)^2$$

□

Lemma 2.18. *Let A a cluster from an optimal clustering O^*, and let C' be an arbitrary clustering. Let's consider the clustering C which has the same centroids as C' plus an additional centroid a_0 chosen at random from A with the D^2-probability, then $E[\phi_C(A)] \leq 8\phi_{O^*}(A)$.*

Proof. Let's calculate $E[\phi_C(A)]$, note that the probability to choose an a_0 from A with the D^2 probability is $\frac{D_{C'}(a_0)^2}{\sum_{a\in A} D_{C'}(a)^2}$, furthermore, the contribution of a point $a \in A$ to the sum $\phi_C(A)$ is either the same contribution it has to $\phi_{C'}(A)$, which is $D_{C'}(a)$, or it is $d(a,a_0)^2$ in case a_0 is closer to a than its centroid in C'. Then, given the new centroid a_0, the potential $\phi_C(A) = \sum_{a\in A} \min\Big(D_{C'}(a), d(a,a_0)\Big)^2$, hence,

$$E[\phi_C(A)] = \sum_{a_0\in A}\frac{D_{C'}(a_0)^2}{\sum_{a\in A} D_{C'}(a)^2}\sum_{a\in A}\min\Big(D_{C'}(a), d(a,a_0)\Big)^2 \tag{1}$$

Given $a \in A$ let's denote its centroid in C' as c_a. Then, for any $a \in A$ it holds

$$D_{C'}(a_0) = d(a_0, c_{a_0}) \leq d(a_0, c_a)$$

by the triangle inequality,

$$\leq d(a_0, a) + d(a, c_a)$$

$$= d(a_0, a) + D_{C'}(a)$$

reordering,

$$= D_{C'}(a) + d(a, a_0)$$

in conclusion,

$$D_{C'}(a_0) \leq D_{C'}(a) + d(a, a_0)$$

by the Lemma 2.17,

$$D_{C'}(a_0)^2 \leq \Big(D_{C'}(a) + d(a, a_0)\Big)^2 \leq 2D_{C'}(a)^2 + 2d(a, a_0)^2$$

by taking the sum of all $a \in A$,

$$\sum_{a \in A} D_{C'}(a_0)^2 = |A| D_{C'}(a_0)^2 \leq 2 \sum_{a \in A} D_{C'}(a)^2 + 2 \sum_{a \in A} d(a, a_0)^2$$

which implies that,

$$D_{C'}(a_0)^2 \leq \frac{2}{|A|} \sum_{a \in A} D_{C'}(a)^2 + \frac{2}{|A|} \sum_{a \in A} d(a, a_0)^2$$

substituting in (1),

$$\begin{aligned} E[\phi_C(A)] &\leq \sum_{a_0 \in A} \frac{\frac{2}{|A|} \sum_{a \in A} D_{C'}(a)^2 + \frac{2}{|A|} \sum_{a \in A} d(a, a_0)^2}{\sum_{a \in A} D_{C'}(a)^2} \sum_{a \in A} \min\Big(D_{C'}(a), d(a, a_0)\Big)^2 \\ &= \frac{2}{|A|} \sum_{a_0 \in A} \frac{\sum_{a \in A} D_{C'}(a)^2}{\sum_{a \in A} D_{C'}(a)^2} \sum_{a \in A} \min\Big(D_{C'}(a), d(a, a_0)\Big)^2 \\ &+ \frac{2}{|A|} \sum_{a_0 \in A} \frac{\sum_{a \in A} d(a, a_0)^2}{\sum_{a \in A} D_{C'}(a)^2} \sum_{a \in A} \min\Big(D_{C'}(a), d(a, a_0)\Big)^2 \end{aligned} \tag{2}$$

now, we note that,

$$\min\Big(D_{C'}(a), d(a,a_0)\Big)^2 \le d(a,a_0)^2$$

and also,

$$\min\Big(D_{C'}(a), d(a,a_0)\Big)^2 \le D_{C'}(a)^2$$

substituting in (2) we get,

$$\le \frac{2}{|A|}\sum_{a_0\in A}\frac{\sum_{a\in A}\ D_{C'}(a)^2}{\sum_{a\in A} D_{C'}(a)^2}\sum_{a\in A} d(a,a_0)^2 + \frac{2}{|A|}\sum_{a_0\in A}\frac{\sum_{a\in A} d(a,a_0)^2}{\sum_{a\in A} D_{C'}(a)^2}\sum_{a\in A} D_{C'}(a)^2$$

$$= \frac{2}{|A|}\sum_{a_0\in A}\sum_{a\in A} d(a,a_0)^2 + \frac{2}{|A|}\sum_{a_0\in A}\sum_{a\in A} d(a,a_0)^2$$

$$= 4\sum_{a_0\in A}\frac{1}{|A|}\sum_{a\in A} d(a,a_0)^2$$

by Lemma 2.16,

$$= 8\phi_{O^*}(A)$$

□

Let's stop for a moment to analyze the consequences of the lemma. If we could guarantee that the cluster C generated by the k-means++ algorithm picked a centroid from each cluster of the optimal clustering O^* (optimal with k centroids) then we would have the bound $E[\phi_C] \le 8\phi_{O^*}$. Unfortunately, the algorithm does not provide such guarantees. It can happen that for some clusters of O^* the algorithm did not pick a centroid from them; in such a case we will say that those clusters are *uncovered* and also say that the points in those clusters are *uncovered*. On the other hand, we will say that a cluster A from O^* is *covered* if C has as a centroid a point from A and we will also say that its points are *covered*.

uncovered

covered

In summary, if the clustering generated by the algorithm covers all the clusters of the optimal clustering, then we have the previous bound and we are done. Now, we need to analyze the case in which the generated clustering left some uncovered clusters and provide a general bound.

Before continuing, let's introduce some notation.

Notation. *For $t \in \mathbb{N}$ we denote by H_t the harmonic sum of the first n numbers:*

$$H_t = \sum_{i=1}^{t}\frac{1}{i}$$

H_t

Lemma 2.19. *Let χ be a set of points, O^* its optimal clustering with k centroids, let C be an arbitrary clustering of χ. Let's suppose there are $u > 0$ uncovered clusters from O^*, let χ_u denote the points in those clusters. Let χ_c denote the points in $\chi - \chi_u$. Suppose we add $t \leq u$ centroids to C, chosen with the D^2-probability. Let C' denote the resulting clustering, then,*

$$E[\phi_{C'}] \leq \Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big)(1 + H_t) + \frac{u-t}{u}\phi_C(\chi_u)$$

Proof. By strong induction,

<u>Basis of induction:</u> Let's prove two cases. First, the case $t = 0$, $u > 0$.

As H_0 is the empty sum,

$$\Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big)(1 + H_t) + \frac{u-t}{u}\phi_C(\chi_u) = \Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big)(1 + 0) + 1 \cdot \phi_C(\chi_u)$$

remembering that $\chi = \chi_c + \chi_u$,

$$= \phi_C + 8\phi_{O^*}(\chi_u)$$

$$\geq \phi_C$$

Since $t = 0$, that means we did nothing to C' so $C = C'$. Hence

$$\phi_C = E[\phi_C] = E[\phi_{C'}].$$

So, the result holds. Now, let's prove another case with $t > 0$, so $C' \neq C$. Let's consider $t = 1 = u$.

We do not have a general bound for $\phi_{C'}(\chi_c)$ besides the broad bound provided by Lemma 2.14 $\phi_{C'}(\chi_c) \leq \phi_C(\chi_c)$, so the strategy will be to divide the problem. For some cases we will use smaller bounds, and for the rest of the cases just the bound of Lemma 2.14 and hope that when we join all the cases the result holds.

Let's consider two events (cases): let e_1 be the event that the chosen centroid was selected from an uncovered cluster and let e_2 be the event that the chosen centroid was selected from a covered cluster. Then, by the total expectation law

$$E[\phi_{C'}] = P(e_1) \cdot E[\phi_{C'} \mid e_1] + P(e_2) \cdot E[\phi_{C'} \mid e_2] \qquad (1)$$

<u>*Case 1*:</u> The centroid was chosen from an uncovered cluster.

Since $u = 1$, χ_u is actually one cluster from O^*, and we can apply Lemma 2.18 to get,

$$E[\phi_{C'}(\chi_u)|e_1] \leq 8\phi_{O^*}(\chi_u) \tag{2}$$

on the other hand,

$$E[\phi_{C'} \mid e_1] = E[\phi_{C'}(\chi_u) + \phi_{C'}(\chi_c) \mid e_1]$$

$$= E[\phi_{C'}(\chi_u) \mid e_1] + E[\phi_{C'}(\chi_c) \mid e_1]$$

for the first term we use (2), and for the second just term notice that by definition of expectation and by Lemma 2.14,

$$E[\phi_{C'}(\chi_c) \mid e_1] = \sum_{a\in\chi} P(a|e_1)\phi_{C'}(\chi_c) \leq \sum_{a\in\chi} P(a|e_1)\phi_C(\chi_c) = \phi_C(\chi_c) \tag{3}$$

then by (2), and (3),

$$E[\phi_{C'} \mid e_1] \leq \phi_C(\chi_c) + 8\phi_{O^*}(\chi_u) \tag{4}$$

That will be our bound for case 1, now let's see what happens with case 2.

<u>*Case 2:*</u> The centroid was chosen from a covered cluster.

In this case, we just use the broad inequality of Lemma 2.14,

$$E[\phi_{C'} \mid e_2] \leq \phi_C \tag{5}$$

Now we just need to calculate the probabilities $P(e_1)$, and $P(e_2)$, that is, the D^2-probability of taking one point from χ_u, and the D^2-probability of taking one point from χ_c.

By definition of D^2-probability, $P(e_1) = \sum_{z\in\chi_u} \frac{D(z)^2}{\sum_{x\in\chi} D(x)^2}$

$$= \frac{\sum_{z\in\chi_u} D(z)^2}{\sum_{x\in\chi} D(x)^2}$$

$$= \frac{\phi_C(\chi_u)}{\phi_C} \tag{6}$$

analogously,

$$P(e_2) = \frac{\phi_C(\chi_c)}{\phi_C} \tag{7}$$

then, by using (1), (4), (5), (6), and (7),

$$E[\phi_{C'}] \leq \frac{\phi_C(\chi_u)}{\phi_C}\Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big) + \frac{\phi_C(\chi_c)}{\phi_C}\phi_C$$

since $\dfrac{\phi_C(\chi_u)}{\phi_C} \leq 1$,

$$\leq \phi_C(\chi_c) + 8\phi_{O^*}(\chi_u) + \phi_C(\chi_c)$$

which obviously,

$$\leq \phi_C(\chi_c) + 8\phi_{O^*}(\chi_u) + \phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)$$

$$= \Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big) \cdot 2$$

$$= \Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big)(1 + H_1) + \frac{1-1}{1}\phi_C(\chi_u)$$

hence, the result holds for both cases of the basis of induction.

Inductive hypothesis: Let's suppose the result holds for any (u', t') such that $u' \in [1, u]$, $t' \in [0, t-1]$, and with $t' \leq u'$ (required by the hypothesis of the statement of the lemma).

Inductive step: We will prove that if the result holds for $(u-1, t-1)$ and $(u, t-1)$, then the result holds for (u, t).

Just as we did on the basis of induction, we will divide into cases and find bounds for each of those cases. Let's consider $u+1$ events: let e_1 be the event that the first centroid was chosen from a covered cluster. Suppose that the u uncovered clusters of O^* are $A_1, A_2, ..., A_u$. Let's call $\overline{A_i}$ the event that the first centroid was chosen in A_i for $i \in \{1, ..., u\}$. Then, by the total expectation law

$$E[\phi_{C'}] = P(e_1)E[\phi_{C'} \mid e_1] + \sum_{i=1}^{u} P(\overline{A_i})E[\phi_{C'} \mid \overline{A_i}]. \tag{8}$$

First, let’s calculate $P(e_1)$, similarly to (7)

$$P(e_1) = \frac{\phi_C(\chi_c)}{\phi_C}, \tag{9}$$

Let’s bound $E[\phi_{C'} \mid e_1]$. Notice that for any clustering D that meets the construction hypothesis, i.e., $D = C \cup \{q_1, q_2, ..., q_t\}$ with q_1 the first point selected and $q_1 \in \chi_c$, we can consider the cluster $D' = C \cup \{q_2, ..., q_t\}$ and because of Lemma 2.14, it will always hold $\phi_D \leq \phi_{D'}$. Then, since q_1 does not cover any uncovered cluster, we can apply the induction hypothesis to D' with the same u uncovered clusters than D' but just $t-1$ new centroids, then,

$$E[\phi_{C'} \mid e_1] = E[\phi_D] \leq E[\phi_{D'}] \leq \big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1 + H_{t-1}) + \frac{u-(t-1)}{u}\phi_C(\chi_u) \tag{10}$$

Now, let's see what happens with the other cases $\overline{A_1}, \overline{A_2}, ..., \overline{A_u}$. Let’s pick $A \in \{A_1, ..., A_u\}$ and examine $P(\overline{A})E[\phi_{C'} \mid \overline{A}]$.

similarity to (6),

$$P(\overline{A}) = \frac{\phi_C(A)}{\phi_C} \tag{11}$$

Then, we just need to bound $E[\phi_{C'} \mid \overline{A}]$. Given $a \in A$, let’s call $\overline{a}$ the event that the first selected centroid was a. Then $\overline{A} = \cup\overline{a}$, which obviously are mutually exclusive events. Then by the total expectation law:

$$E[\phi_{C'} \mid \overline{A}] = \sum_{a \in A} P(a|\overline{A})|E[\phi_{C'} \mid \overline{a}] \tag{12}$$

The probability of selecting a given $\overline{A}$ is $P(a|\overline{A}) = \dfrac{D(a)^2}{\sum_{x\in A} D(x)^2}$, hence we just need to bound $E[\phi_{C'} \mid \overline{a}]$. Let’s denote by D_a the clustering $D_a = C \cup \{a\}$. Now, to construct C', and since a was already selected, we only need to pick $t-1$ centroids that we will add to D_a which is a clustering in which A is covered. That is, in D_a the covered points are $\chi_c \cup A$ and the uncovered points are $\chi_u - A$, so it has $u-1$ uncovered clusters $\{A_1, ..., A_u\} - A$. Then, we can apply the induction hypothesis and we get:

$$E[\phi_{C'} \mid \overline{a}] \leq \big(\phi_{D_a}(\chi_c \cup A) + 8\phi_{O^*}(\chi_u - A)\big)(1 + H_{t-1}) + \frac{(u-1)-(t-1)}{u-1}\big(\phi_{D_a}(\chi_u - A)\big)$$

If we denote $P(a|\overline{A})$ by p_a, and substitute in (12) the last equation,

$$\leq \sum_{a\in A} p_a\Big(\big(\phi_{D_a}(\chi_c \cup A) + 8\phi_{O^*}(\chi_u - A)\big)(1+H_{t-1}) + \frac{(u-1)-(t-1)}{u-1}\big(\phi_{D_a}(\chi_u - A)\big)\Big)$$

multiplying for $P(\overline{A})$ to obtain the second term of (8) and by (11) we get that $P(\overline{A})E[\phi_{C'} \mid \overline{A}]$

$$\leq \frac{\phi_C(A)}{\phi_C}\sum_{a\in A} p_a\Big(\big(\phi_{D_a}(\chi_c \cup A) + 8\phi_{O^*}(\chi_u - A)\big)(1+H_{t-1}) + \frac{(u-1)-(t-1)}{u-1}\big(\phi_{D_a}(\chi_u - A)\big)\Big)$$

$$= \frac{\phi_C(A)}{\phi_C}\sum_{a\in A} p_a\Big(\big(\phi_D(\chi_c) + \phi_D(A) + 8\phi_{O^*}(\chi_u) - 8\phi_{O^*}(A)\big)(1+H_{t-1}) + \frac{u-t}{u-1}\big(\phi_D(\chi_u - A)\big)\Big)$$

by Lemma 2.14 $\phi_D \leq \phi_C$, then

$$\leq \frac{\phi_C(A)}{\phi_C}\sum_{a\in A} p_a\Big(\big(\phi_C(\chi_c) + \phi_D(A) + 8\phi_{O^*}(\chi_u) - 8\phi_{O^*}(A)\big)(1+H_{t-1}) + \frac{u-t}{u-1}\big(\phi_C(\chi_u - A)\big)\Big)$$

rearranging and because the sum of all the probabilities is 1,

$$= \frac{\phi_C(A)}{\phi_C}\Big(\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1+H_{t-1}) + \frac{u-t}{u-1}\big(\phi_C(\chi_u) - \phi_C(A)\big) +$$

$$(1+H_{t-1})\Big[\sum_{a\in A}\Big(p_a\phi_D(A)\Big) - 8\phi_{O^*}(A)\Big]\Big)$$

by Lemma 2.18 $\sum_{a\in A}\Big(p_a\phi_D(A)\Big) \leq 8\phi_{O^*}(A)$, hence

$$\leq \frac{\phi_C(A)}{\phi_C}\Big(\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1+H_{t-1}) + \frac{u-t}{u-1}\big(\phi_C(\chi_u) - \phi_C(A)\big)\Big)$$

This result was for $P(\overline{A})E[\phi_{C'} \mid \overline{A}]$ for some $A \in \{A_1, ..., A_u\}$, but since we want $\sum_{i=1}^{u} P(\overline{A_i})E[\phi_{C'} \mid \overline{A_i}]$, we need to take the whole sum:

$$\sum_{i=1}^{u}\frac{\phi_C(A_i)}{\phi_C}\Big(\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1+H_{t-1}) + \frac{u-t}{u-1}\big(\phi_C(\chi_u) - \phi_C(A_i)\big)\Big)$$

$$= \big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1+H_{t-1})\cdot\frac{1}{\phi_C}\sum_{i=1}^{u}\phi_C(A_i) + \frac{u-t}{u-1}\cdot\frac{1}{\phi_C}\sum_{i=1}^{u}\phi_C(A_i)\big(\phi_C(\chi_u) - \phi_C(A_i)\big)$$

since $\phi_C(\chi_u) = \sum_{i=1}^{u} \phi_C(A_i)$,

$$= \frac{\phi_C(\chi_u)}{\phi_C}\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1 + H_{t-1}) + \frac{u-t}{u-1} \cdot \frac{1}{\phi_C}\Big[\phi_C(\chi_u)^2 - \sum_{i=1}^{u} \phi_C(A_i)^2\Big]$$

by Lemma 2.17 $\quad -\sum_{i=1}^{u} \phi_C(A_i)^2 \leq -\frac{1}{u}\phi_C(\chi_u)^2$, then

$$\leq \frac{\phi_C(\chi_u)}{\phi_C}\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1 + H_{t-1}) + \frac{u-t}{u-1} \cdot \frac{1}{\phi_C}\Big[\phi_C(\chi_u)^2 - \frac{1}{u}\phi_C(\chi_u)^2\Big]$$

$$= \frac{\phi_C(\chi_u)}{\phi_C}\Big[\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1 + H_{t-1}) + \frac{u-t}{u-1}\big(\phi_C(\chi_u) - \frac{1}{u}\phi_C(\chi_u)\big)\Big]$$

$$= \frac{\phi_C(\chi_u)}{\phi_C}\Big[\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1 + H_{t-1}) + \frac{u-t}{u-1}(1 - \frac{1}{u})\phi_C(\chi_u)\Big]$$

$$= \frac{\phi_C(\chi_u)}{\phi_C}\Big[\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1 + H_{t-1}) + \frac{u-t}{u}\phi_C(\chi_u)\Big] \tag{13}$$

then, by using (8), (9), (10), and (13) we get,

$$E[\phi_{C'}] \leq \frac{\phi_C(\chi_c)}{\phi_C}\Big[\Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big)(1 + H_{t-1}) + \frac{u-(t-1)}{u}\phi_C(\chi_u)\Big] +$$

$$\frac{\phi_C(\chi_u)}{\phi_C}\Big[\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1 + H_{t-1}) + \frac{u-t}{u}\phi_C(\chi_u)\Big]$$

$$= \Big(\frac{\phi_C(\chi_c)}{\phi_C} + \frac{\phi_C(\chi_u)}{\phi_C}\Big)\Big[\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)(1 + H_{t-1}) + \frac{u-t}{u}\phi_C(\chi_u)\Big] + \frac{\phi_C(\chi_c)}{\phi_C} \cdot \frac{1}{u}\phi_C(\chi_u)$$

since $\phi_C(\chi_c) + \phi_C(\chi_u) = \phi_C(\chi_c \cup \chi_u) = \phi_C$,

$$= \Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big)(1 + H_{t-1}) + \frac{u-t}{u}\phi_C(\chi_u) + \frac{\phi_C(\chi_c)}{\phi_C} \cdot \frac{\phi_C(\chi_u)}{u} \tag{14}$$

notice that $\phi_C(\chi_c)\phi_C(\chi_u) \leq \phi_C(\chi_c)\phi_C$, then

$$\frac{\phi_C(\chi_c) \cdot \phi_C(\chi_u)}{\phi_C} \leq \phi_C(\chi_c)$$

$$\leq \phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)$$

substituting in (14),

$$\leq \Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big)(1 + H_{t-1}) + \frac{u-t}{u}\phi_C(\chi_u) + \frac{1}{u}\big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\big)$$

$$= \Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big)\Big(1 + H_{t-1} + \frac{1}{u}\Big) + \frac{u-t}{u}\phi_C(\chi_u)$$

by hypothesis of the statement of the lemma $t \leq u$, then $\frac{1}{u} \leq \frac{1}{t}$, which means that $H_{t-1} + \frac{1}{u} \leq H_{t-1} + \frac{1}{t} = H_t$, then

$$\leq \Big(\phi_C(\chi_c) + 8\phi_{O^*}(\chi_u)\Big)\Big(1 + H_t\Big) + \frac{u-t}{u}\phi_C(\chi_u)$$

□

Theorem 2.20. *If C is a clustering of a set of points χ, constructed with the k-means++ algorithm, and O^* is the optimal clustering of k clusters. Then, $E[\phi_C] \leq 8(\ln k + 2)\phi_{O^*}$.*

Proof. First, notice that after a clustering C with k centroids has been generated by step 1, by Proposition 2.10 the next steps of the algorithm always reduce the value of $\phi_{\overline{C}}$ for any clustering $\overline{C}$ generated after C. Then, if we provide a boundary just for the clustering C generated after step 1 we have finished.

Let's suppose a clustering C is constructed with the first step of the algorithm, that is, C has the first k centroids that the algorithm generates. Let's suppose A is the cluster of O^* to which the first generated centroid x belongs, let C' be the clustering which only has x as its centroid. Then, applying the Lemma 2.19**,**

$$E[\phi_C] = E[E[\phi_C]] \leq E\Big[\Big(\phi_{C'}(A) + 8\phi_{O^*}(\chi - A)\Big)(1 + H_{k-1})\Big]$$

$$= E\Big[\phi_{C'}(A) + 8\phi_{O^*}(\chi) - 8\phi_{O^*}(A)\Big](1 + H_{k-1})$$

$$= \Big(E[\phi_{C'}(A)] + 8\phi_{O^*}(\chi) - 8\phi_{O^*}(A)\Big)(1 + H_{k-1})$$

by Lemma 2.16,

$$\leq \Big(2\phi_{O^*}(A) + 8\phi_{O^*}(\chi) - 8\phi_{O^*}(A)\Big)(1 + H_{k-1})$$

$$\leq 8\phi_{O^*}(\chi)(1 + H_{k-1}) \tag{1}$$

since $\frac{1}{x}$ is a decreasing function, we can bound the lower Riemann sum,

$$\sum_{x=2}^{k} \frac{1}{x} \leq \int_{1}^{k} \frac{1}{x} dx = \ln k$$

then,

$$H_{k-1} \leq H_k = 1 + \sum_{x=2}^{k} \frac{1}{x} \leq 1 + \ln k$$

substituting in (1) we get,

$$E[\phi_C] \leq 8\phi_{O^*}(\chi)(2 + \ln k)$$

$$= 8(\ln k + 2)\phi_{O^*}$$

□

3. Drone package delivery

The usage of Unmanned Aerial Vehicles (UAV) has been recorded since the year 400 BC when the ancient mathematician Archytas invented a steam-powered aerial vehicle known as the *flying pigeon*. It is believed that the invention had a military purpose; to generate a weapon with a longer range than an arrow. However, the earliest large-scale usage of drones did not happen until 1849 when the city of Venice was attacked with unmanned explosive hot air balloons.

UAV have been used for military purposes for centuries, but it was not until the appearance of the first commercial and widely available drone in 2013 (DJI Phantom) that scientists became interested in civil applications as can be seen by the number of published articles about UAV by year:

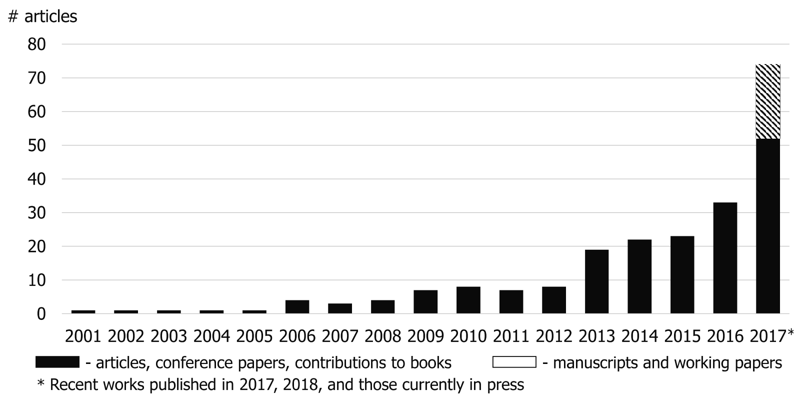

Figure 1, source: [23]

In fact, in 2021 the number of published articles about civilian applications surpassed the number of articles about military applications:

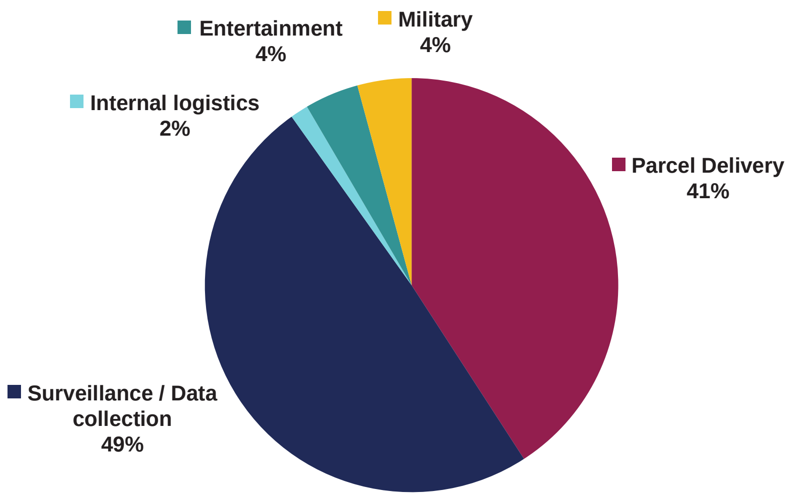

Figure 2, source: [27]

For this work, we will classify the drone applications in six categories:

1. Package delivery
2. Surveillance/Patrolling
3. Agriculture/Irrigation
4. Topography/Photogrammetry
5. Telecommunications
6. Cinematography/Entertainment

It is important to mention that those six applications are not theoretical, today drones are being used for those applications: In some rural areas of France drones are being used to make package deliveries, in the United States drones are used to patrol the frontier with Mexico and reduce illegal border crossing of migrants, in Japan drones are used to irrigate the paddy fields, in Mexico City drones are used by the Natural Resources and Rural Development Commission (CORENADR) for territorial planning, in Ukraine the company Starlink has deployed drones to be used as routers to enable (temporary) internet and mobile access in areas affected by the war, and finally the applications of drones in cinematography require no example [23], [27].

For the current study, we will address only the application of package delivery. Firstly, because it is one of the areas in which most of the research is being carried out, and secondly, because it is an application that the common man will actually notice. Seeing flying drones delivering packages will become commonplace.

We should mention that all major companies (DHL, UPS, etc.) are currently doing intense research to make use of the drone technology to deliver packages [1], currently the most ambitious project is from Amazon who pretends to put in service drones to deliver small packages to their prime clients next year.

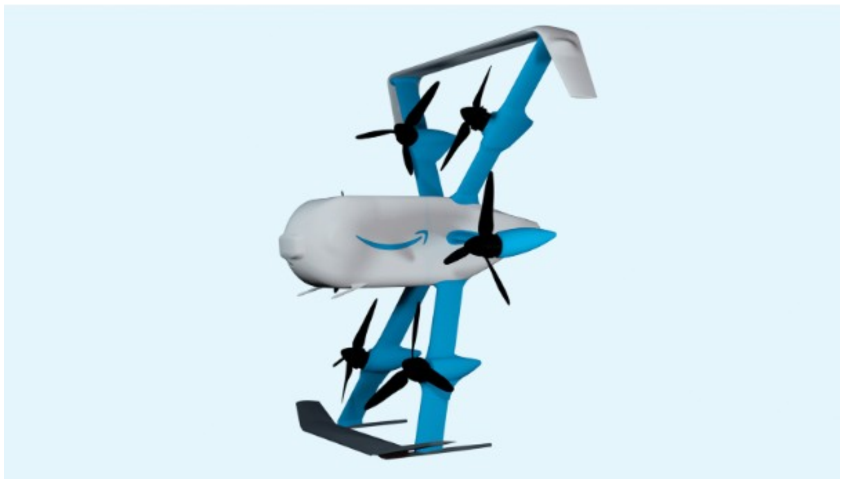

Figure 3, source: Amazon

Worst-case analysis

Although military articles about Unmanned Aerial Vehicles have been written for decades, following the previous discussion, it should not come as a surprise that the first mathematical article about drones was just published eight years ago in 2015 [21]. Ever since, hundreds of mathematical articles about drones have been published, unfortunately not so many with theoretical results. For this work we managed to identify just four articles with theoretical results published in peer-reviewed journals [6], [13], [25], [34]. The first two articles present some models and study the complexity of those models while the latter two study worst-case results. We also found a PhD thesis [11] which works primarily with synchronization problems and a conference article about approximation algorithms [19].

Following the study of [34], we will present several quite interesting worst-case results. But before we announce the results, we need to introduce some notation and describe the models with which we are going to work.

The VRP (Vehicle Routing Problem) is a well-studied problem, there are even recent books about the problem [15], [33]. Hence, besides a brief description of the problem, we will not talk much about it.

VRP

In principle, the *VRP*$_m$ can be thought as a generalization of the TSP in which instead of having just one salesman to visit a set of n clients (or cities), now there are m salesmen all of which exit from the same origin, each visits its own subset of clients and returns to the origin. That is, the set of clients is partitioned, we solve a TSP instance for each of those subsets and the objective is to minimize the time at which the last salesman returns to the origin. However, since it is a more modern version of the TSP, now we say that there are m vehicles instead of salesmen, the vehicles carry packages to be delivered, and they may have additional constraints.

For example, there may be a large truck pulling a trailer, that vehicle would have a large package capacity, but it would be a terribly slow vehicle. On the other hand, a vehicle may be a motorcycle which would be an extremely fast vehicle but with an exceptionally low package capacity. A more general definition for the VRP is provided by [33]:

> *Given a set of transportation requests and a fleet of vehicles, determine a set of vehicle routes to perform all transportation requests with the given fleet at minimum cost; in particular, decide which vehicle handles which requests in which sequence so that all vehicle routes can be executed.*

For our purposes, we will assume there are n clients to be visited (and the request is to deliver exactly one package to each client) by a fleet of m homogeneous trucks, that is, the trucks have the same package capacity and travel at the same speed (unless otherwise stated). The cost we aim to minimize is the trip duration of the last vehicle to return to the origin.

Now we will introduce a problem that is more related to our interests. As the name suggests, the $\mathit{VRPD}_{m,\alpha,k}$ (Vehicle Routing Problem with Drones) is an extension of the VRP model in *VRPD*
which we intend to visit (deliver exactly one package) to n customers with a fleet of m homogeneous trucks, each one carrying a fleet of k homogeneous drones, each one capable of carrying exactly one package and with α speed. The truck package capacity is C, and we will assume that deliveries are instantaneous, that is, there is no waiting time to deliver a package once the truck or drone arrives to the client. Formally, we make the following assumptions:

1. A drone can carry exactly one package when flying.
2. A drone has a battery which lasts U units of time per charge, for this study we will assume U is big enough to allow any drone to carry out its assigned task and travel to the next pickup location.
3. We will assume that we are working in a metric space and that the drones and trucks use the same metric to measure distances.
4. We will assume that the time required to swap the battery of a drone to make it ready for the next flight is negligible.
5. We will denote by $\alpha = \frac{\text{drone speed}}{\text{truck speed}}$. *α*
6. A drone launched from a truck must return to the same truck. That can be either at the same location where it was launched or at a different location.
7. A drone can only be launched when the truck is at the depot or in a client location, and the drone can only be picked up when the truck is at the depot or in a client location.
8. The vehicle (drone or truck) that arrives first at the pick-up location must wait for the other vehicle to arrive.

The objective will be to minimize the time of return of all the trucks and drones, or as we mentioned earlier, minimize the arrival time of the last vehicle (truck or drone) to the depot.

Given a solution f to a problem P (TSP, VRP, VRPD, etc.) we will denote by $Z^f(P)$ the cost *Z(P)*
of the solution f (time in our case), and we will denote by $Z(P)$ the cost of the optimal solution to a problem P, which in our case will always be a minimum, $Z(P) = \min_f Z^f(P)$.

We will say that R is a *route* if it is a path contained on some tour, and we will denote the *route*
length of route R by L_R. *L*

Finally, given a vehicle *veh* that can be either a truck or a drone (which we will abbreviate as *trc* and *drn* respectively) let's denote by W_R^{veh} the waiting time of *veh* at route R, let's denote by *W*
D_R^{veh} the trip duration of *veh* at route R, and let's denote by T_R^{veh} the actual travel time of a *veh* *D*
through route R, that is $D_R^{veh} = T_R^{veh} + W_R^{veh}$. *T*

As mentioned before, we will present the results of [34]. First, we will present a result which gives a bound that relates the trip duration of a single truck with the trip duration of a truck with k drones. For this result, we will assume that the truck and the drones travel at the same speed, later we will relax this assumption, but this first result is useful in presenting the proof technique that will be used through this section.

Theorem 3.1. $\dfrac{Z(\mathrm{TSP})}{Z(\mathrm{VRPD}_{1,1,k})} \leq k+1$ *and the bound is tight.*

Proof: Let's consider an optimal $\mathrm{VRPD}_{1,1,k}$ solution, we can consider the route R_0 that was followed by the truck and the routes $R_1, \ldots, R_k$ that were followed by each of the k drones (the routes are composed by the part in which they were in the truck and the part when they were flying).

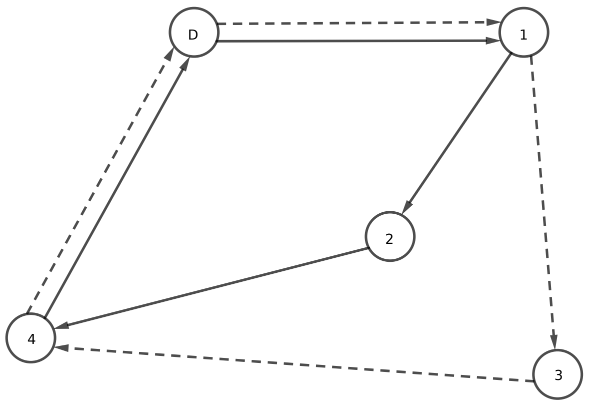

Figure 4

In figure 4 we have a solution for a $\mathrm{VRPD}_{1,1,1}$ problem with 4 customers, the truck route is represented with the continuous line:

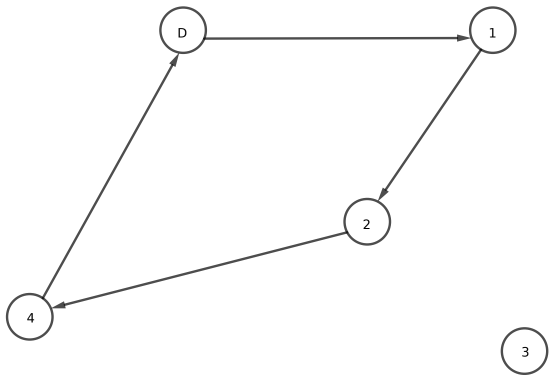

Figure 5

and the drone route is represented with the dotted lines:

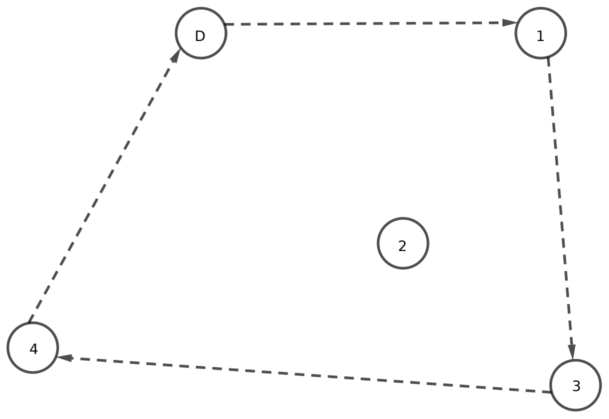

Figure 6

Notice that the drone is not flying its entire route, sometimes it is being carried by the truck, but we do not make any distinction. In the previous example, it was carried by the truck when it traveled from the depot to customer 1, and from customer 4 to the depot, but we do not care. We will denote the followed paths with arrows such as $A \to B$ to indicate that A was followed by B, or as $R_i \to R_j$ to indicate that route R_i was followed by route R_j.

Let R be the route generated by traveling with a truck (which has the same speed as a drone by hypothesis) through all the $k+1$ routes: $R_0 \to \cdots \to R_k$, then the route R visits all the customers. Because of the waiting time, all the routes R_i have the same duration for $i \in \{0, ..., k\}$ hence,

$$D_R^{trc} = \sum_{i=0}^{k} D_i^{trc} = (k+1)Z(\text{VRPD}_{1,1,k}) \tag{1}$$

and by definition of T and D,

$$T_R^{trc} \leq D_R^{trc} \tag{2}$$

We can construct a solution f to the TSP by traveling through the route R but skipping any customer already visited and continuing to the next unvisited client until all clients are visited, returning to the depot at the end. By the triangle inequality,

$$Z^f(\text{TSP}) \leq T_R^{trc} \tag{3}$$

By combining (1), (2), (3), and because the optimal value of the TSP is less than any feasible solution:

$$Z(\text{TSP}) \leq Z^f(\text{TSP}) \leq T_R^{trc} \leq (k+1)Z(\text{VRPD}_{1,1,k})$$

which means,

$$\frac{Z(\text{TSP})}{Z(\text{VRPD}_{1,1,k})} \leq k+1$$

Now, we just need to prove that the bound is tight. Let’s consider an example with $k+1$ customers $c_0, ..., c_k$, each at a distance 1 from the depot. For $i, j \in \{0, ..., k\}$ and $i \neq j$ let $d(c_i, c_j) = 2$. Also, let’s suppose the truck has capacity for the $k+1$ packages. Finally, let’s suppose the drones and the truck have speed 1, that is, 1 distance unit per unit of time.

Then, a feasible solution f for the $\text{VRPD}_{1,1,k}$ is to travel with the truck from the depot to the customer c_0 while at the same time the k drones are launched from the depot, and each one visits a distinct client from the set of clients $\{c_1, ..., c_k\}$. Notice the truck and the drones return to the depot at the same time after 2 units of time. Hence,

$$2 = Z^f(\text{VRPD}_{1,1,k}) \geq Z(\text{VRPD}_{1,1,k}). \tag{4}$$

Finally, let’s note that any solution to the TSP has a duration of $2(k+1)$. Then, in particular,

$$Z(\text{TSP}) = 2(k+1)$$

By (4)

$$\frac{Z(\text{TSP})}{Z(\text{VRPD}_{1,1,k})} \geq \frac{Z(\text{TSP})}{Z^f(\text{VRPD}_{1,1,k})} = k+1$$

□

In general, it is assumed that drones travel faster than trucks because they are not affected by car traffic, traffic lights, car accidents, etc. Hence, we will generalize the result from theorem 1.

Theorem 3.2. *If* $\alpha \geq 1$, $\dfrac{Z(\text{TSP})}{Z(\text{VRPD}_{1,\alpha,k})} \leq \alpha k + 1$ *and the bound is tight.*

Proof. As we did in the previous proof, consider a solution f of the $\text{VRPD}_{1,1,k}$. It defines $k+1$ routes $R_0, ..., R_k$ with R_0 the truck route and the others, the drones' routes. Let R be the route generated by traveling $R_0 \to \cdots \to R_k$. Since α is just a ratio, we can assume that the truck speed is 1 distance unit per unit of time, which forces the drones to travel at α distance units per unit of time.

Then, for $i \in \{1, ..., k\}$ the time required for the truck to travel R_i is the length of the route divided by the speed of the truck:

$$T^{trc}_{R_i} = \frac{L_{R_i}}{1} = L_{R_i} \tag{5}$$

We are taking $i > 0$ to only considered the drones' routes, in any such route a drone may have traveled some part carried by the truck, which travels at a speed less or equal to the speed of the drone, hence we have the inequality

$$\frac{L_{R_i}}{T_{R_i}^{drn}} = \frac{T_{R_i}^{trc}}{T_{R_i}^{drn}} \leq \alpha$$

The equality holds when $\alpha = 1$ or when the drone did not travel in the truck, then

$$L_{R_i} \leq \alpha T_{R_i}^{drn} \tag{6}$$

then by using (5) and (6),

$$T_R^{trc} = \sum_{i=0}^{k} T_{R_i}^{trc} = T_{R_0}^{trc} + \sum_{i=1}^{k} T_{R_i}^{trc} = T_{R_0}^{trc} + \sum_{i=1}^{k} L_{R_i} \leq T_{R_0}^{trc} + \sum_{i=1}^{k} \alpha T_{R_i}^{drn}$$

Just as we did in the proof of Theorem 3.1, we can construct a solution f to the TSP by following the route R and skipping clients already visited. By the triangle inequality,

$$Z(\mathrm{TSP}) \leq Z^f(\mathrm{TSP}) \leq T_R^{trc} \leq T_{R_0}^{trc} + \sum_{i=1}^{k} \alpha T_{R_i}^{drn} \tag{7}$$

Notice that,

$$Z(\mathrm{VRPD}_{1,1,k}) = \max\{D_{R_0}^{trc}, D_{R_1}^{drn}, ..., D_{R_k}^{drn}\}$$

$$\geq \max\{T_{R_0}^{trc}, T_{R_1}^{drn}, ..., T_{R_k}^{drn}\}$$

Let $T^* = \max\{T_{R_0}^{trc}, T_{R_1}^{drn}, ..., T_{R_k}^{drn}\}$, then

$$T^* = \frac{1}{\alpha k + 1}(\alpha k + 1)T^*$$

$$= \frac{1}{\alpha k + 1}\left(T^* + \sum_{i=1}^{k} \alpha T^*\right)$$

since T^* is the maximum,

$$\geq \frac{1}{\alpha k+1}\Big(T^{trc}_{R_0} + \sum_{i=1}^{k} \alpha T^{drn}_{R_i}\Big)$$

by (7)

$$\geq \frac{Z(\text{TSP})}{\alpha k+1}$$

then,

$$\frac{Z(\text{TSP})}{Z(\text{VRPD}_{1,\alpha,k})} \leq \alpha k+1$$

Finally, we just need to show that the inequality is tight. Let's consider an example with $k+1$ customers $c_0, ..., c_k$, a truck with capacity for at least $k+1$ packages, k drones, and a depot $\mathcal{D}$. Consider the distances:

- $d(c_0, \mathcal{D}) = 1$

for $i, j \in \{1, ..., k\}, i \neq j$:

- $d(c_i, \mathcal{D}) = \alpha$
- $d(c_0, c_i) = 1 + \alpha$
- $d(c_i, c_j) = 2\alpha$

A solution f for the $\text{VRPD}_{1,\alpha,k}$ is to send the truck and the k drones at the same time from the depot, the truck to the customer c_0 while each of the drones to a different customer from $c_1, ..., c_k$. All the drones and the truck return at the same time after 2 units of time. Then, $2 = Z^f(\text{VRPD}_{1,\alpha,k}) \geq Z(\text{VRPD}_{1,\alpha,k})$.

To calculate $Z(\text{TSP})$ we can calculate the cost that every customer contributes to the total cost of any route. Instead of calculating the distances from client to client we can consider the following equivalent costs:

1. Arriving to c_0 has a cost of 1.
2. Leaving c_0 has a cost of 1.
3. Arriving to c_i has a cost of α for any $i \in \{1, ..., k\}$.
4. Leaving c_i has a cost of α for any $i \in \{1, ..., k\}$.
5. Arriving or leaving the depot has zero cost.

Now we can see that since in any TSP solution every customer should be visited once and only once, any such solution has a cost of $2(\alpha k+1)$. Hence, $Z(\text{TSP}) = 2(\alpha k+1)$. Then,

$$\frac{Z(\mathrm{TSP})}{Z(\mathrm{VRPD}_{1,\alpha,k})} \geq \frac{Z(\mathrm{TSP})}{Z^f(\mathrm{VRPD}_{1,\alpha,k})} = \frac{2(\alpha k+1)}{2} = \alpha k + 1$$

□

In general, delivering to a set of clients C with m trucks should be faster than delivering with just one truck, that is, $Z(\mathrm{VRP}_m) \leq Z(\mathrm{TSP})$. The next result gives the reverse boundary, i.e., delivering with m trucks should be faster, but not faster than a given boundary. Let's denote by TSP$_v$ the TSP with a vehicle of speed *v*.

Theorem 3.3. *Let n customers be served by a fleet of m trucks of different speeds $v_1, ..., v_m$, and call $V = \sum_{i=1}^{m} v_i$ the combined speed. On the other hand, if the n customers are served by one truck of speed v with sufficient package capacity, we have:*

$$\frac{Z(\mathrm{TSP}_v)}{Z(\mathrm{VRP}_m)} \leq \frac{V}{v}$$

and the bound is tight.

Proof. Let's consider a VRP_m optimal solution, we can consider the m routes $R_1, ..., R_m$ generated by the m trucks. Let R be the route generated after traveling through the m routes $R_1 \to \cdots \to R_m$ where the R_i route corresponds to the route traveled by the truck with v_i speed for $i \in \{1, ..., m\}$. Since in this case we are not using drones but only trucks, there is no waiting time, so, for any $i \in \{1, ..., m\}$ we have $D_{R_i} = T_{R_i}$, hence,

$$Z(\mathrm{VRP}_m) = \max\{D_{R_1}, ..., D_{R_m}\} = \max\{T_{R_1}, ..., T_{R_m}\}$$

Let's call $T^* = \max\{T_{R_1}, ..., T_{R_m}\}$, note that

$$T^* = \frac{(v_1 + \cdots + v_m)T^*}{v_1 + \cdots + v_m} = \frac{v_1 T^* + \cdots + v_m T^*}{V}$$

since T^* is the maximum,

$$\geq \frac{v_1 T_{R_1} + \cdots + v_m T_{R_m}}{V} = \frac{L_{R_1} + \cdots + L_{R_m}}{V}$$

$$= \frac{L_R}{V}$$

$$= \frac{L_R}{v}\frac{v}{V} \tag{8}$$

We can construct a solution f for the TSP_v by taking the route R but skipping any repeated customer and returning to the depot at the end. Since the TSP_v involves a truck with speed v, the length of such route is $vZ^f(\mathrm{TSP}_v)$, and it should hold that $vZ^f(\mathrm{TSP}_v) \geq vZ(\mathrm{TSP}_v)$. Also, by the triangle inequality,

$$L_R \geq vZ^f(\mathrm{TSP}_v) \geq vZ(\mathrm{TSP}_v)$$

by (8),

$$Z(\mathrm{VRP}_m) = T^* \geq Z(\mathrm{TSP}_v)\frac{v}{V}$$

reordering the elements,

$$\frac{Z(\mathrm{TSP}_v)}{Z(\mathrm{VRP}_m)} \leq \frac{V}{v}$$

It only remains to show that the bound is tight. Consider an example with m customers $c_1, ..., c_m$, and a depot $\mathcal{D}$. For $i, j \in \{1, ..., m\}$, $i \neq j$ let $d(c_i, \mathcal{D}) = v_i$, and $d(c_i, c_j) = v_i + v_j$.

A solution f for the VRP_m problem is to send all the trucks at the same time from the depot, the truck with speed v_i visits the client c_i and returns to the depot. In that solution all the trucks return to the depot at the same time after 2 units of time. So,

$$2 = Z^f(\mathrm{VRP}_m) \geq Z(\mathrm{VRP}_m)$$

By the same analysis we followed in the previous proof, any solution to the TSP_v problem takes $\dfrac{2(v_1 + \cdots v_m)}{v} = \dfrac{2V}{v}$. Then,

$$Z(\mathrm{TSP}_v) = \frac{2V}{v}$$

which means that,

$$\frac{Z(\mathrm{TSP}_v)}{Z(\mathrm{VRP}_m)} \geq \frac{Z(\mathrm{TSP}_v)}{Z^f(\mathrm{VRP}_m)} = \frac{V}{v}$$

□

We will generalize Theorem 3.2. As we mentioned earlier, it is assumed that drones travel faster than trucks, at least during the working hours that deliveries take place, but the motivation to generalize this result and allow drones to travel slower than trucks is to have general results which can apply in case that occurs for some application.

Theorem 3.4. $\dfrac{Z(\mathrm{TSP})}{Z(\mathrm{VRPD}_{1,\alpha,k})} \leq \alpha k + 1$ *and the bound is tight.*

Proof. An optimal solution to $\mathrm{VRPD}_{1,\alpha,k}$ defines $k+1$ routes $R_0, ..., R_k$. Let's suppose R_0 is the route of the truck and the others, the drones' routes. We will construct as solution for the TSP from the $R_0, ..., R_k$ routes. Let's pick the drone which traveled a route R_u, for $u \in \{1, ..., k\}$, and pick a client t that was visited in R_u but not in R_0. Suppose that in the solution of the $\mathrm{VRPD}_{1,\alpha,k}$ in order to visit client t, the drone was launched from the truck at client i and recovered at client j .

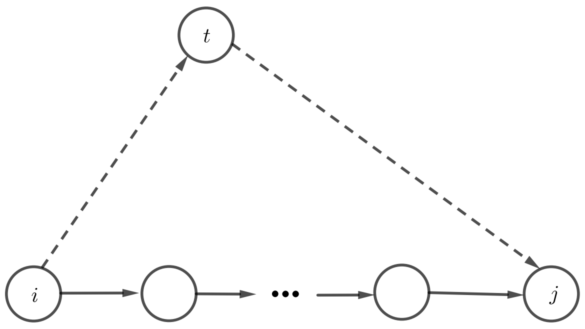

Figure 7

Let's suppose $d(i,t) \leq d(t,j)$ (the analysis is analogous in the other case). (9)

Then, if the truck wanted to visit client t while traveling in R_0, it could travel the loop $i \to t \to i$ and continue.

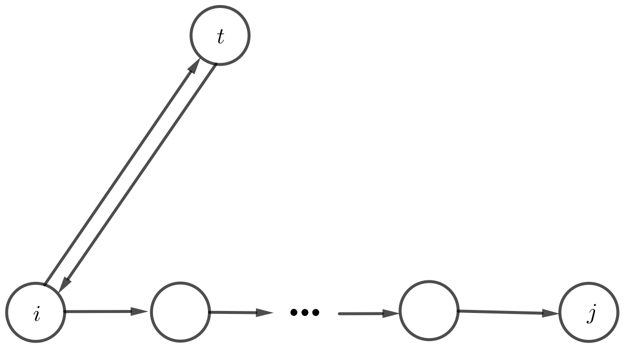

Figure 8

We will calculate the increase in time to $D^{trc}_{R_0} = Z(\mathrm{VRPD}_{1,\alpha,k})$ if the truck travels that loop to visit t. First, let's introduce some notation. In the $\mathrm{VRPD}_{1,\alpha,k}$ solution we will denote by $D^{trc}_{i,j}$ the time that passes since the departure of the truck from client *i* until it arrives to client *j*, and we denote by W^{trc}_j the waiting time of the truck at client j and by W^{drn}_j the waiting time of the drone (that visited client t) at client j. Then, by remembering that we always suppose the speed of the truck being 1 and the speed of the drones being α, and by denoting $L_{a,b} = d(a,b)$ for any a, b clients,

$$D_{i,j}^{trc} + W_j^{trc} = \frac{L_{i,t}}{\alpha} + \frac{L_{t,j}}{\alpha} + W_j^{drn}$$

which means that,

$$L_{i,t} + L_{t,j} = \alpha(D_{i,j}^{trc} + W_j^{trc} - W_j^{drn})$$

$$\leq \alpha(D_{i,j}^{trc} + W_j^{trc})$$

then, by the supposition we made at (9),

$$2L_{i,t} \leq L_{i,t} + L_{t,j} \leq \alpha(D_{i,j}^{trc} + W_j^{trc})$$

We are assuming the truck travels at 1 distance unit per unit of time, so the time it takes to travel $i \to t \to i$ is $2L_{i,t}$, and now we have a bound for $2L_{i,t}$ given by $\alpha(D_{i,j}^{trc} + W_j^{trc})$. That is, if the truck decides to visit client t from i and return to i, that takes at most $\alpha(D_{i,j}^{trc} + W_j^{trc})$. And this was for an arbitrary client t in R_u that is not visited in R_0, we can follow the same reasoning for every customer in R_u that was not visited in R_0 and calculate the time increase. To formalize this idea, let's say that a pair of customers (a, b) have the property P_u if

1. The route R_0 visits customer a and later, at some point, it visits customer b.
2. The route R_u contains the path $a \to c \to b$ for some customer c.
3. c is not visited in the route R_0.

Then, the time increase to $D_{R_0}^{trc}$ if the truck decided to visit the customers in R_u that were not visited in R_0 is at most

$$\alpha \cdot \sum_{(a,b)\in P_u} (D_{a,b}^{trc} + W_b^{trc})$$

Notice that for any $(a, b) \in P_u$, the time $D_{a,b}^{trc} + W_b^{trc}$ considered in $D_{R_0}^{trc}$, and if $(a, b), (q, r) \in P_u$ it is not possible that in R_u q is visited after a and before b because that would mean that R_u contains $a \to q \to b$, and that q is not visited by R_0 (by property 2) which contradicts the fact that $(q, r) \in P_u$. By the same reasoning, a cannot be visited after q and before r. Hence, for $(a, b) \in P_u$ the times $D_{a,b}^{trc} + W_b^{trc}$ never intersect, then

$$\alpha \cdot \sum_{(a,b)\in P_u} (D^{trc}_{a,b} + W^{trc}_{b}) \leq \alpha D^{trc}_{R_0}$$

That is a bound for the time increase if the truck visits all the customers visited in the route R_u but not visited in the route R_0, where R_u is an arbitrary route of $\{R_1, ..., R_k\}$. Therefore, if we apply the same procedure for all the R_u routes, the increase of time to visit all the customers not visited by R_0 is at most

$$\alpha k D^{trc}_{R_0}$$

Which means that we can generate a truck route that visits every customer and takes at most

$$\alpha k D^{trc}_{R_0} + D^{trc}_{R_0} = D^{trc}_{R_0}(\alpha k + 1)$$

and it may look something like

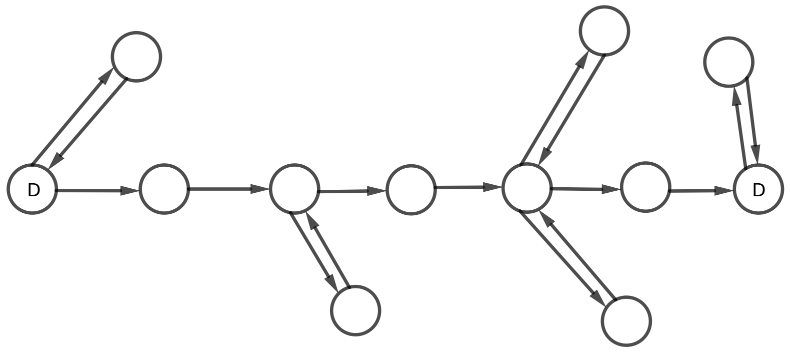

Figure 9

From there, we can construct a solution f for the TSP by removing loops, the strategy is to remove one loop at a time by returning to the next client to be visited instead of returning to the same client (or the reverse idea if the loop is on the depot at the end). In the previous example,

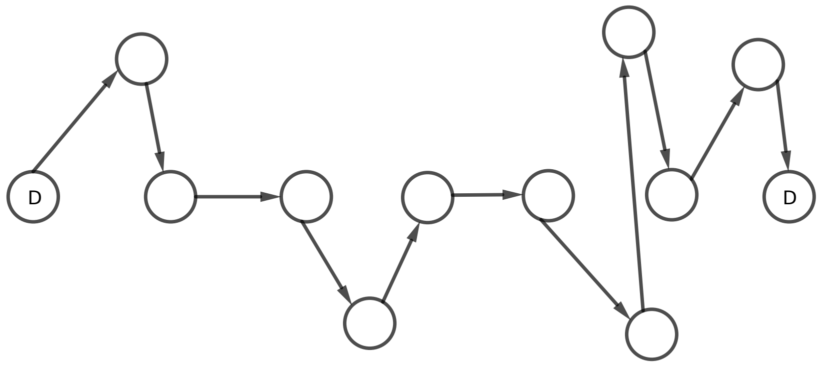

Figure 10

and by the triangle inequality,

$$Z^f(\mathrm{TSP}) \le D^{trc}_{R_0}(\alpha k + 1)$$

and also,

$$Z(\mathrm{TSP}) \le Z^f(\mathrm{TSP}) \le D^{trc}_{R_0}(\alpha k + 1)$$

by remembering that $D^{trc}_{R_0} = Z(\mathrm{VRPD}_{1,\alpha,k})$ we get,

$$\frac{Z(\mathrm{TSP})}{Z(\mathrm{VRPD}_{1,\alpha,k})} \le \alpha k + 1$$

To show that the bound is tight we consider the same example we used to prove the tightness in Theorem 3.2, and the proof is analogous.

□

The next result is a direct consequence of Theorem 3.4.

Corollary 3.5. $\dfrac{Z(\mathrm{TSP})}{Z(\mathrm{VRPD}_{m,\alpha,k})} \le m(\alpha k + 1)$ *and the bound is tight.*

Proof. Let's consider an optimal solution of the $\mathrm{VRPD}_{m,\alpha,k}$, since the solution is optimal, by the triangle inequality, two different trucks never visit the same client. Then we can partition the set of clients $C = C_1 \cup ... \cup C_m$ where C_i is the set of clients visited by the i^{th} truck, or a drone in that truck, for $i \in \{1, ..., m\}$.

Therefore, the optimal solution to the $\mathrm{VRPD}_{m,\alpha,k}$ for C generates m $\mathrm{VRPD}_{1,\alpha,k}$ solutions *VRPD*$^{(i)}$ $f_1, ..., f_m$ for each set of clients $C_1, C_2, ..., C_m$ respectively. Let's denote by $\mathrm{VRPD}^{(i)}_{1,\alpha,k}$ the *TSP*$^{(i)}$ $\mathrm{VRPD}_{1,\alpha,k}$ restricted to the set of clients C_i, and by $\mathrm{TSP}^{(i)}$ the TSP problem restricted to the set of clients C_i for $i \in \{1, ..., m\}$ then by Theorem 3.4,

$$Z(\mathrm{TSP}^{(i)}) \le (\alpha k + 1) Z(\mathrm{VRPD}^{(i)}_{1,\alpha,k})$$

and for the given solution f_i,

$$\le (\alpha k + 1) Z^{f_i}(\mathrm{VRPD}^{(i)}_{1,\alpha,k}) \tag{10}$$

note that because $Z(\mathrm{TSP})$ is the optimal value of the TSP over all the customers,

$$Z(\mathrm{TSP}) \leq \sum_{i=1}^{m} Z(\mathrm{TSP}^{(i)})$$

by (10)

$$Z(\mathrm{TSP}) \leq \sum_{i=1}^{m} Z(\mathrm{TSP}^{(i)}) \leq \sum_{i=1}^{m} (\alpha k + 1) Z^{f_i}(\mathrm{VRPD}^{(i)}_{1,\alpha,k})$$

and because $Z(\mathrm{VRPD}_{m,\alpha,k})$ is the travel time of the last truck to return to the depot, i.e., $Z(\mathrm{VRPD}_{m,\alpha,k}) = \max\limits_{i\in\{1,\ldots,m\}} Z^{f_i}(\mathrm{VRPD}^{(i)}_{1,\alpha,k})$, we have,

$$Z(\mathrm{TSP}) \leq \sum_{i=1}^{m} Z(\mathrm{TSP}^{(i)}) \leq \sum_{i=1}^{m} (\alpha k + 1) Z^{f_i}(\mathrm{VRPD}^{(i)}_{1,\alpha,k}) \leq \sum_{i=1}^{m} (\alpha k + 1) Z(\mathrm{VRPD}_{1,\alpha,k})$$

$$= m(\alpha k + 1) Z(\mathrm{VRPD}_{1,\alpha,k})$$

then,

$$\frac{Z(\mathrm{TSP})}{Z(\mathrm{VRPD}_{m,\alpha,k})} \leq m(\alpha k + 1)$$

Finally, let's show the bound is tight. Let's consider an example with $m(k+1)$ clients that we will denote by $c_j^{(i)}$ with $j \in \{0, 1, ..., k\}$, $i \in \{1, ..., m\}$ and a depot $\mathcal{D}$. We define the distances between the clients and depot as: $c_j^{(i)}$

1. $d(\mathcal{D}, c_0^{(i)}) = 1$ for $i \in \{1, ..., m\}$.
2. $d(\mathcal{D}, c_j^{(i)}) = \alpha$ for $i \in \{1, ..., m\}$ and $j \in \{1, ..., k\}$.
3. $d(c_0^{(i)}, c_0^{(j)}) = 2$ for $i, j \in \{1, ..., m\}, i \neq j$.
4. $d(c_0^{(i)}, c_v^{(j)}) = 1 + \alpha$ for $i, j \in \{1, ..., m\}, v \in \{1, ..., k\}$.
5. $d(c_u^{(i)}, c_v^{(j)}) = 2\alpha$ for $i, j \in \{1, ..., m\}$, and $u, v \in \{1, ..., k\}$ with $(u, i) \neq (v, j)$.

Then, a solution f for the $\mathrm{VRPD}_{m,\alpha,k}$ is to launch the m trucks from the depot to visit the clients $c_0^{(1)}, ..., c_0^{(m)}$ such that the i^{th} truck visits $c_0^{(i)}$, and the j^{th} drone in the i^{th} truck visits customer $c_j^{(i)}$ for $j \in \{1, ..., k\}$, $i \in \{1, ..., m\}$. All the trucks and drones return to the depot at same time after 2 units of time, hence $Z(\mathrm{VRPD}_{m,\alpha,k}) = 2$.

On the other hand, following an analysis similar to the one we followed in the proof of the tightness of Theorem 3.2, we can calculate the contribution that visiting and leaving a customer has for any TSP solution:

1. Arriving to $c_0^{(i)}$ has a cost of 1 for $i \in \{1, ..., m\}$.
2. Leaving $c_0^{(i)}$ has a cost of 1 for $i \in \{1, ..., m\}$.
3. Arriving to $c_j^{(i)}$ has a cost of α for $i \in \{1, ..., m\}$ and $j \in \{1, ..., k\}$.
4. Leaving $c_j^{(i)}$ has a cost of α for $i \in \{1, ..., m\}$ and $j \in \{1, ..., k\}$.
5. Arriving or leaving the depot has zero cost.

Given that in any TSP solution every client must be visited once and only once, any TSP solution g has $Z^g(\text{TSP}) = 2m(\alpha k + 1)$, then, $Z(\text{TSP}) = 2m(\alpha k + 1)$. (11)

Therefore,

$$\frac{Z(\text{TSP})}{Z(\text{VRPD}_{m,\alpha,k})} \geq \frac{Z(\text{TSP})}{Z^f(\text{VRPD}_{m,\alpha,k})} = \frac{2m(\alpha k + 1)}{2} = m(\alpha k + 1)$$

□

Finally, the most important result. Given a fleet of m trucks we will provide a bound on the time benefit if we give to each truck a fleet of k drones. The result is a corollary of the previous result but given its importance we will state it as a theorem.

Theorem 3.6. $\dfrac{Z(\text{VRP}_m)}{Z(\text{VRPD}_{m,\alpha,k})} \leq \alpha k + 1$ *and the bound is tight.*

Proof. Just as we did in the proof of Corollary 3.5 let's consider an optimal solution to the $\text{VRPD}_{m,\alpha,k}$, and partition the set of clients $C = \bigcup_{i=1}^{m} C_i$ such that the C_i are the clients either served by the i^{th} truck or a drone in that truck.

Also, by using the same notation, in inequality (10) of that proof we had

$$Z(\text{TSP}^{(i)}) \leq (\alpha k + 1) Z^{f_i}(\text{VRPD}^{(i)}_{1,\alpha,k}) \tag{12}$$

By the triangle inequality,

$$Z(\text{VRP}_m) \leq \max_{i \in \{1,...,m\}} \{Z(\text{TSP}^{(i)})\} \tag{13}$$

Let $t = \underset{i \in \{1,\ldots,m\}}{\operatorname{argmax}} \{Z(\text{TSP}^{(i)})\}$, then

$$\max_{i \in \{1,\ldots,m\}} \{Z(\text{TSP}^{(i)})\} \leq (\alpha k + 1) Z^{f_t}(\text{VRPD}^{(t)}_{1,\alpha,k})$$

$$\leq (\alpha k + 1) \max_{i \in \{1,\ldots,m\}} \{Z^{f_t}(\text{VRPD}^{(i)}_{1,\alpha,k})\}$$

$$= (\alpha k + 1) Z(\text{VRPD}_{m,\alpha,k})$$

by (13),

$$\frac{Z(\text{VRP}_m)}{Z(\text{VRPD}_{m,\alpha,k})} \leq \alpha k + 1$$

To show that the bound is tight, let's consider the example of $m(k+1)$ clients that we saw in the proof of Corollary 3.5, with the same distances as defined there. In that proof we constructed a solution f to the $\text{VRPD}_{m,\alpha,k}$ with $Z^f(\text{VRPD}_{m,\alpha,k}) = 2$. Now, let's construct a solution g for the VRP_m.

For $i \in 1, ..., m$ let the i^{th} truck travel the tour $\mathcal{D} \to c^{(i)}_0 \to c^{(i)}_1 \to \cdots \to c^{(i)}_k \to \mathcal{D}$. Then, every truck returns to the depot after $1 + (1+\alpha) + (k-1) \cdot 2\alpha + \alpha = 2(\alpha k + 1)$ units of time.

Then,

$$Z(\text{VRP}_m) \leq Z^g(\text{VRP}_m) = 2(\alpha k + 1) \tag{14}$$

Since we are supposing all the trucks have the same speed, by Theorem 3.3,

$$Z(\text{TSP}) \leq m Z(\text{VRP}_m) \tag{15}$$

In (11) of the proof of Corollary 3.5, we calculated

$$2m(\alpha k + 1) = Z(\text{TSP}) \tag{16}$$

Therefore, by (14), (15), and (16),

$$2m(\alpha k + 1) \leq m Z(\text{VRP}_m) \leq 2m(\alpha k + 1)$$

then,

$$Z(\text{VRP}_m) = 2(\alpha k + 1)$$

substituting,

$$\frac{Z(\text{VRP}_m)}{Z(\text{VRPD}_{m,\alpha,k})} \geq \frac{Z(\text{VRP}_m)}{Z^f(\text{VRPD}_{m,\alpha,k})} = \frac{2(\alpha k + 1)}{2} = \alpha k + 1$$

□

Constraints of drone package delivery

Until now, we have only talked about theoretical models of drones, we have not considered important constraints such as government regulations regarding drones. For example, in Mexico there is a norm which classifies drones according to their weight (with payload included):

CLASSIFICATION OF REMOTELY PILOTED AIRCRAFT SYSTEMS			
MAXIMUM PAYLOAD	**CATEGORY**	**USE**	**Compliance with the Numeral of the Official Mexican Norm**
Less or equal than 2 Kg	RPAS Micro	Private Non-Commercial or Commercial	4.10, 4.11, 5.1, 5.2 y 8*
Greater than 2 kg and up to 25 Kg	RPAS Small	Private Non-Commercial or Commercial	4.10, 4.11, 6.2 y 8*
Greater than 25 kg	RPAS Large	Private Non-Commercial or Commercial	4.10, 4.11, 7.2 y 8*

Table 1, source: DOF

From the mentioned norm, we must highlight three important restrictions:

1. Drones with a weight of more than 25 kg cannot fly near residential areas. Hence, only small drones can be used to make deliveries, which in turn means that only small packages can be delivered.
2. Drones must always be within the line of sight of the pilot (i.e., the pilot must be able to see the drone as it flies). This restriction is of great concern because it means that even if the drone can technologically travel a long distance, the effective range will be considerably limited by the line of sight of the pilot.
3. Based on the previous restriction, no programmed flights are allowed. This restriction limits the feasible number of drones that can be carried by a truck, every time a drone is flying a pilot must control it. And these last two restrictions, in principle, make infeasible drone delivery models in which the drone is recovered at a different location from where it was launched.

It is believed that as drones become more widely available and the collision avoidance systems mature, the regulations will relax [23]. However, even if the line of sight restriction was relaxed, drones certainly have a maximum flight range. The flight range depends, among other things, on the design of the drone; fixed-wing drones look like airplanes and have longer flight ranges than rotor drones (drones with propellers like helicopters) but cannot land easily, so unpopular alternatives have been proposed such as dropping the packages with a parachute. In general, rotor drones that can land at the delivery location and take off again are preferred.

Unfortunately, the flight range of rotor drones is much more limited than those of fixed-wing drones. For this work, we identified that the maximum range for commercially available rotor drones to be 15 km (DJI M300, M30, Mavic 3 enterprise) but this comes with another advantage, they are cheaper.

It is currently unknown the cost per unit of the drones being developed by delivery companies, but for now, we can compare the lowest prices that we found for commercial fixed-wing drones: the eBee X with a cost of 190,000 pesos[1], and the WingtraOne Gen II with a cost of 550,000 pesos[1] to the cost of the most expensive commercial rotor drone we found, the DJI M30, with a cost of 172,000 pesos[1]. That is, the most affordable fixed-wing drones are more expensive than the most expensive rotor drone. And although that was not a formal market research about prices, it helps to clarify ideas since it is expected that delivery companies will have much lower cost per drone unit. Finally, the cost of drone batteries should also be considered.

Other important constraints are the flight time, and payload capacity. We could not find any commercial rotor drone with longer flight time than 1 hour per battery, while the fixed-wing drones that we mentioned easily break that mark. And as for the payload, even if drones were able to carry heavy packages, current government norms classify drones with respect to the drone weight and payload, so under no circumstance can the weight of the drone plus the payload it carries surpass the 25 kg restriction to visit residential areas.

The last constraint that should be considered is the lifetime of a drone, since package delivery has not been implemented in large scale, and the drones that will carry out this task are not operational yet, this constraint is difficult to estimate.

The model that we will review next considers most of the constraints mentioned. It uses a divide a conquer approach to formulate two optimization problems: first the clustering problem and later the vehicle route problem. It uses a very modern technique to provide solutions, a Deep Reinforcement Learning heuristic which in the field of drone package delivery is the first time it is being used, and it is also presented as an alternative to the current state of the art genetic algorithms such as the one it will be compared against.

[1] prices according to their official websites eBee, Wingtra, DJI. 1 United States Dollar = 17.2185 Mexican pesos.

The Parking Location and TSP with Homogeneous Drones

As we have mentioned, we will review one of the most recent models about drone package delivery. The model and results are due [1], and it will make use of most of the theory that we have presented so far.

PLTSPHD

The Parking Location and Traveling Salesman Problem with Homogeneous Drones (PLTSPHD) is a model which assumes there is exactly one truck which must deliver packages to n clients by using a fleet of m homogeneous drones. Each launched drone will deliver exactly one package and will return to the truck. Once the truck has launched one or more drones it will not move until those drones return. Because of this last property, the model can fulfill the line of sight restriction since the operator can monitor the drones while the truck is parked.

Further, it can be seen that the first problem to solve is to determine those parking locations in which the truck will stop and launch the drones to serve the customers. That is, we need to cluster the set of clients.

One initial thought can be to just launch all the drones from the depot (and no need for a truck). That works and certainly can be a solution, if for example, the drones' range, line of sight, and battery capacity are enough to travel from the depot to any client and return to the depot. Obviously, to make the problem interesting we will assume that does not happen, that there are clients far enough such that no drone can visit them and still satisfy the constraints.

An important constraint in this model is that at each cluster the number of packages to be delivered must be less or equal to the number m of drones in the truck. Notice we are not talking about the number of clients but about the number of packages to be delivered, that is because we allow clients to receive more than one package. In principle this is not necessary because when clients buy multiple items the store packs them in one single package, but it could be the case that a client ordered from different stores which use the same delivery company. We still assume that drones can only deliver one package, so in those cases multiple drones will visit any such client. Then, the restriction is that at any parking location, which will be the centroid of the clients in that cluster, the number of packages to be delivered is no more than the number of drones in the truck.

There are ways to avoid that constraint, for instance the drones can be launched and as soon as they return, they can be launched again to serve the missing customers in that cluster. However, for this model we will not use that strategy and instead construct clusters such that the number of packages to be delivered is less or equal than the number of drones in the truck.

Another constraint for the clusters is that the distance from the clients to the parking location (i.e., the centroid of the cluster) should be close enough such that for any client in the cluster a drone can travel from the truck to the client and return given the constraints mentioned earlier.

The heuristic to solve this clustering problem is, as in almost all the literature [1], the K-means algorithm, but a constrained version to keep track of the numbers of packages that were already added to the cluster. For this constrained k-means there are no known boundaries, and it is dependent on the initialization as the regular k-means. Hence, the strategy is to run it several times and pick the best clustering according to the SSE. In principle, the reason to use the K-means is just because it is easy to implement and very popular but any other clustering algorithm could have been used, such as the k-means++ or even those available in the literature without an optimization function (like the SSE) such as DBSCAN, or agglomerative clustering given that we have already studied efficient methods to rank clusters such as the silhouette coefficient discussed in chapter 2.

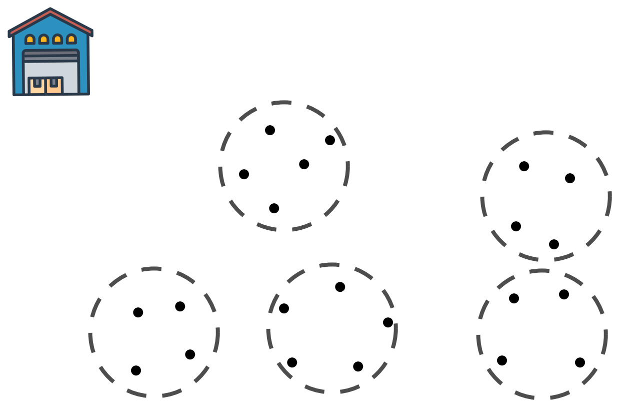

Figure 11

Once the clusters have been constructed, the dimensionality of the problem has been reduced and all is needed is to present a tour that the truck must follow to visit each centroid and return to the depot. That is, we do not care anymore about the clients or package orders and now just consider the centroids. Then, the problem becomes a VRP_1 which is essentially a TSP with added constraints. This last problem is solved with Deep Reinforcement Learning heuristic.

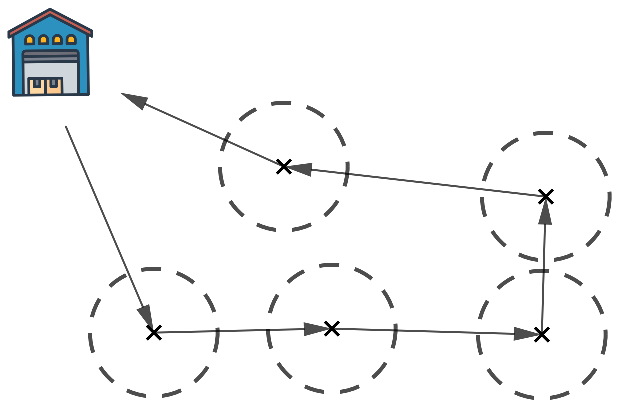

Figure 12

Before continuing, let's present the formal assumptions of the PLTSPHD.

- A1. Clients are far enough from the depot, so drones cannot be launched directly from the depot (in fact we can suppose that clients that were close enough were already served).
- A2. The truck is carrying m homogeneous drones and has unlimited space to carry the packages.
- A3. The truck can only launch the drones in the k stops (parking locations) and must wait for all the drones to return before moving to the next stop.
- A4. The weight of any package w is less than the drones' payload Q.
- A5. At any given cluster C, the packages to be delivered q_C are not more than the number of drones m.
- A6. At any given cluster a drone will make, at most, one delivery.
- A7. Any customer c with a demand of q_c packages, with $q_c > 1$, will be attended by q_c drones.
- A8. Any drone travels in a straight line from the centroid of each cluster (the parking location) to the client, delivers and returns also in a straight line.
- A9. On any trip, a drone cannot exceed the maximum flight distance R.
- A10. Every time a drone returns to the truck, a battery swap is performed to make the drone available for the next delivery (at a distinct parking location).
- A11. Once a drone has arrived at a customer location, we assume the delivery time is negligible, as we also do for the time required for battery swaps.

DRL

Deep Reinforcement Learning (DRL) is a machine learning method that was born when two of the most powerful techniques of machine learning were joined: Deep Learning and Reinforcement Learning.

On the other hand, Deep Learning was the new name given to a set of algorithms developed in the 1980s, known as Neural Networks, that resurged stronger in 2010 thanks to the large number of databases available nowadays [18].

l(i)

Notation. *Given a vector l of dimension d_l, let's denote the i^{th} coordinate of l as $l(i)$.*

Neural Network

$\mathcal{L}$

Neural Networks are supervised machine learning algorithms which means they assume there is a known training dataset. Briefly, given a training dataset $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ with $(x_i, y_i) \in \mathbb{R}^{d_0} \times \mathbb{R}^k$ for $1 \leq i \leq m$, $d_0, k \in \mathbb{N}$, the construction of a neural network of $n \geq 1$ layers (of dimensions $d_1, \ldots, d_n \in \mathbb{N}$, with $d_n = k$) given $g_1, \ldots, g_n$ nonlinear functions from $\mathbb{R}$ to $\mathbb{R}$, consists of finding coefficients $b_i^{(j)}, w_{i,r}^{(j)} \in \mathbb{R}$ for $j \in \{1, \ldots, n\}$, $i \in \{1, \ldots, d_j\}$, $r \in \{1, \ldots, d_{j-1}\}$ that minimize some function $\mathcal{L}$ that takes them as arguments. Given $x = l_0^{(x)} \in \mathbb{R}^{d_0}$, those coefficients define n vectors $l_1^{(x)}, \ldots, l_n^{(x)} \in \mathbb{R}^{d_1}, \ldots, \mathbb{R}^{d_n}$ recursively as

$$l_j^{(x)}(i) = g_j\Big(\sum_{r=1}^{d_{j-1}} w_{i,r}^{(j)} l_{j-1}^{(x)}(r) + b_i^{(j)}\Big)$$

for $j \in \{1, ..., n\}$, $i \in \{1, ..., d_j\}$. The function $\mathcal{L}$ is usually referred as the *loss function*, a typical example is the SSE [17]:

loss function

$$\min \sum_{t=1}^{m} (y_t - l_n^{(t)})^2$$

Once the coefficients $b_i^{(j)}, w_{i,r}^{(j)}$ that minimize the loss function, at least locally, are found (by initializing them at random and solving the optimization problem with a gradient descent type method [16]) they are fixed, and the predictor function f of the neural network is defined as $f(x) = l_n^{(x)}$. Finally, it should be said that the term *deep* refers to many layers in the neural network, which was impractical in the 1980s because of the great computing power they require.

Returning to the PLTSPHD problem and now that we have discussed how neural networks work, the question that now comes to mind is how to get a training set for the problem? That would be impractical because it would require calculating many optimal solutions in some way and the current state of the art exact algorithms take hours and cannot solve problems with more than 40 orders [26], [35]. The good news is that it is not the case, it is not required to calculate the optimal solutions because this is exactly where reinforcement learning comes into play.

Reinforcement learning is not a supervised model, but it is also not an unsupervised model because it does not try to find hidden features in a set of data as supervised models do, but rather it aims to optimize some function (called the reward function). Hence, it is considered the third machine learning paradigm [31]. Informally, the way in which it works can be explained with a diagram:

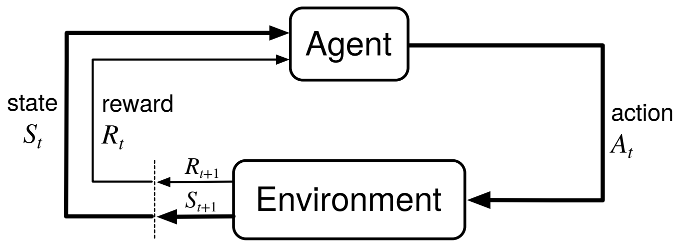

Figure 13, source: [31]

An agent (for example, a traveling salesman) is in some state and it must take some action from a set of available actions given its current state. The action to take can be decided at random or based on some policy. After the action is taken the environment is modified, so the state of the agent is updated to a new state, and a reward function assigns a value given the action that was taken. By a trial and error approach the agent is expected to learn a good policy to maximize rewards.

The problem to solve is to find a policy that maximizes the sum of the rewards, one popular strategy is to use dynamic programming [31].

The construction of the DRL algorithm to solve this problem belongs to the field of artificial intelligence which is a different area from the subject of this work. We just need to remember that the DRL model only takes as input the centroids of the clustering, and it outputs the route that the truck will follow to visit all the centroids.

More interesting for us are the experimental results generated with the model which we are going to analyze.

The algorithms were tested experimentally with random locations in Wichita, Kansas. Three region sizes were considered:

1. small: 70 clients in a 258 km^2 area.
2. medium: 110 clients in a 259 km^2 area.
3. large: 250 clients in a 518 km^2 area.

Each client c was assigned, at random, an order size of $q_c \in \{1, 2, 3, 4\}$. Hence, at the end, the demand by region was established to be:

1. small: 127 packages.
2. medium: 205 packages.
3. large: 350 packages.

We suppose that the weight of any package is within the payload of a drone, that the truck has unlimited space to carry all the packages, that any drone has a traveling cost of $1.38 pesos per km and a fixed cost of $206.62 pesos, and, that the truck has a traveling cost of $38.23 pesos per km, and a fixed cost of $1,721.85 pesos. There is a fixed cost of $17.22 pesos every time the truck parks. Those costs were not assigned at random, instead they are the costs commonly found in the literature [1].

The DRL algorithm was tested against the Nearest Neighbor algorithm (NN), a genetic algorithm (GA), and Google OR-tools solver. Each algorithm was run 30 times.

GA

First, the delivery problems were solved without any drones, just with one truck that delivers all the packages, a VRP$_1$, and the costs were,

Region	NN	GA	OR-tools	DRL
Small	6,339.34	6,244.29	6,125.48	6,088.63
Medium	7,417.90	7,204.22	7,109.69	7,038.41
Large	12,856.02	12,513.37	12,461.72	12,418.33

Table 2, source: [1]

1 United States Dollar = 17.2185 Mexican pesos.

At first sight those results do not look interesting at all, for instance, in the large region the improvement of the DRL was less than 5% compared to the NN. But we must remember this is a work about drones, hence, let's see what happens with the same regions but this time by considering the truck with three fleets of drones $m = 5, 10, 15$.

Region	Number of drones	Average number of clusters	NN	GA	OR-tools	DRL
Small	5	28	5,179.67	5,132.32	5,070.33	5,032.45
	10	15	5,180.19	5,156.42	5,130.60	5,104.08
	15	11	5,814.17	5,790.41	5,790.41	5,790.41
Medium	5	47	6,036.81	5,941.76	5,870.48	5,799.19
	10	24	5,664.89	5,593.60	5,546.08	5,522.32
	15	19	6,124.96	6,075.03	6,022.86	5,991.87
Large	5	79	8,412.61	8,222.52	8,008.67	7,961.15
	10	36	6,866.74	6,780.30	6,724.17	6,676.65
	15	25	7,638.13	7,519.32	7,448.03	7,424.27

Table 3, source: [1]

These results are much more interesting, they show the superiority of the DRL for the PLTSPHD, but more interesting for us is that even the results of the NN solution with drones shows a significant saving compared to the NN solution with only one truck that we saw in the previous table. In the case of the small region, the NN with a fleet of 5 drones has an 18.29% saving compared to the NN without drones, in the case of a medium region the NN with a fleet of 10 drones has a 23.63% saving compared to the NN without drones, and in the large region the NN with 10 drones has a 46.59% saving compared to the NN without drones.

In conclusion, these results further support the proposal we have made through this work about the great potential that Unmanned Aerial Vehicles represent, at least, in package delivery. And we have also provided an insight into the mathematics that are at the core of any drone package delivery problem.

Finally, we will mention that this was just a brief example of a DRL application to solve a combinatorial optimization problem. But DRL has matured to a point that there are now job offerings (PhD positions) which aim to tackle combinatorial optimization problems with DRL[1].

[1] Bielefeld University.

Future work

Currently, there are two schools of thought: those who think that UAV models should be nothing more than extensions of TSP and VRP models and those who believe that UAV are a groundbreaking technology, and their models should be addressed in a separate way.

Most proposed UAV models belong to the first school of thought, where constraints such as battery capacity and payload are neglected. However, things become more interesting, and difficult, when we address UAV as a different technology and put special attention to their specific constraints and requirements such as battery capacity, battery life (cycles of charge), payload, energy consumption, speed, landing and launching location requirements, etc.

Since UAV applications, such as package delivery, are not still in widespread use there is a limited amount of data to compare how realistic the current theoretical models are, hence, scientists working with UAV models should perform a careful sensitivity analysis considering the attributes we have mentioned. It will not be until UAV applications execute on a large scale that we will know if the considered assumptions were enough [24].

Now, the most important unaddressed issues that require extensive research are:

1. Battery usage/capacity: Battery duration is strongly dependent on the payload of the drones, the speed at which they fly, and the wind resistance exerted upon them [23], [36].
2. Temporal restrictions: These can be related to adverse weather with strong wind or rain that makes it impossible for the UAVs to fly safely or some temporal airspace closure implemented by the airspace authorities [24], [36].
3. Objective functions related to economic costs: Most UAV models aim to minimize the time of the last vehicle to return, or minimize the flying time of drones, or minimize the travel length of trucks, etc. Once a solution is found, the economic cost of the solution is calculated. There is a lack of models that use the economic cost as the initial objective function [23], [24].
4. Synchronization problems: These problems can range from the most basic scenarios such as collision avoidance and launching/landing synchronization to more complex problems related to combined operations such as in-flight payload transfer and in-flight battery switching [23], [24].
5. Dynamic path replanning: If the weather becomes adverse, or in a malfunction event, or if a pickup truck takes longer than some threshold, UAVs need to modify their scheduled path accordingly [24].
6. Multiple tasks per trip: Multi-parcel delivery, delivery and pickup, delivery while mapping a region [1], [15], [27].
7. Heterogeneous fleets of drones: Trucks carrying distinct types of drones, some of those drones can fly longer distances while others can lift heavier packages [24].
8. Safety and Risks: Identify the potential risks associated with UAVs carrying parcels above people and households, as well as the risk associated with the deliveries itself [36].

References

[1] Arishi A, Krishnan K, Arishi M. Machine learning approach for truck-drones based last-mile delivery in the era of industry 4.0. Engineering Applications of Artificial Intelligence. 2022 Nov 1;116:105439.

[2] Arthur D, Vassilvitskii S. K-means++ the advantages of careful seeding. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms 2007 Jan 7 (pp. 1027-1035).

[3] Arora K, Agarwal S, Tanwar R. Solving TSP using genetic algorithm and nearest neighbour algorithm and their comparison. International Journal of Scientific & Engineering Research. 2016 Jan;7(1):1014-8.

[4] Bangert PD. The Traveling Salesman Problem. David L. Applegate, Robert E. Bixby, Vašek Chvátal, William J. Cook. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ 2006.

[5] Bellmore M, Nemhauser GL. The traveling salesman problem: a survey. Operations Research. 1968 Jun;16(3):538-58.

[6] Boysen N, Briskorn D, Fedtke S, Schwerdfeger S. Drone delivery from trucks: Drone scheduling for given truck routes. Networks. 2018 Dec;72(4):506-27.

[7] Bronts MM. Giving a step-by-step reduction from SAT to TSP and giving some remarks on Neil Tennant's Changes of Mind' (Thesis, Faculty of Science and Engineering). 2014.

[8] Christofides N. Worst-case analysis of a new heuristic for the travelling salesman problem. In Operations Research Forum 2022 Mar 3 (Vol. 3, No. 1, p. 20). Cham: Springer International Publishing.

[9] Cook WJ. In pursuit of the traveling salesman: mathematics at the limits of computation. Princeton university press; 2012.

[10] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. MIT press; 2022 Apr 5.

[11] de la Cruz LE, Báñez JM. Algorithmic and combinatorial problems on multi-UAV systems (Thesis, Universidad de Sevilla). 2019.

[12] Drineas P, Frieze A, Kannan R, Vempala S, Vinay V. Clustering large graphs via the singular value decomposition. Machine learning. 2004 Jul;56:9-33.

[13] Erlebach T, Luo K, Spieksma FC. Package Delivery Using Drones with Restricted Movement Areas. arXiv preprint arXiv:2209.12314. 2022 Sep 25.

[14] Glover F, Gutin G, Yeo A, Zverovich A. Construction heuristics for the asymmetric TSP. European Journal of Operational Research. 2001 Mar 16;129(3):555-68.

[15] Golden BL, Raghavan S, Wasil EA, editors. The vehicle routing problem: latest advances and new challenges. New York: Springer; 2008 Jul 20.

[16] Grohs P, Kutyniok G, editors. Mathematical Aspects of Deep Learning. Cambridge University Press; 2022 Dec 22.

[17] Hastie T, Tibshirani R, Friedman JH, Friedman JH. The elements of statistical learning: data mining, inference, and prediction. New York: springer; 2017 Jan.

[18] James G, Witten D, Hastie T, Tibshirani R. An introduction to statistical learning. New York: Springer; 2021.

[19] Jana S, Mandal PS. Approximation Algorithms for Drone Delivery Packing Problem. In 24th International Conference on Distributed Computing and Networking 2023 Jan 4 (pp. 262-269).

[20] Kanungo T, Mount DM, Netanyahu NS, Piatko CD, Silverman R, Wu AY. A local search approximation algorithm for k-means clustering. In Proceedings of the eighteenth annual symposium on Computational geometry 2002 Jun 5 (pp. 10-18).

[21] Murray CC, Chu AG. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. Transportation Research Part C: Emerging Technologies. 2015 May 1;54:86-109.

[22] Ong HL, Moore JB. Worst-case analysis of two travelling salesman heuristics. Operations Research Letters. 1984 Mar 1;2(6):273-7.

[23] Otto A, Agatz N, Campbell J, Golden B, Pesch E. Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. Networks. 2018 Dec;72(4):411-58.

[24] Poikonen S, Campbell JF. Future directions in drone routing research. Networks. 2021 Jan;77(1):116-26.

[25] Poikonen S, Wang X, Golden B. The vehicle routing problem with drones: Extended models and connections. Networks. 2017 Aug;70(1):34-43.

[26] Roberti R, Ruthmair M. Exact methods for the traveling salesman problem with drone. Transportation Science. 2021 Mar;55(2):315-35.

[27] Rojas Viloria D, Solano-Charris EL, Muñoz-Villamizar A, Montoya-Torres JR. Unmanned aerial vehicles/drones in vehicle routing problems: a literature review. International Transactions in Operational Research. 2021 Jul;28(4):1626-57.

[28] Rosenkrantz DJ, Stearns RE, Lewis, II PM. An analysis of several heuristics for the traveling salesman problem. SIAM journal on computing. 1977 Sep;6(3):563-81.

[29] Sipser M. Introduction to the Theory of Computation. Cengage Learning; 2012 Jun 27.

[30] Sneyers J, Schrijvers T, Demoen B. Dijkstra's algorithm with Fibonacci heaps: An executable description in CHR. In Proceedings of the 20th Workshop on logic programming 2006 (Vol. 1843, pp. 182-191). Technische Universität Wien, Austria.

[31] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT press; 2018 Nov 13.

[32] Tan PN, Steinbach M, Kumar V. Introduction to data mining (2nd Edition). Pearson Education India; 2019.

[33] Toth P, Vigo D, editors. Vehicle routing: problems, methods, and applications. Society for industrial and applied mathematics; 2014 Nov 24.

[34] Wang X, Poikonen S, Golden B. The vehicle routing problem with drones: several worst-case results. Optimization Letters. 2017 Apr;11:679-97.

[35] Yang Y, Yan C, Cao Y, Roberti R. Planning robust drone-truck delivery routes under road traffic uncertainty. European Journal of Operational Research. 2023 Sep 16;309(3):1145-60.

[36] Zheng C. Review on Combined Vehicle Routing Problem of Drones and Vehicles. Journal of Innovation and Development. 2023 Feb 22;2(1):55-7.

Appendix

Greedy algorithm

Run this *greedy.py* file with:

```
$ python3 greedy.py
```

The code assumes there is a file named *instance.txt* in the same directory as the file with this code. In the first line it indicates the size of the distance matrix, and then, the matrix (of positive real numbers). No other assumption is made (not even the triangle inequality). For example:

```
3
21  12  9
8    3  2
1    6  3
```

At the end, we will provide a random instance generator.

```
import numpy as np
import sys
np.set_printoptions(threshold=sys.maxsize)

### Here modify the name of the instance if it is different than instance.txt:
File = 'instance.txt'
```

We call D the given matrix. All entries (i, i) are assigned the infinity value to avoid loops later.

```
size = np.genfromtxt(File, dtype=int, max_rows=1)
D = np.genfromtxt(File, skip_header=1)
for i in range(size):
    D[i][i] = float('inf')
```

First, we create a list L which will have all the entries of the given weight matrix D but ordered according to their weights. $L = [(a,b)|a,b < size(D)]$ and (a,b) appears before (c,d) if and only if $D[a][b] \leq D[c][d]$. We order with Merge sort which takes $O(n \lg n)$. Writing the entries to L takes $O(n^2)$, hence this whole operation is in $O(n^2)$.

```
D = np.array(D)
ord1 = D.argsort(axis=None, kind='mergesort')
ord2 = np.unravel_index(ord1, D.shape)
L = np.vstack(ord2).T
```

The T variable will be the set of edges of the desired tour, it will have all the edges but in no particular order. We append to T the first element of L which is the edge of the graph of minimum value, and we initialize a weight variable.

```
T = []
T.append(list(L[0]))
weight = D[L[0][0]][L[0][1]]
```

We create a list of pairs of numbers named *originDestiny*. Given an index j, the element $T[j] = [s, e]$ records the first element s and the last element e of the longest path of T which contains j. This list will be used later in the main loop to avoid cycles.

```
originDestiny =  [[x,x] for x in list(range(size))]
originDestiny[L[0][0]][1] = L[0][1]
originDestiny[L[0][1]][0] = L[0][0]
```

Every time we put an element $[i, j]$ to T, we update the matrix D so that all the entries in the i-row and in the j-column have infinity weight so we never need to use them again. Since we just added $L[0]$ we need to update D:

```
for u in range(size):
    D[u][L[0][1]] = float('inf')
    D[L[0][0]][u] = float('inf')
np.delete(L, 0)
```

Now we will use the previous procedure but for all the matrix: we take the first element i of L with weight less than infinity, then by using the *originDestiny* list we check if that edge is not creating a loop in which case we add it to T and update the *originDestiny* list according to the new *origin* and *destiny* of the path (i.e., the new first and last vertex of the two joined paths). The *flgs* variable will indicate when there is only one element left in the array (i.e., only the last edge which closes the path to create the tour). The most expensive operations are the matrix writings: n values in a row + n values in a column $= 2n$. But we only execute the update procedure once for each edge of the path, i.e., at most n times. Hence $O(2n \cdot n) = O(n^2)$.

```
flgs = size-1
for i in range(len(L)):
    if flgs == 1:
        break
    if (D[L[i][0]][L[i][1]] != float('inf') and L[i][1] != originDestiny[L[i][0][0]):
        originDestiny[originDestiny[L[i][1]][1]][0] = originDestiny[L[i][0]][0]
        originDestiny[originDestiny[L[i][0]][0]][1] = originDestiny[L[i][1]][1]
        flgs = flgs - 1
        T.append(list(L[i]))
        weight += D[L[i][0]][L[i][1]]
        for u in range(size):
            D[u][L[i][1]] = float('inf')
            D[L[i][0]][u] = float('inf')
```

Now T is the set of all the edges in the maximum path but in no particular order, so we must order T to generate the final solution. We pick the first edge of T $[i, j]$ and add it to *Solution*, then we remove it from T and iterate T to find the next edge which either starts with j or ends with i and attach it to *Solution*: at the end in the first case or at the beginning in the second case.

We repeat this process until T is empty, at most it takes
$n + (n - 1) + (n - 2) + ... + 1 = \frac{n(n+1)}{2} = O(n^2)$ operations.

```
Solution = []
Solution.append(T[0][0])
Solution.append(T[0][1])
T.remove(T[0])
while T != []:
    for x in T:
        if x[0] == Solution[-1]:
            Solution.append(x[1])
            T.remove(x)
        elif x[1] == Solution[0]:
            Solution.insert(0,x[0])
            T.remove(x)
```

Then we just close the path to get the cycle and sum 1 to each vertex to have the numeration starting in 1 instead of 0,

```
weight += D[Solution[-1]][Solution[0]]
Solution.append(Solution[0])
Solution = np.array(Solution)
Solution += 1
print("Greedy:")
print(Solution)
print(weight)
print()
```

Random instances generator

A small program was written to create instances of a given size with random integers between 1 and 100. It prints the result in the format required for the Greedy program.

The only parameter that might need to be modified (in the code) is the size of the desired matrix. It is run with the command:

```
$ python3 randM.py
```

```
import random
# Here we edit the desired size of the matrix:
size = 10

# Random matrix
matrix = [[random.randint(1, 100) for j in range(size)] for i in range(size)]

#Result printed in desired format for greedy.py program:
print(size)
for y in matrix:
    print(" ".join(str(x) for x in y))
```

Index