UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS SUPERIORES,
UNIDAD MORELIA

IMPLEMENTACIÓN DE UNA METAHEURÍSTICA PARA RESOLVER UN PROBLEMA DE *clustering* BALANCEADO MULTI-OBJETIVO APLICADO EN LA LOGÍSTICA DE ENTREGAS

T E S I S

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN TECNOLOGÍAS PARA LA INFORMACIÓN EN CIENCIAS

P R E S E N T A

EDUARDO MANUEL CEJA CRUZ

TUTORA

DRA. ADRIANA MENCHACA MÉNDEZ

CO-TUTORA

DRA. ELIZABETH MONTERO URETA

MORELIA, MICHOACÁN AGOSTO 2023

MTRA. IVONNE RAMÍREZ WENCE
DIRECTORA
DIRECCIÓN GENERAL DE ADMINISTRACIÓN ESCOLAR
P R E S E N T E

Por medio de la presente me permito informar a usted que en la **sesión ordinaria 04** del **Comité Académico** de la **Licenciatura en Tecnologías para la Información en Ciencias** de la Escuela Nacional de Estudios Superiores (ENES), Unidad Morelia, celebrada el día **25 de mayo de 2023**, se acordó poner a su consideración el siguiente jurado para la presentación del Trabajo Profesional del alumno **Eduardo Manuel Ceja Cruz** de la Licenciatura en **Tecnologías para la Información en Ciencias,** con número de cuenta **41812858-3**, con el trabajo titulado: **"Implementación de una metaheurística para resolver un problema de clustering balanceado multi-objetivo aplicado en la logística de entregas",** bajo la dirección como tutora de la **Dra. Adriana Menchaca Méndez** y como co-tutora la **Dra. Elizabeth Montero Ureta.**

El jurado queda integrado de la siguiente manera:

Presidente:	Dra. María del Río Francos
Vocal:	Dr. Daniele Colosi
Secretario:	Dra. Adriana Menchaca Méndez
Suplente:	Dr. Sergio Rogelio Tinoco Martínez
Suplente:	Dr. Saúl Zapotecas Martínez

Sin otro particular, quedo de usted.

A t e n t a m e n t e
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Morelia, Michoacán a 04 de septiembre de 2023.

DRA. YUNUEN TAPIA TORRES
SECRETARIA GENERAL

Agradecimientos Institucionales

Agradezco formalmente a la Licenciatura en Tecnologías para la Información en Ciencias, así como a la Escuela Nacional de Estudios Superiores Unidad Morelia de la Universidad Nacional Autónoma de México por proveer educación gratuita, moderna y de calidad, así como por darme la oportunidad de realizar y concluir mis estudios en la institución. Igualmente agradezco al Programa UNAM-DGAPA-PAPIME PE106923 por el apoyo proveído para la realización de este trabajo de tesis. Finalmente, agradezco a mis tutoras y miembros del jurado por su tiempo dedicado y retroalimentación a este trabajo:

- Dra. María del Río Francos
- Dr. Daniele Colosi
- Dr. Sergio Rogelio Tinoco Martínez
- Dr. Saúl Zapotecas Martínez
- Dra. Adriana Menchaca Méndez
- Dra. Elizabeth Montero Ureta

Agradecimientos Personales

Dedicada a ....

Mi familia

Índice general

Índice de figuras

Índice de tablas

Resumen

En este trabajo de tesis estamos interesados en el problema de agrupamiento balanceado aplicado en la logística de entregas. Dicho problema se modeló como un problema de optimización bi-objetivo. La primera función objetivo busca crear grupos balanceados con respecto al tiempo que toma recorrer todos los puntos de un grupo. Mientras que la segunda función objetivo busca que los grupos estén balanceados con respecto a las cargas que se deben transportar. Para poder resolver este problema se utilizó un algoritmo multi-objetivo basado en recocido simulado conocido como "Archived Multiobjective Simulated Annealing (AMOSA)". Dada la complejidad computacional de las funciones objetivo, se implementaron versiones que realizan cálculos en paralelo, logrando una reducción de hasta el 13 % en el tiempo de ejecución del algoritmo. Para probar la eficiencia del algoritmo implementado se utilizaron datos de dos estados mexicanos, Morelos e Hidalgo, y se comparó con respecto a un algoritmo evolutivo multi-objetivo llamado NSGA-II. Los resultados experimentales muestran que el algoritmo propuesto obtiene mejores resultados tanto en la calidad de las soluciones como en los tiempos de ejecución.

Abstract

In this work, we are interested in the balanced cluster problem for delivery logistics. This problem is modeled as a bi-objective optimization problem. The objective of the first function is to create groups that are well-balanced according to the time of visit to their points. In contrast, the second function aims to balance the groups concerning the weight to be transported. To solve the bi-objective problem, we use a multiobjective algorithm based on Simulated Annealing called "Archived Multiobjective Simulated Annealing (AMOSA)." Due to the computational cost of the objective functions, we introduced a parallel version of the AMOSA framework to reduce the algorithm's running time, obtaining an improvement of 13 %. The algorithm was evaluated in a real-world scenario using data from two Mexican states, Hidalgo and Morelos. AMOSA was compared to a popular Pareto-based MOEA. Preliminary results show that the proposed approach outperformed the well-known NSGA-II concerning the quality of solutions and execution time.

Capítulo 1

Introducción

El seccionamiento de zonas geográficas es una labor que se realiza para hacer la logísticas de las entregas mas fácil, es gracias a estas ideas que las regiones postales existen, así como el sistema de código postal, ya que permiten tener un mejor control sobre dónde y cómo repartir. Sin embargo, debido al comercio en línea y al gran crecimiento industrial y urbano en los últimos años, se ha visto la necesidad de resolver el problema tomando en cuenta los tiempos de entrega. A partir de esto último surge el problema de balanceo de grupos para el problema logístico.

1.1. Antecedentes

Múltiples enfoques han sido usados para poder obtener grupos balanceados con respecto al número de puntos. Por ejemplo, en He, Xu, Sun, y Zu (2009) usan k-means para resolver un problema de enrutamiento de vehículos de la siguiente forma. Primero, se definen las áreas iniciales usando k-means. Posteriormente, se evalúan los grupos usando su cardinalidad. Finalmente, se transfieren puntos de los grupos más grandes a los mas pequeños hasta satisfacer una restricción de balance.

En Lin et al. (2022), se presenta otro enfoque usando k-means. En lugar de asignar cada punto al centroide más cercano, definen $|Q|/k$ como el máximo número de puntos en cada grupo, donde $|Q|$ es el tamaño del conjunto de datos y k representa el número de grupos . Si el grupo del centroide más cercano está lleno, entonces el punto se pasa al siguiente grupo más cercano.

1.2. Justificación

Las propuestas antes mencionadas no pueden ser aplicadas al problema de agrupamiento en la logística de entregas. El objetivo, en este último, es balancear los

grupos con respecto a la carga de trabajo en los agrupamientos. Esta carga de trabajo no está basada en la cardinalidad del conjunto, sino en el tiempo que se tarda en recorrer el grupo o en la cantidad/peso del producto que se transporta. Recientemente, se publicó una propuesta para resolver el balanceo de grupos con respecto al tiempo de recorrido (Menchaca-Méndez, Montero, Flores-Garrido, y Miguel-Antonio, 2022). Sin embargo, ese trabajo no considera la parte de balanceo con respecto a la cantidad/peso del producto transportado, el cual es un objetivo interesante para las empresas. Debido a que la optimización del tiempo y la carga están en conflicto entre ellos, se necesita resolver un problema de optimización multi-objetivo. Hasta donde sabemos, no se han presentado trabajos que traten de balancear grupos considerando estos dos objetivos a la vez. Además, es importante mencionar que los tiempos de ejecución del algoritmo propuesto no deben exceder de 24 horas, esto debido a las necesidades de las empresas.

1.3. Objetivo General

Implementar una metaheurística para resolver el problema de *clustering* multi-objetivo balanceado relacionado con la logística de entrega de productos.

1.4. Objetivos Específicos

1. Implementar una metaheurística para resolver problemas de optimización multi-objetivo.

2. Adaptar la metaheurística del punto anterior para abordar el problema de *clustering* balanceado en la logística de entrega de productos.

3. Evaluar la metaheurística propuesta utilizando la información de los puntos de venta de al menos dos Estados de la República Mexicana registrados en el Instituto Nacional de Estadística, Geografía e Informática (INEGI).

4. Disminuir los tiempos de ejecución de la metaheurística aplicando técnicas de paralelización de tareas.

1.5. Estructura del trabajo

En el Capítulo 2 se presentan algunos de los conceptos más importantes en optimización así como algunos de los métodos que se han utilizado para resolver este tipo de problemas. En el Capítulo 3 se estudia el problema de *clustering* y se define

el problema de clustering balanceado que se abordará en este trabajo de tesis. En el Capítulo 4 se estudia un algoritmo de optimización multi-objetivo basado en la metaheurística de recocido simulado y en el Capítulo 5 se adapta para resolver el problema de clustering balanceado aplicado a la logística de entregas. En el Capítulo 6 se muestran los resultados de los experimentos realizados a lo largo de este trabajo de tesis. Finalmente, en el Capítulo 7 se presentan conclusiones y trabajo futuro.

Capítulo 2

Optimización

La optimización, conocida también como programación matemática, consiste en la búsqueda del mejor elemento, respecto a un criterio, dentro de un conjunto de elementos disponibles. Este tipo de problemas tienen numerosas aplicaciones en distintas áreas del conocimiento como lo son: ciencias de la computación, economía, ingenierías, investigación de operaciones, entre otras. En otras palabras, un problema de optimización consiste en encontrar la solución que **minimiza** o **maximiza** una función dentro de un dominio determinado. Formalmente, se puede definir un problema de optimización de la siguiente manera

Definición 2.0.1 (Problema de optimización). Sea A el conjunto de búsqueda del problema, $f : A \to \mathbb{R}$ una función y $\Omega \in A$ el conjunto de puntos que satisfacen las restricciones del problema. Se busca un elemento $x^* \in \Omega$ tal que $f(x^*) \leq f(x)\ \forall x \in \Omega$ en el caso de minimizar o $f(x^*) \geq f(x)\ \forall x \in \Omega$ en el caso de maximizar.

Con lo anterior, se puede decir que $f(x^*) \geq f(x) \iff -f(x^*) \leq f(x)$, lo que permite que un problema de maximización se pueda convertir a uno de minimización y viceversa. Suponiendo un problema de minimización, tenemos las siguientes definiciones:

- Un punto $x \in A$ es un **mínimo global** si y solo si $f(x) \leq f(y)\ \forall y \in A$.
- Un punto $x \in A$ es un **mínimo local** si y solo si $f(x) \leq f(y)\ \forall y \in N(x) \cap A$ donde $N(x)$ es el vecindario del punto x.

2.1. Condiciones de Optimalidad

Cuando la función que se desea optimizar cumple con ciertas características, es posible verificar si un punto x es un óptimo utilizando las siguientes condiciones:

- **Condiciones necesarias**: Sea $f : (a, b) \to \mathbb{R}$, entonces $x^* \in (a, b)$ es un candidato a ser un mínimo local si f es diferenciable en x^*, y su derivada es igual con cero, $f'(x^*) = 0$.
- **Condiciones suficientes**: Sea $f : (a, b) \to \mathbb{R}$ dos veces diferenciable en x^*, entonces $x^* \in (a, b)$ es un mínimo local si su derivada es igual con cero, $f'(x^*) = 0$, y su doble derivada mayor a cero, $f''(x^*) > 0$.

La mayoría de los métodos de optimización clásicos, están enfocados en encontrar los puntos que satisfagan las condiciones necesarias y suficientes.

2.2. Métodos de optimización

Existe una gran variedad de métodos para resolver problemas de optimización, esto es porque un método no puede resolver eficientemente todos los tipos de problemas de optimización.

Los problemas de optimización pueden ser clasificados con base a las características que poseen. Por ejemplo, si tanto su función como sus restricciones son lineales, se considera un problema de *programación lineal*. Si la función objetivo es cuadrática y las restricciones lineales, se trata de un problema de *programación cuadrática*. Si su función o alguna de sus restricciones no son lineales, es un problema de *programación no lineal*, etc. Por otro lado, si nos fijamos en el tipo de variables que utilizan, podemos tener problemas de *programación entera*, cuando las variables solo pueden tomar valores enteros y problemas de *optimización continua*, cuando las variables toman valores reales, etc.

2.2.1. Optimización no lineal

Existen varios métodos clásicos para resolver problemas de optimización no lineal. En "Nonlinear Programming I: One-Dimensional Minimization Methods" (2019) se pueden consultar los siguientes métodos: búsqueda exhaustiva, fase de acotamiento, división de intervalos por la mitad, método de Fibonacci, sección dorada, estimaciones cuadráticas, Newton-Raphson, bisecciones y método de la secante, entre otros. Para problemas de varias variables se pueden consultar los siguientes métodos en "Nonlinear Programming II: Unconstrained Optimization Techniques" (2019): operación evolutiva, Simplex, Hooke-Jeeves, Powell, etc. Las principales desventajas de estos métodos es que a lo más aspiran a obtener óptimos locales y algunos de ellos no se pueden aplicar a todo tipo de problemas, por ejemplo, cuando la función objetivo no es diferenciable.

2.2.2. Metaheurísticas

Cuando no es posible aplicar un método clásico porque el problema de optimización que se quiere resolver ya sea porque es multimodal (tiene varios óptimos locales), la función objetivo no es diferenciable o, incluso, no puede expresarse de forma algebraica, se puede recurrir al uso de heurísticas. Una heurística es un método que busca buenas soluciones a un problema en un tiempo de cómputo razonable.

Una metaheurística es una heurística de propósito general, es decir, que no es diseñada para resolver un problema de optimización en particular, sino más bien un problema de optimización general. Es importante recordar que un método de optimización no puede resolver cualquier tipo de problema (No Free Lunch Theorem) (Wolpert y Macready, 1997), por lo que, es importante ajustar los componentes de una metaheurística para mejorar su desempeño en un problema particular. Existen varias metaheurísticas ampliamente utilizadas, a continuación se describen algunas.

Recocido Simulado

El Recocido Simulado (RS o SA por sus siglas en ingles), es una metaheurística que está basada en el proceso físico de calentar el acero o las cerámicas para posteriormente dejarlos enfriar lentamente hasta alcanzar una configuración de mínima energía. Su objetivo es el de encontrar mínimos globales en funciones que tienen múltiples mínimos locales. Este algoritmo se ha usado para resolver diversos problemas complicados, como lo han sido el problema del viajero, problema de coloreado de grafos, asignación cuadrática, el problema de la mochila y asignación de recursos y ubicaciones, diseño de rutas y redes, planeación, etc. Este algoritmo lo utilizaron por primera vez Kirkpatrick, Gelatt, y Vecchi (1983), para poder resolver el problema del viajero. El método está basado en el algoritmo Metrópolis, el cual permite emular de manera eficiente una colección de átomos en equilibrio a una temperatura dada (Metropolis, Rosenbluth, Rosenbluth, Teller, y Teller, 1953).

El algoritmo de RS puede verse como una cadena de Markov no homogénea, es decir, es un proceso estocástico discreto en el que la probabilidad de pasar de un estado a otro depende del estado anterior y del tiempo en el que se encuentre. Al ser un método que simula el recocido de la materia, al inicio, la temperatura va a ser elevada y va a provocar que la probabilidad de aceptar soluciones que no son mejores a la actual sea alta. Conforme la temperatura vaya descendiendo, la probabilidad de aceptar malas soluciones irá decreciendo. Esto le permite al algoritmo salir de óptimos locales y poder aspirar a encontrar el óptimo global.

Búsqueda Tabú

La Búsqueda Tabú (BT) es una metaheurística enfocada en la búsqueda local, la cual tiene incorporada una memoria de lugares que ya ha visitado y que cataloga como tabú, de ahí su nombre. La memoria tabú permite mejorar la estrategia de búsqueda y escapar de óptimos locales. Este algoritmo fue propuesto por Glover (1986) y formalizado en 1989 (Glover, 1989, 1990).

BT se ha usado para resolver problemas de optimización combinatoria. Algunos ejemplos de este tipo de problemas son el problema del viajero, el problema de la mochila, el de asignación cuadrática, problema del clique, entre otros. En Karp; Koopmans y Beckmann se pueden consultar las definiciones de los problemas mencionados. También ha sido utilizado para problemas de la vida real como lo han sido: planeación de recursos, telecomunicaciones, diseño de circuitos integrados, análisis financiero, por nombrar algunos ejemplos.

BT parte de una solución inicial x, de ahí procede a iterar una serie de pasos hasta alcanzar un criterio de paro: Primero define un vecindario alrededor de la solución actual x llamado $N(x)$. Posteriormente, utilizando la información de la memoria tabú y del vecindario decide a qué punto moverse. Una vez que se mueve al siguiente punto actualiza la memoria tabú.

Algoritmos Evolutivos

Los Algoritmos Evolutivos (AEs) son metaheurísticas que están inspiradas en el proceso evolutivo de los seres vivos. Por lo anterior, se basan en cuatro procesos principales: la reproducción, la mutación, la competencia y la selección. Los AEs pertenecen al área de Inteligencia Artificial porque se considera el proceso de adaptación como un tipo de inteligencia que permite a los individuos predecir y sobrevivir en el ambiente que se desenvuelven. En los AEs las soluciones candidatas del problema de optimización juegan el papel de individuos en poblaciones, y la calidad de cada individuo está determinada por una función de aptitud (*fitness*) que se define a partir de la función objetivo que se desea optimizar. Los individuos de la población se reproducen a través de operadores de cruza y mutación y compiten entre ellos para sobrevivir. En los inicios de la computación evolutiva surgieron tres paradigmas principales: programación evolutiva, estrategias evolutivas y algoritmos genéticos. Sin embargo, a través de los años se han propuesto otros AEs que han resultado muy eficientes, por ejemplo, optimización por enjambre de partículas (Kennedy, 1997) y evolución diferencial (Storn y Price, 1997).

2.3. Optimización Multi-Objetivo

Los problemas que tienen más de una función objetivo a optimizar se conocen como *problemas de optimización multi-objetivo (POM)*. En este tipo de problemas, las funciones objetivo están en conflicto entre sí, por lo que se aspira a obtener un conjunto de soluciones tales que, un objetivo no puede ser mejorado sin empeorar otro. Suponiendo problemas de minimización, se puede definir un POM de la siguiente manera:

$$\min_{\vec{x}\in\Omega} \vec{F}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \cdots, f_M(\vec{x}))^T$$

donde $M \geq 2$ es el número de funciones objetivo, Ω es el conjunto de soluciones factibles y $\vec{F}$ se define como el vector de las funciones objetivo. Si trabajamos con problemas de optimización continua tenemos que $f_i : \mathbb{R}^M \to \mathbb{R}$.

Dominancia y frentes de Pareto

En la optimización multi-objetivo no hay una solución factible que minimice todas las funciones objetivo simultáneamente. Por eso se le da atención a las soluciones que no pueden ser comparadas entre sí. A estas soluciones se les llama soluciones *no dominadas*, lo que quiere decir que ninguna es mejor que otra en todos los objetivos. Asumiendo la minimización de todos los objetivos, la dominancia se puede definir de la siguiente manera.

Definición 2.3.1 (Dominancia). Dados $\vec{x}, \vec{y} \in \Omega$, decimos que $\vec{x}$ **domina** a $\vec{y}$ ($\vec{x} \prec \vec{y}$) $\iff$ $f_i(\vec{x}) \leq f_i(\vec{y})\ \forall i \in \{1, 2, \ldots, M\}$ y $\vec{F}(\vec{x}) \neq \vec{F}(\vec{y})$.

Definición 2.3.2 (Solución Pareto óptima). Dada $\vec{x}^* \in \Omega$, decimos que $\vec{x}^*$ es una solución **Pareto óptima** si no hay otra solución $\vec{y} \in \Omega$ tal que $\vec{y} \prec \vec{x}^*$.

Definición 2.3.3 (Conjunto de Pareto). El **Conjunto de Pareto** se define como:

$$PS = \{\vec{x} \in \Omega \mid \vec{x} \text{ es una solución } \textbf{Pareto óptima}\}$$

Definición 2.3.4 (Frente de Pareto). El **frente de Pareto** es definido como:

$$PF = \{\vec{F}(\vec{x}) \mid \vec{x} \in PS\}$$

En la optimización multi-objetivo lo más deseable es obtener tantos elementos de PS como sea posible y que se mantenga una distribución uniforme de soluciones a lo largo de PF. Al igual que en la optimización mono-objetivo, existen varios métodos clásicos para resolver este tipo de problemas. Algunos de estos métodos convierten el problema multi-objetivo en uno mono-objetivo para, posteriormente, utilizar uno de

los métodos bien conocidos existentes para obtener una solución del PS. Además de heredar las desventajas de los métodos clásicos, su costo computacional es elevado, ya que se debe ejecutar varias veces para obtener diferentes soluciones del PF.

Por lo anterior, las metaheurísticas poblacionales han sido ampliamente utilizadas para resolver este tipo de problemas, pues una ejecución de ellas devuelve varias soluciones del PF. Algunos ejemplos de metaheurísticas populares son: NSGA-II (Deb, Pratap, Agarwal, y Meyarivan, 2002), SMS-EMOA (Beume, Naujoks, y Emmerich, 2007) y MOEA/D (Zhang y Li, 2007). Todos ellos son algoritmos evolutivos y su principal diferencia radica en el operador de selección. NSGA-II está basado en el concepto de dominancia de Pareto. SMS-EMOA optimiza un indicador llamado hipervolumen , el cual se utiliza para medir la calidad de una aproximación del frente de Pareto. Finalmente, MOEA/D descompone el problema multi-objetivo en varios problemas mono-objetivo, los cuales resuelve de manera simultánea.

Capítulo 3

Clustering

3.1. *Clustering*

La segmentación de datos, conocida comúnmente como *clustering*, consiste en dividir un conjunto de objetos en grupos. El objetivo es que los objetos de un mismo grupo estén más relacionados entre sí que con los objetos de los otros grupos. El grado de relación entre pares de objetos se calcula aplicando una medida de distancia. Existen varias aplicaciones de problemas de *clustering* (Hastie, Tibshirani, y Friedman, 2009; Lantz, 2015; Xu y Wunsch, 2010) por ejemplo:

1. Segmentación de clientes para campañas de marketing.
2. Análisis de datos para expresiones génicas, análisis de secuencias genómicas y otras aplicaciones de biomedicina.
3. Detección de comportamientos anómalos mediante la identificación de patrones de uso fuera de los grupos conocidos (filtros de spam, identificación de actividades fraudulentas, etc.)
4. Simplificación de conjuntos de datos muy grandes en conjuntos de datos más pequeños, etc.

Formalmente, podemos definir el problema de *clustering* como un problema de optimización (Handl y Knowles, 2006); que consiste en encontrar la k-partición C^* para la cual se cumple:

$$f(C^*) = \min_{C \in \Omega} f(C) \tag{3.1}$$

donde Ω es el conjunto de las k-particiones factibles; $C = \{C_1, \ldots, C_k\}$ es una k-partición de un conjunto finito de datos S; $C_1, \ldots C_k$ son conjuntos disjuntos y $f : \Omega \to \mathbb{R}$ es la función objetivo basada en la noción de similitud o disimilitud entre objetos.

3.1.1. Algoritmos de *Clustering*

En la actualidad, existen varios métodos de *clustering*, sin embargo, no todos estos métodos funcionan bien en cualquier conjunto de datos. Por ejemplo, k-means, k-medoids (J., 1967) y agrupamiento aglomerativo de enlace promedio (Voorhees, 1986) trabajan bien cuando los grupos están bien separados, pero fallan en estructuras de grupos más complejas. El agrupamiento aglomerativo de enlace simple (Voorhees, 1986) o los métodos basados en densidad (Ankerst, Breunig, Kriegel, y Sander, 1999; Ester, Kriegel, Sander, y Xu, 1996) trabajan bien en estructuras de grupos complejas pero fallan cuando la separación entre los grupos es muy pequeña o ambigua (un objeto puede estar asociado a más de un grupo). Las técnicas de agrupamiento difuso (Bezdek, 1981; Dembélé y Kastner, 2003; Krishnapuram y Keller, 1993) son útiles cuando la separación entre los grupos es ambigua.

Agrupamientos Aglomerativos

El agrupamiento aglomerativo es una técnica de agrupamiento ~~jerárquico~~ que busca crear jerarquías entre los grupos. Es aglomerativo porque empieza de "abajo hacia arriba", es decir, cada observación empieza como un *cluster* y, posteriormente, los *clusters* similares se van agrupando mientras se va subiendo de jerarquía.

En este método es posible calcular la similitud de las observaciones (elementos) utilizando cualquier medida válida de distancia. Algunos ejemplos de las más comunes son:

- Distancia euclidiana
$$||a - b||_2 = \sqrt{\sum_i (a_i - b_i)^2}$$
- Distancia euclidiana al cuadrado
$$||a - b||_2^2 = \sum_i (a_i - b_i)^2$$
- Distancia Manhattan
$$||a - b||_1 = \sum_i |a_i - b_i|$$
- Distancia de Chebyshev, también conocida como distancia máxima
$$||a - b||_\infty = \max_i |a_i - b_i|$$

- Distancia de Mahalanobis

$$d(a,b) = \sqrt{(a-b)^T S^{-1}(a-b)}$$

donde S es la matriz de covariancia.

Los criterios de enlace sirven para determinar la distancia entre dos *clusters*. A continuación se describen dos criterios que se utilizan frecuentemente.

Sean A y B dos *clusters*, tenemos que:

- Un **enlace simple** se define como la mínima distancia entre los elementos de cada *clúster*.

$$\text{mín}\{d(x,y) : x \in A, y \in B\}$$

- Un **enlace promedio** se define como la distancia promedio entre los elementos de cada *clúster*.

$$\frac{1}{|A| \cdot |B|} \sum_{x \in A} \sum_{y \in B} d(x,y)$$

Donde $|C|$ es la cardinalidad del conjunto C y $\cdot$ es la multiplicación aritmética.

Agrupamientos divisivos

Los agrupamientos divisivos, como su nombre lo indica, hacen divisiones de las observaciones para que se queden en grupos de similitud. El objetivo es minimizar la varianza intra-clúster:

$$\arg\min_S \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2 = \arg\min_S \sum_{i=1} |S_i| Var(S_i)$$

donde S_i es el conjunto de puntos del *clúster* i y μ_i es la media de los puntos en S_i. Dado que la varianza es constante la siguiente igualdad se cumple ya que es equivalente a maximizar la suma de las desviaciones al cuadrado de los puntos en los diferentes grupos (Kriegel, Schubert, y Zimerk, 2017):

$$|S_i| \sum_{x \in S_i} ||x - \mu_i||^2 = \sum_{x \neq y \in S_i} ||x - y||^2$$

Este problema es equivalente a minimizar la desviación por pareja de puntos en el mismo *clúster*:

$$\arg\min_S \sum_{i=1}^{k} \frac{1}{|S_i|} \sum_{x,y \in S_i} ||x - y||^2$$

A continuación se explican dos métodos divisivos.

- k-**means**. Este método divide las observaciones en k *clusters*. Para ello define k centroides aleatorios y, posteriormente, envía cada observación a su centroide más cercano. Una vez formados los k grupos, recalcula los centroides calculando la media de los puntos del grupo y repite nuevamente el proceso de agrupación, hasta que los grupos están definidos y dejan de cambiar. Es importante notar que los centroides no son necesariamente parte de las observaciones.

- k-**medoids**. Este método es parecido a k-means pero en lugar de definir centroides define *medoids*. Un *medoid* es el objeto del grupo cuya disimilaridad media a todos los puntos del grupo es mínima. Además a diferencia de k-means este método ha sido utilizado con medidas de distancia diferentes a la euclidiana.

Clustering con metaheurísticas

Los métodos de *clustering* clásicos tienen varias desventajas, por ejemplo, pueden quedar atrapados en óptimos locales y muchas veces requieren saber a priori el número de grupos en los que se dividirán los datos. Para hacer frente a estas desventajas, se han propuesto algunas técnicas basadas en metaheurísticas como recocido simulado (Bandyopadhyay, Maulik, y Pakhira, 2001; Brown y Huntley, 1992; Klein y Dubes, 1989; Lee y Perkins, 2021; Selim y Alsultan, 1991) y algoritmos evolutivos (Bandyopadhyay y Maulik, 2001; Bandyopadhyay y Maulik, 2002; Das, Abraham, y Konar, 2008; Johnson y Sahin, 2009; José-García y Gómez-Flores, 2016; Sheng y X., 2006; van der Merwe y Engelbrecht, 2003; Wang, Wang, Zhu, Wang, y Wang, 2021).

3.2. *Clustering* multi-objetivo

Por otro lado, algunas aplicaciones requieren capturar diferentes características en los objetos, lo cual lleva a optimizar varias funciones objetivo de manera simultánea. En estos casos, se puede definir el problema de *clustering* como un POM de la siguiente forma: Encontrar la k-partición C^* para la cual se cumple:

$$F(C^*) = \min_{C \in \Omega} F(C) = (f_1(C), \ldots, f_M(C))^T \tag{3.2}$$

donde Ω es el conjunto de las k-particiones factibles, $C = \{C_1, \ldots, C_k\}$ es una k-partición de un conjunto finito de datos S, $C_1, \ldots C_k$ son conjuntos disjuntos y $F : \Omega \to \mathbb{R}^n$ es un vector de funciones objetivo donde $f_i : \Omega \to \mathbb{R}$.

Actualmente, existen varias propuestas para resolver el problema de *clustering* multi-objetivo, algunos ejemplos se pueden encontrar en: Bandyopadhyay (2011); Bandyopadhyay y Saha (2013); Handl y Knowles (2006); Handl y Knowles (2007); Heloulou, Radjef, y Kechadi (2017); Law, Topchy, y Jain (2004).

3.2.1. El problema de logística de entregas

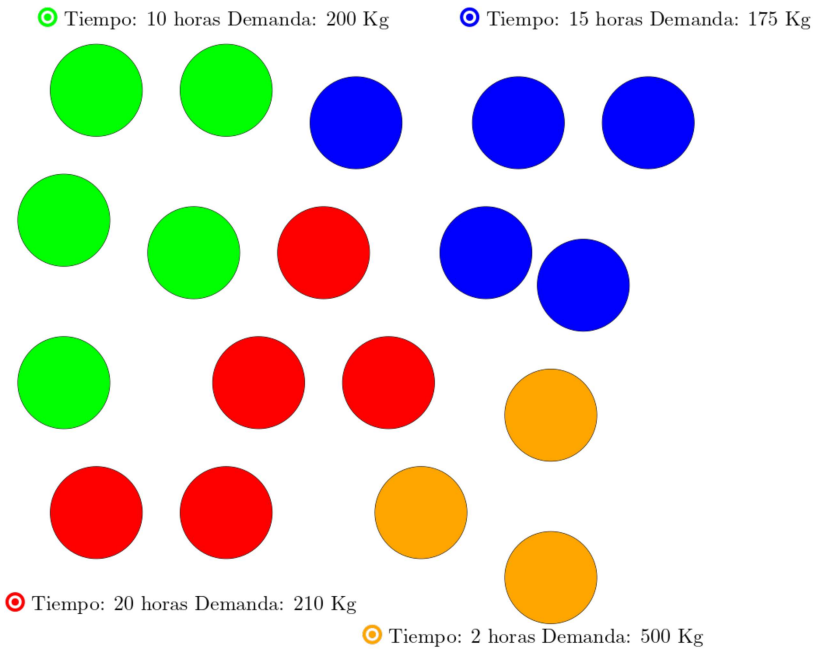
Desde hace varios años las empresas están interesadas en el diseño de rutas que les permita entregar sus productos al menor costo posible. Regularmente, una ruta parte de un punto (almacén) y debe visitar n puntos de venta para posteriormente regresar al punto de partida. En muchos casos, las empresas deben considerar miles e incluso millones de puntos de venta. Este problema es famoso en computación y se conoce como el problema del viajero (TSP por sus siglas en inglés). TSP es clasificado como un problema NP-difícil (Johnson, 1990) porque cumple con la propiedad conocida como "hardness" (cualquier problema NP puede ser reducido a él) y no existe un algoritmo que pueda verificar si una solución es correcta o no en tiempo polinomial.

En la actualidad, existen varios algoritmos basados en programación dinámica o técnicas de ramificación y poda que resuelven TSP de manera exacta. Sin embargo, su complejidad es exponencial con respecto al número de puntos que se deben visitar. Por esta razón, existen varias propuestas basadas en metaheurísticas que buscan soluciones buenas al TSP. No obstante, sus costos computacionales siguen creciendo cuando el número de puntos incrementa.

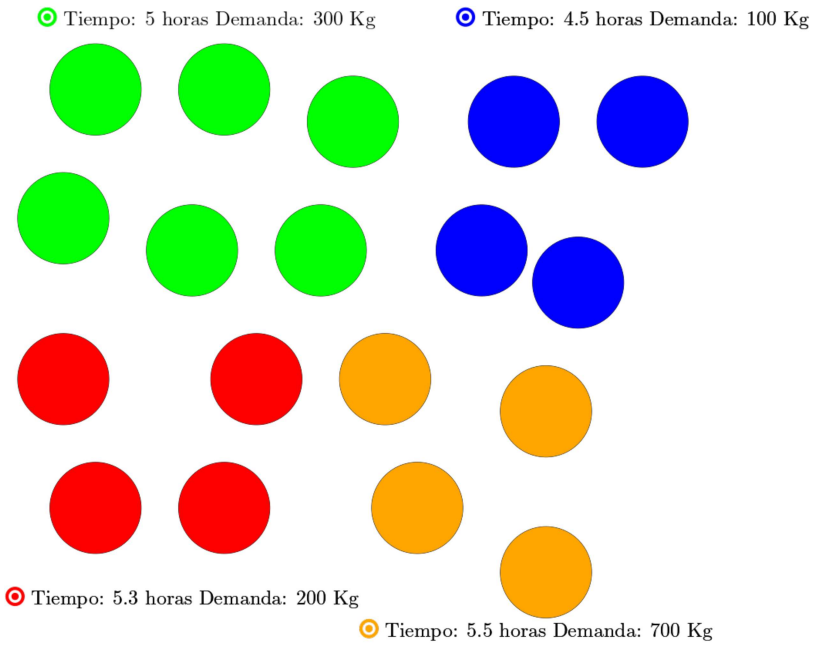
Por lo anterior, muchas empresas optan por sectorizar las grandes ciudades para posteriormente diseñar la logística de entrega en cada sector. Regularmente las empresas desean dividir las grandes ciudades en áreas bien definidas, donde todas las áreas tengan una carga de trabajo similar, ya sea en tiempo o peso. Es decir, si asignamos un repartidor en cada sector, se desea que todos los repartidores requieran un tiempo muy similar para visitar todos los puntos del sector. De igual forma, se desea que todos los repartidores puedan llevar todos los productos en un solo camión, lo que lleva a que el peso total de los productos que se deben entregar en los diferentes sectores también sea similar. Este problema puede ser definido como un problema de *clustering* multi-objetivo, que busca generar grupos donde los puntos de un mismo grupo estén más relacionados entre sí que con los puntos de otros grupos pero, además, que los grupos generados estén balanceados con respecto al costo que implica recorrer cada uno de sus puntos y entregar los productos, ver Figura 3.1.

Formalmente podemos plantear el problema como sigue. Encontrar la k -partición C^* para la cual se cumple:

$$F(C^*) = \min_{C \in \Omega} F(C) = (f_1(C), f_2(C), f_3(C)^T \tag{3.3}$$



(a) Balanceo por peso



(b) Balanceo por tiempo

Figura 3.1: Ejemplo de balanceos conflictivos. Se observa como al intentar minimizar una función objetivo la otra empeora.

donde $F : \Omega \to \mathbb{R}^3$ y f_1, f_2 y f_3 se definen como sigue:

$$f_1(C) = \sum_{i=1}^{k} \sum_{x_j \in C_i} d(x_j - \bar{x}_i)^2$$

$$f_2(C) = \frac{1}{k} \sum_{i=1}^{k} (t_i - \bar{t})^2$$

$$f_3(C) = \frac{1}{k} \sum_{i=1}^{k} (w_i - \bar{w})^2$$

donde x_i es un punto de venta; $\bar{x}_i$ es el centroide del grupo C_i; $d(\cdot)$ es la distancia entre dos puntos; t_i es el costo de visitar todos los puntos en C_i, $\bar{t} = \frac{1}{k} \sum_{i=1}^{k} t_i$, w_i es el peso total de los productos que se van a entregar en el grupo C_i y $\bar{w} = \frac{1}{k} \sum_{i=1}^{k} w_i$.

Minimizar f_1 permite que se obtengan áreas geográficas bien definidas y busca que cada punto pertenezca al grupo que tiene asociado el centroide más cercano. Minimizar f_2 implica obtener grupos que tengan tiempos de entrega similares. Finalmente, minimizar f_3 implica obtener grupos que tengan cargas similares.

Menchaca-Méndez et al. (2022) proponen una metaheurística basada en Recocido Simulado que permite obtener áreas geográficas bien definidas, con tiempos de entrega similares en los grupos que genera. Sin embargo, esta propuesta resuelve un problema de optimización mono-objetivo porque no considera el balanceo con respecto al peso total de los productos en cada grupo. Por lo anterior, en este trabajo de tesis abordaremos el problema multi-objetivo definido en la Ec. 3.3.

Capítulo 4

AMOSA

"Archive Multi Objective Simulated Annealing (AMOSA)" (Bandyopadhyay, Saha, Maulik, y Deb, 2008) es un algoritmo de optimización multi-objetivo basado en recocido simulado que ha sido ampliamente utilizado debido a su flexibilidad y capacidad de salir de óptimos locales. Se decidió utilizar este algoritmo para resolver el problema de agrupamiento balanceado aplicado a la logística de entregas porque se tiene la hipótesis de que los algoritmos evolutivos no son una buena opción dado que requieren evaluar las funciones objetivos varias veces en cada iteración. En las siguientes secciones describiremos el funcionamiento de AMOSA.

4.1. Recocido Simulado

Recocido simulado (RS) es un método probabilístico utilizado para buscar el óptimo global de una función que puede poseer varios óptimos locales. Está basado en el proceso físico de un sólido que se enfría lentamente hasta alcanzar una configuración de energía mínima. Los principales componentes del RS son:

1. Generación de una solución inicial
2. Función de variación de la temperatura
3. Definición de un vecindario

Suponiendo que queremos minimizar una función $f(x)$ y que x es la solución actual, RS va a definir un vecindario para x, denotado como $N(x)$. Posteriormente, se elige una solución y tal que $y \in N(x)$. Una vez obtenida esta solución y, la probabilidad de que y sea la nueva solución actual es 1 si $f(y) \leq f(x)$ y $e^{\frac{-(f(y)-f(x))}{T(t)}}$ de lo contrario, donde $T(t)$ es la temperatura en el tiempo t. El Algoritmo 1 muestra el procedimiento completo, el cual recibe como parámetros de entrada la temperatura inicial T_0, la temperatura final T_f y el factor de enfriamiento $T(t)$.

Algoritmo 1 Recocido Simulado(T_0, T_f, $T(t)$)

$T(0) \leftarrow T_0$
$t \leftarrow 0$
$x \leftarrow$ Generar solución inicial
while $T(t) \geq T_f$ **do**
    Elegir aleatoriamente $y \in N(x)$
    if $f(y) \leq f(x)$ **then**
        $x \leftarrow y$
    else
        if Random$(0, 1) < \exp \frac{-(f(y)-f(x))}{T(t)}$ **then**
            $x \leftarrow y$
    $t \leftarrow t + 1$
    Obtener $T(t + 1)$

4.2. Algoritmo de AMOSA

A diferencia de un RS enfocado en resolver problemas de optimización mono-objetivo, que devuelve una única solución, AMOSA devuelve un archivo de tamaño finito que almacena las soluciones no dominadas encontradas durante el proceso de búsqueda. Dicho archivo empieza vacío y se va actualizando de acuerdo a cuatro procedimientos principales:

1. Inicialización del archivo
2. *Clustering* de las soluciones no dominantes en el archivo
3. Nivel de dominancia entre soluciones
4. Proceso principal de AMOSA

4.2.1. Inicialización del archivo

Se crea un archivo el cual se va a rellenar a partir de soluciones iniciales, generadas aleatoriamente, y refinamientos de las mismas a través de una técnica de búsqueda local llamada *escalando la colina* (la cual se describe en la siguiente sección). Si las soluciones creadas no son dominadas por alguna solución del archivo, entonces se agregan al archivo. Si dichas soluciones dominan a otras soluciones del archivo, estas últimas son eliminadas.

Escalando la colina

El Algoritmo 2 describe la técnica de escalando la colina (Russell, Norvig, y Canny, 2003) la cual recibe como entrada una solución x y un número de itera-

ciones num_iter. La técnica consiste en hacer pequeños cambios a la solución x para tratar de mejorarla.

Algoritmo 2 Escalando la colina(x, num_iter)

```
1: i ← 0
2: while i < num_iter do
3:     y ← x
4:     k ← Elegir aleatoriamente una variable de decisión
5:     d ← Elegir aleatoriamente la dirección de búsqueda (1 ó -1)
6:     f ← Obtener tamaño de paso (x[k], límite inferior de x[k], límite superior de
       x[k]), d)
7:     Se muta la nueva solución: y[k] ← x[k] + d · f
8:     if y domina a x then
9:         x ← y
10:    i ← i + 1
   return x
```

Algoritmo 3 Obtener tamaño de paso(x, límite inferior, límite superior, dirección)

```
1: if ( límite inferior = x and dirección = -1) or (límite superior = x and dirección =
   1) then
2:     return 0
3: else
4:     if dirección = 1 then
5:         return número aleatorio entre[0, límite superior − x)
6:     else
7:         return número aleatorio entre[0, x − límite inferior)
```

4.2.2. *Clustering* de las soluciones en el archivo

El *clustering* en el archivo se realiza para poder garantizar variedad en las soluciones que se tienen y se aplica cuando el tamaño del archivo llega a un límite llamado *Soft Limit* (SL). El objetivo es reducir el archivo para que tenga un tamaño igual al límite llamado *Hard Limit* (HL) tal que $SL > HL$. Tener estos límites, permite reducir la cantidad de veces que se realiza el proceso de *clustering* y, a su vez, garantiza variedad en el espacio de soluciones.

AMOSA utiliza agrupamiento de enlace simple (Jain y Dubes, 1988). Esta es una técnica de agrupamiento jerárquico que empieza con un elemento por grupo, posteriormente, combina los grupos que sean más cercanos entre sí hasta formar un solo grupo, comos e puede observar en la Figura 4.1. Al final, AMOSA se queda con las HL soluciones más representativas. La solución representativa de cada grupo es aquella cuya distancia promedio hacia los otros miembros del grupo sea la mínima.

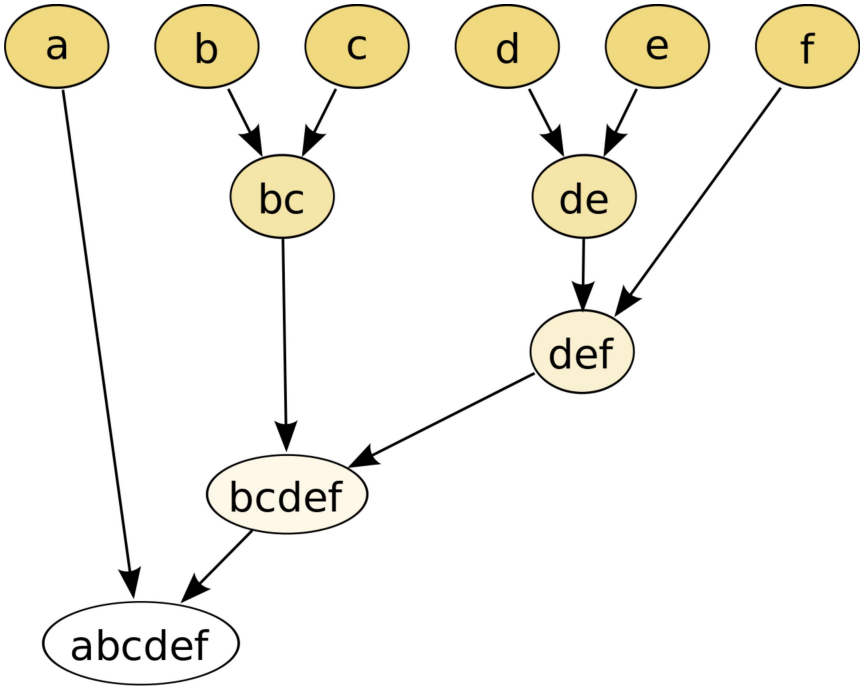


Figura 4.1: Agrupamiento de enlace simple. Diagrama elaborado por Stathis Sideris (2005).

4.2.3. Nivel de dominancia entre soluciones

AMOSA determina la cantidad de dominancia entre dos soluciones, a y b, de la siguiente manera:

$$\Delta\text{dom}_{a,b} = \prod_{i=1, f_i(a) \neq f_i(b)}^{M} \frac{|f_i(a) - f_i(b)|}{R_i}$$

donde M es el número de objetivos y R_i es el rango del i-ésimo objetivo. Los rangos de los objetivos puede que no se conozcan a priori, por lo que el optimizador puede calcularlos basándose en las soluciones que se tienen actualmente en el archivo junto con la solución actual y la nueva solución, tomando la diferencia del valor máximo de cada una de las funciones objetivo con el valor mínimo de estas:

$$R_i = \text{máx}\, F_i - \text{mín}\, F_i$$

donde $F = \{F_1, F_2, \ldots F_M\}$ es el conjunto de todas las evaluaciones de la i-ésima función objetivo de las soluciones del archivo, junto con la solución actual y la nueva solución.

4.2.4. Proceso principal de AMOSA

Para empezar el proceso de optimización principal, primero se debe inicializar el archivo. Luego, se selecciona aleatoriamente una solución *Sol* del archivo y se genera

una nueva solución $NuevaSol$, perturbando la solución Sol. Posteriormente, se revisa la cantidad de dominancia entre $NuevaSol$, y las soluciones del archivo, lo cual puede generar los siguientes casos:

- Caso 1: Sol domina a $NuevaSol$ y k ($k \geq 0$) soluciones del archivo dominan a $NuevaSol$, ver Figura 4.2. En este caso, la probabilidad de que $NuevaSol$ se vuelva Sol es:

$$prob = \frac{1}{1 + exp(\Delta \text{dom}_{\text{avg}}) * temp} \tag{4.1}$$

  Donde

$$\Delta \text{dom}_{\text{avg}} = \frac{(\sum_{i=1}^{k} \Delta \text{dom}_{i,NuevaSol}) + (\Delta \text{dom}_{Sol,NuevaSol})}{(k+1)} \tag{4.2}$$

  y $temp$ es la temperatura en ese momento.

- Caso 2: Sol y $NuevaSol$ son no dominantes entre si, por lo que se tiene que revisar la dominancia de $NuevaSol$ con respecto a los puntos del archivo.

  1. $NuevaSol$ es dominada por k puntos, con $k \geq 1$, del archivo. Ver Figura 4.3a. Elegimos $NuevaSol$ como la nueva solución con la probabilidad descrita en la Eq. (4.1) pero calculando $\Delta \text{dom}_{\text{avg}}$ como sigue:

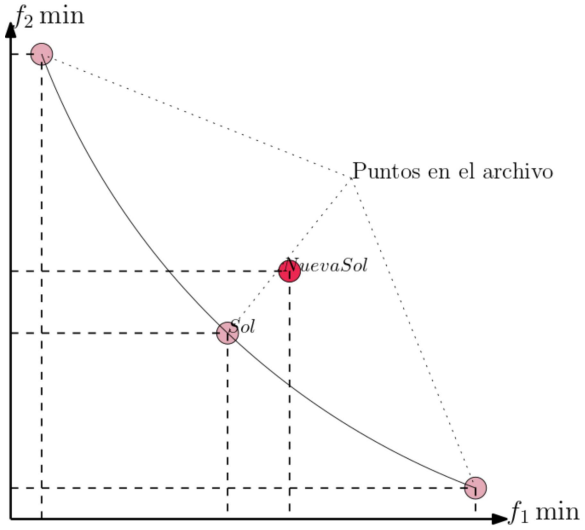
$$\Delta \text{dom}_{\text{avg}} = \frac{\sum_{i=1}^{k} (\Delta \text{dom}_{i,NuevaSol})}{k} \tag{4.3}$$

  2. $NuevaSol$ es no dominante con respecto de los otros puntos del Archivo. Ver Figura 4.3b. En este caso, $NuevaSol$ está en el mismo frente que el resto del archivo por lo que solamente se añade al archivo. En caso de que el tamaño del archivo sea mayor a SL, se ejecuta el clustering.
  3. $NuevaSol$ domina a k puntos del archivo, con $k \geq 1$. Ver Figura 4.3c. $NuevaSol$ pasa a ser Sol y se añade al archivo mientras que, los k puntos dominados, son removidos del archivo:

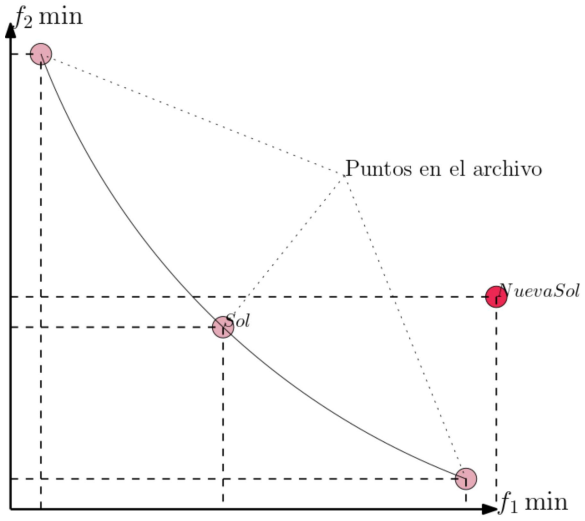
- Caso 3: $NuevaSol$ domina a Sol. En este caso, nuevamente se tiene que revisar la dominancia de $NuevaSol$ con respecto a los puntos en el archivo:

  1. $NuevaSol$ domina a Sol, pero es dominada por k ($k \geq 1$) puntos en el archivo. Ver Figura 4.4a. Se calcula la mínima diferencia de dominancia entre $NuevaSol$ y los k puntos del archivo $\Delta \text{dom}_{\text{mín}}$. El punto del archivo que corresponda a la mínima diferencia es seleccionado como Sol, con la siguiente probabilidad:

$$prob = \frac{1}{(1 + exp(-\Delta \text{dom}_{\text{mín}}))} \tag{4.4}$$

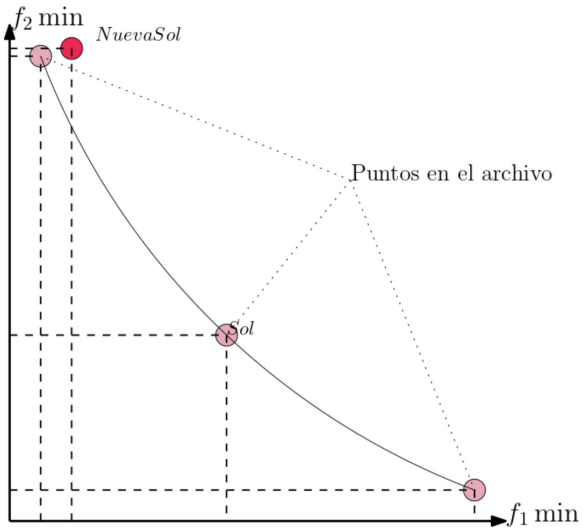


(a) $NuevaSol$ es dominada por $k = 1$ soluciones del archivo

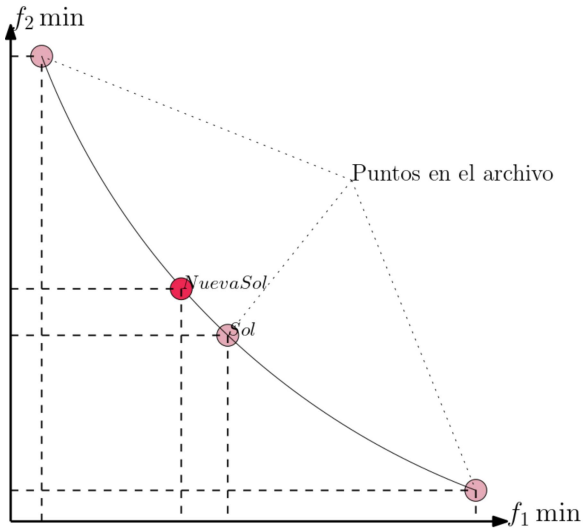


(b) $NuevaSol$ es dominada por $k = 2$ soluciones del archivo

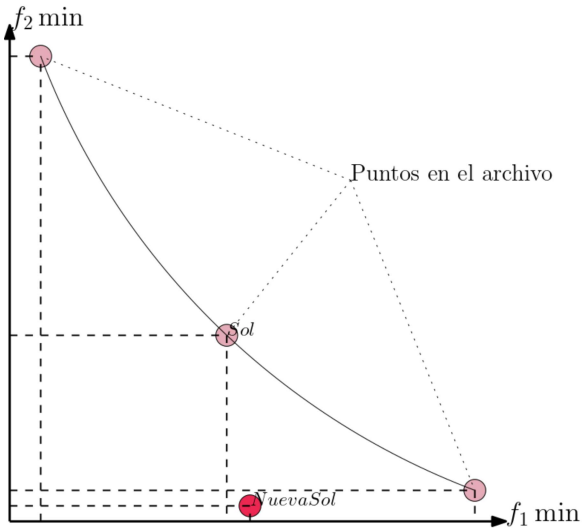
Figura 4.2: Primer caso de AMOSA: Sol domina a $NuevaSol$



(a) $NuevaSol$ es dominada por $k \geq 1$ puntos del archivo



(b) $NuevaSol$ es no dominante con respecto a los demás puntos del archivo



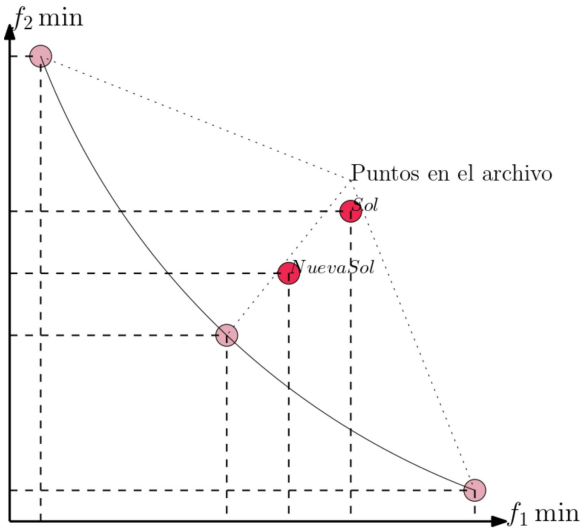
(c) $NuevaSol$ domina a $k \geq 1$ soluciones del archivo

Figura 4.3: Segundo caso de AMOSA: Sol y $NuevaSol$ son no dominantes entre sí

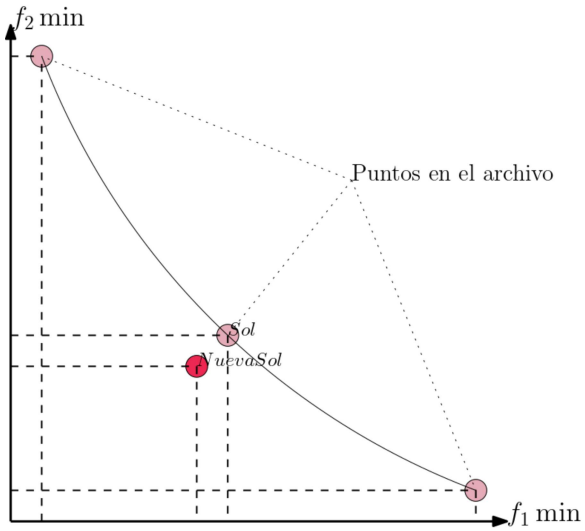
de lo contrario, $NuevaSol$ pasa a ser Sol.

2. $NuevaSol$ domina a Sol y es no dominada con el resto de los puntos en el archivo. Ver Figura 4.4b. En este caso, $NuevaSol$ pasa a ser Sol y es añadida al archivo. Si al hacer esta operación, la longitud del archivo es mayor al SL, se hace el *clustering* de las soluciones.
3. $NuevaSol$ domina k ($k \geq 1$) otros puntos del archivo. Ver Figura 4.4c. Por lo tanto, $NuevaSol$ pasa a ser Sol y se añade al archivo, mientras que todos los puntos dominados del archivo, son removidos.

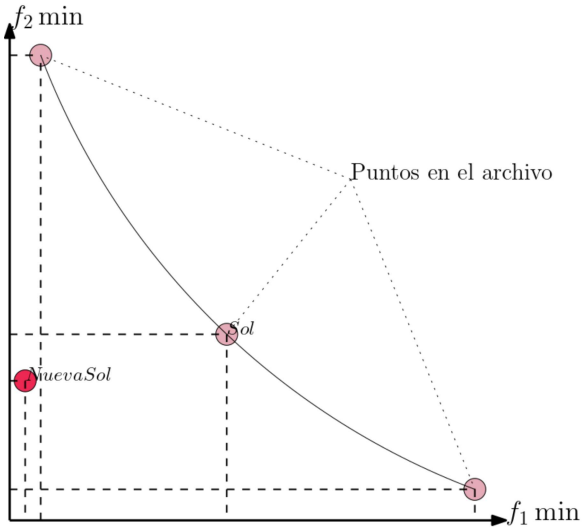
El Algoritmo 4 muestra el procedimiento completo de AMOSA.



(a) $NuevaSol$ es dominada por $k \geq 1$ soluciones del Archivo



(b) $NuevaSol$ es no dominada con los demás puntos del archivo



(c) $NuevaSol$ domina a $k \geq 1$ puntos del archivo

Figura 4.4: Tercer caso de AMOSA: $NuevaSol$ domina a Sol

Algoritmo 4 Algoritmo de AMOSA

```
1: Tmax, Tmin. HL, SL, iter, cool, temp=Tmax        ▷ cool es el factor de enfriamiento
2: Iniciar el archivo A.
3: Sol ← random(Archive)                            ▷ elegir una solución aleatoria del archivo
4: while temp > Tmin do
5:     for i = 0; i < iter; i = i + 1 do
6:         NuevaSol ← Escalando la colina(Sol, 1)
7:         if Sol domina a NuevaSol then
8:             Se calcula ΔDom_avg con la Ecuación 4.2
9:             Calcular prob con la Ecuación 4.1
10:            Sol ← NuevaSol con probabilidad prob
11:        if Sol y NuevaSol son no dominantes entre ellas then
12:            Revisar la dominancia entre NuevaSol y los puntos de A
13:            if NuevaSol es dominada por k puntos then
14:                Se calcula ΔDom_avg con la Ecuación 4.3
15:                Se calcula prob con la Ecuación 4.1
16:                Sol ← NuevaSol con probabilidad prob
17:            if NuevaSol es no domintante con el resto de los puntos de A then
18:                Sol ← NuevaSol
19:                Añadir NuevaSol a A
20:                if Longitud del archivo > SL then
21:                    Hacer Clustering
22:            if NuevaSol domina a k puntos de A then
23:                Sol ← NuevaSol
24:                Añadir NuevaSol a A
25:                Quitar los k puntos dominados de A
26:        if NuevaSol domina a Sol then
27:            Revisar dominancia entre NuevaSol y los puntos en A
28:            if NuevaSol es dominado por k puntos de A then
29:                Δdom_mín ← mínimo de las diferencias de dominancia entre NuevaSol
   y los k puntos
30:                Calcular prob con la Ecuación 4.4
31:                Sol ← punto que corresponde al Δdom_mín con probabilidad prob de lo
   contrario Sol ← NuevaSol
32:            if NuevaSol es no dominante con los puntos de A then
33:                Sol ← NuevaSol
34:                Añadir NuevaSol a A
35:                if Longitud de A > SL then
36:                    Hacer clustering
37:            if NuevaSol domina k puntos del archivo then
38:                Sol ← NuevaSol
39:                Añadir NuevaSol a A
40:                Quitar los k puntos dominados de A
41:    temp ← cool ∗ temp
42: if Longitud de A > SL then
43:    Hacer clustering
```

Capítulo 5

Propuesta

5.1. Representación de la solución candidata

Para resolver el problema planteado en este trabajo, ver Ecuación (3.2.1), se propone el uso de un matriz de valores reales, donde cada fila corresponde con un centroide, la primera columna con la latitud y la segunda con la longitud.

$$\bar{x} = \begin{pmatrix} lat_1, & lon_1 \\ lat_2, & lon_2 \\ & \vdots \\ lat_k, & lon_k \end{pmatrix}$$

Los grupos se definen de la siguiente forma: cada punto del conjunto de datos se asigna al grupo que corresponde al centroide más cercano. De esta forma se considera la función objetivo f_1 y se obtienen grupos que generan áreas geográficas bien definidas[1], dejando que el optimizador se encargue únicamente de las funciones f_2 y f_3.

5.2. Uso de AMOSA y sus modificaciones

AMOSA no está diseñado para problemas de clustering balanceado, ni clustering en general, por lo que se hicieron algunas adaptaciones.

5.2.1. Funciones objetivo

La primera adaptación fue la incorporación de las funciones objetivo. Dado que evaluar las funciones implica tener los agrupamientos ya definidos, se estableció una

[1]Se considera que las áreas geográficas están bien definidas cuando no existe traslape entre ellas.

matriz que guarda la distancia de cada punto en el conjunto de datos a los centroides. Dicha matriz se calcula con la solución inicial y, de ahí, se procede a realizar cambios solamente en las columnas de los centroides que se modifican, esto debido a que la solución se perturba modificando un solo centroide en una de sus coordenadas. De esta forma, se reduce el número de operaciones que se deben realizar en cada agrupamiento.

Una vez que los grupos están formados, la función objetivo f_2 implica establecer el orden en el que los puntos de venta serán visitados. Como obtener un orden óptimo es muy costoso (Problema *TSP*), se decidió visitar los puntos de manera ordenada considerando una de las coordenadas geográficas. Para esto se ordenan todos los puntos del conjunto de datos, antes de iniciar el proceso de búsqueda. Posteriormente, cuando se forman los grupos, se recorren los puntos ya ordenados y se van a asignando a los grupos, por lo que los puntos en los grupos también estarán ordenados.

5.2.2. Vecindario

La segunda adaptación tiene que ver con la generación de la solución vecina. En nuestro caso, la perturbación del vector solución se realiza tomando una de las coordenadas de un centroide al azar, y sumándole o restándole un número aleatorio en el rango de la diferencia del punto con el límite superior o con el límite inferior de la variable, según sea la dirección elegida. Esto es parecido a lo presentado en el Algoritmo 2, pero ahora multiplicado por un escalar α para ajustar el tamaño de la búsqueda. α permite controlar el tamaño del vecindario y puede tomar un valor entre 0 y 1. Un valor cercano a cero genera cambios pequeños que permiten realizar una búsqueda fina en la zona (explotación), mientras que un valor cercano a uno permite explorar la zona, pues realiza cambios grandes en la solución. El Algoritmo 5 muestra el procedimiento completo.

5.2.3. Paralelización

Varios de los cálculos que se deben de realizar son computacionalmente costosos, esto se debe principalmente a la gran cantidad de puntos que se tienen en los conjuntos de datos de entrada. Al momento de inicializar el archivo en el algoritmo de AMOSA, la matriz de distancia debe recalcularse por cada solución generada. Dado que esta tarea es pesada, se decidió paralelizarla de la siguiente forma:

- El número de centroides es menor al número de núcleos de cómputo disponibles (cores). En este caso cada centroide es asignado a un núcleo y se hacen los cálculos de las distancias de cada punto al centroide correspondiente.

Algoritmo 5 Perturbación de la solución(x, α)

```
1: P ← 0
2: while P = 0 do
3:     k ← Elegir aleatoriamente una variable de decisión
4:     d ← Elegir aleatoriamente la dirección de búsqueda (1 ó -1)
5:     Límite inferior ← (Límite inferior de x[k]) - x[k]
6:     Límite superior ← (Límite superior de x[k]) - x[k]
7:     if d = 1 then
8:         P ← random(0, Límite superior]
9:     else
10:        P ← random[Límite inferior, 0)
11: x[k] ← x[k] + α* P
12: return x
```

- El número de centroides es mayor que el número de núcleos disponibles. En este caso, la matriz de distancia es dividida en rangos y cada rango se asocia a un núcleo. Por lo tanto, cada núcleo calcula las distancias de los centroides a los puntos de su sección.

Una vez que se tiene la matriz de distancia y se modifica una solución, se identifica el centroide que fue perturbado, para posteriormente, calcular solamente la distancia de los puntos a este nuevo centroide. Esto último también se realiza de forma paralela, para ello se divide la matriz en rangos, cada rango se asigna a un núcleo y cada núcleo calcula las distancias de los puntos del rango asociado.

Capítulo 6

Experimentos y Resultados

6.1. Conjuntos de Datos

Para realizar la evaluación de nuestra propuesta, se utilizaron datos reales de dos Estados de la República Mexicana, Hidalgo y Morelos, ver Figura 6.1. Cada uno de estos conjuntos de datos contiene coordenadas geográficas de puntos de venta reportados por el Instituto Nacional de Estadística, Geografía e Informática (INEGI) y fueron obtenidos a través del Directorio Estadístico Nacional de Unidades Económicas (DENUE) 05_2021 [1]. Estos conjuntos contienen 47687 y 38701 puntos, respectivamente, después de ser limpiados y procesados.

6.2. Rendimiento de la versión paralelizada

6.2.1. Descripción de las herramientas utilizadas

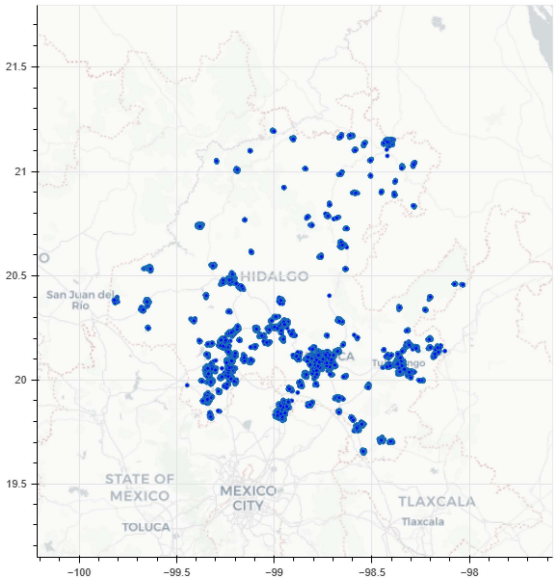
Los experimentos de paralelización se realizaron en una computadora con un CPU AMD Ryzen 5 1600 @ 3.20 GHz de 6 núcleos, con 16 GB de memoria RAM corriendo la distribución de GNU/Linux Ubuntu 22.04, usando el lenguaje de programación *Python* versión 3.9, con las librerías *Pandas 1.5, NumPy 1.23* y *Cython 0.29*.

Paralelización con *threading*

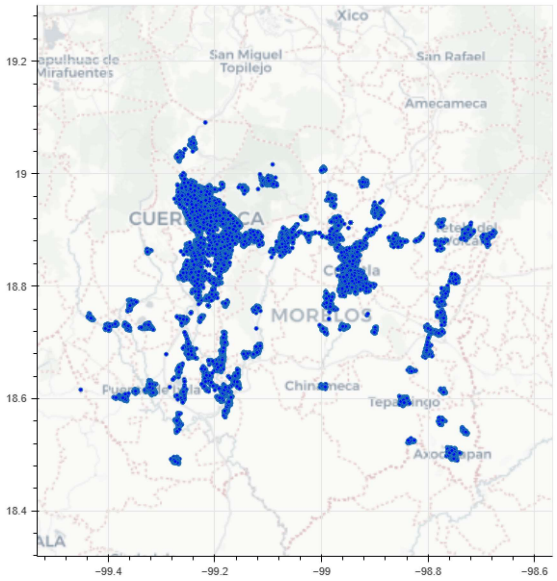
Threading es un módulo de *Python*, que permite la interacción con el módulo de bajo nivel llamado *_thread*[2], el cual provee las funciones y primitivas necesarias para poder interactuar con múltiples procesos manteniendo el control de las variables y

[1] `https://www.inegi.org.mx/app/descarga/`

[2] `https://docs.python.org/3/library/_thread.html`



(a) Puntos de venta en el Estado de Hidalgo.



(b) Puntos de venta en el Estado de Morelos.

Figura 6.1: Visualización de los puntos de venta.

datos contenidos en el espacio global, lo cual se hace por medio de candados, comúnmente llamados *mutex* o semáforos binarios. Este módulo es parte de la librería estándar de *Python*[3] desde la versión 3.7 del lenguaje, por lo que no es necesario realizar ninguna instalación adicional.

Perfilación de tiempo de las funciones

Para poder evaluar el rendimiento de la paralelización de las funciones objetivo, se utilizaron dos herramientas incluidas en la librería estándar de *Python*: [4]

- *cProfile*. Es una extensión de *Python* escrita en el lenguaje de programación *C*, la cual permite la medición del tiempo de ejecución por llamada de cada función, así como el tiempo total, tanto del programa como de cada llamada.
- *pstats*. Es un módulo de *Python* creado para tomar los valores arrojados por *cProfile* y ponerlos en el formato que se le especifique. Por ejemplo, podemos especificar qué funciones son de nuestro interés, si queremos que estén ordenadas en orden ascendente o descendente, el formato del archivo de salida, etc.

La Tabla 6.1 muestra los tiempos acumulativos por función, tanto de la versión del algoritmo que no está paralelizada como de la versión que tiene la paralelización propuesta. La implementación de AMOSA que se utilizó en este trabajo, utiliza un parámetro γ para modificar el tamaño del archivo de la siguente manera: $\gamma \times (SL + HL)$. En esta tabla podemos observar que nuestra propuesta logra una mejora de tiempo del 13 %. Es importante mencionar que la mayoría de las funciones para la versión paralelizada, fueron re-implementadas en *cython* con la finalidad de que se convirtieran en funciones de *C* y pudieran ser compiladas e inicializadas. Esto último permite salirse del candado global del intérprete (GIL por sus siglas en inglés) de *Python* y lograr la mejora de rendimiento en tiempo. Salirse del GIL es algo que otras librerías de *Python*, como *NumPy*, hacen para mejorar su rendimiento.

Uso del módulo *pyximport*

El módulo de *pyximport* ayuda a "compilar" las partes de *Python* que tienen interacciones con lenguajes de mayor velocidad, como lo son *C* y *C++*. La Tabla 6.2 muestra una comparación de tiempo entre la versión solamente paralelizada y la que utiliza este módulo. Esta comparación permite observar una mejora de solamente el 3 % en cuestión de tiempo de ejecución.

[3] `https://docs.python.org/3/library/threading.html`
[4] `https://docs.python.org/3/library/profile.html`

Tabla 6.1: Comparación de tiempo (acumulativo) por llamada de función usando los parámetros $HL = 75, SL = 100, \gamma = 2, climb = 2500, iter = 2500, T_{\text{máx}} = 500, T_{\text{mín}} = 1 \times 10^{-7}, cool = 0.9$. La función de distancia fue removida porque fue incorporada en eval_std_distance como una función de *Cython*.

Función	Tiempo paralelizado (segs.)	Tiempo no paralelizado (segs.)
Minimize	76460	87462.72
evaluate	76100	87230.12
eval_std_distance	14120	18584.07
eval_std_weight	321.2	187.71
distance	N/A	65656.37
get_objectives	76340	87368.96
__initialize_archive	21650	36360.18
hill_climbing	21560	36214.03

Tabla 6.2: Comparación de tiempo (acumulativo) usando el módulo *pyximport* con los parámetros $HL = 5, SL = 10, \gamma = 1, climb = 25, iter = 250, T_{\text{máx}} = 50, T_{\text{mín}} = 1, cool = 0.9$.

Función	Tiempo paralelizado (segs.)	Tiempo paralelizado y "compilado" (segs.)
Minimize	4065	3946
evaluate	4044	3928
eval_std_distance	1008	974.9
eval_std_weight	28.6	26.52
random_perturbation	3954	3838
get_objectives	4063	3943
__initialize_archive	109.3	105.9
hill_climbing	104.3	101.1

6.3. Indicadores de rendimiento

Es fundamental para cualquier algoritmo el poder medir su rendimiento, esto usualmente se hace midiendo la distancia al óptimo global, pero en el caso de la optimización multi-objetivo, no se puede hacer debido a que el óptimo consiste de un conjunto de soluciones (conjunto de Pareto óptimo) y en la mayoría de los casos dicho conjunto no es conocido o tiene un infinito de soluciones. Esto provoca que se empleen otro tipos de técnicas para la evaluación. En este caso, se va a emplear un indicador llamado hipervolumen.

6.3.1. Hipervolumen

Contrario a otras técnicas de evaluación, el hipervolumen, denotado como Hv, no necesita un conjunto de referencia sino un punto de referencia. Por esta razón y dado que Hv permite evaluar tanto la convergencia como la diversidad de las soluciones a lo largo del Frente de Pareto (PF), Hv es uno de los indicadores más utilizados (Guerreiro, Fonseca, y Paquete, 2021). Tomando en cuenta que $\mathcal{L}(\cdot)$ hace referencia a la medida de Lebesgue y que M es el número de funciones objetivo, Hv se define como sigue:

$$Hv(A, \vec{y}_{ref}) = \mathcal{L}\left(\bigcup_{\vec{z}\in A}\{\vec{y} \mid \vec{z} \prec \vec{y} \prec \vec{y}_{ref}\}\right) \tag{6.1}$$

donde $\vec{y}_{ref} \in \mathbb{R}^M$ es un punto de referencia que todas las soluciones en A deben de dominar (Zitzler & Thiele, 1998). Usualmente, $\vec{y}_{ref}$ se construye a partir de los peores valores encontrados para cada función objetivo. Por lo anterior, podemos decir que Hv mide el tamaño del espacio cubierto por un conjunto de aproximación A.

Normalización del hipervolumen

Dado que se quiere realizar la comparación de las distintas configuraciones de los algoritmos, es necesario tener un solo punto de referencia para las evaluaciones. Por tal razón se normalizaron todas las aproximaciones A que arrojaron los algoritmos y se utilizó como punto de referencia $\vec{y}_{ref} = [1.1, 1.1]$.

Para el proceso de normalización se utilizaron los mejores y peores valores encontrados por todos los algoritmos para cada función objetivo. Posteriormente, se llevaron todos los valores al intervalo $[0, 1]$, donde 0 corresponde con el peor valor encontrado y 1 el mejor valor encontrado.

6.4. Estudio Experimental

6.4.1. Descripción de las herramientas utilizadas

Los experimentos fueron realizados en una computadora con un CPU Intel Xeon E5-2680 v3 @ 2.50 GHz de 12 núcleos y 24 hilos, con 64 GB de memoria RAM, corriendo la distribución GNU/Linux Ubuntu x64 16.10, usando el lenguaje de programación *Python* versión 3.9, con las librerías *Pandas 1.5, NumPy 1.23* y *Cython 0.29*.

6.4.2. Proceso de Sintonización

Definir los parámetros de un algoritmo que utiliza metaheurísticas es un reto por sí solo, el cual se ha tratado por varias décadas en el área (Montero, Riff, y Neveu, 2014). Existen dos tipos de métodos para enfrentar este problema: métodos de sintonización de parámetros y métodos de control de parámetros. Los segundos son incorporados dentro de los algoritmos, para ayudarlos a auto-ajustar sus parámetros mientras el proceso de búsqueda va avanzando. La sintonización de parámetros consiste en procesos externos cuyo objetivo es la búsqueda de la mejor combinación de estos valores, ejecutando el algoritmo múltiples veces en un proceso de aprendizaje. Para este trabajo, se decidió aplicar una sintonización de parámetros utilizando el método de búsqueda local iterada en el espacio de configuración de parámetros (ParamILS) (Hutter, Hoos, Leyton-Brown, y Stützle, 2009), el cual es un método de búsqueda local iterativo que, dada una configuración de parámetros iniciales, realiza búsquedas en el vecindario cambiando un parámetro a la vez. Para lograr esto, una discretización del espacio de parámetros es necesaria para poder definir los vecindarios reducidos. En la Tabla 6.3 se pueden observar los parámetros que se van a sintonizar, así como los valores que van a tomar y sus valores por defecto. El parámetro *climb*, determina el número de iteraciones de la estrategia de escalando la colina, $Tmax$ es la temperatura inicial del recocido simulado, *iter* define el número de iteraciones para el recocido simulado, α define el factor de enfriamiento y el parámetro de *step* define el tamaño de la perturbación del paso. Los parámetros que se mantuvieron fijos fueron $Tmin = 0$, $HL = 5$, $SL = 10$. La manera de evaluar las configuraciones de los parámetros es considerando la calidad promedio y el número de ejecuciones de cada configuración.

Terminado este proceso de sintonización, se eligieron dos configuraciones. La primera utiliza 5 iteraciones de escalando la colina, 100 como la temperatura inicial de recocido simulado, 50 iteraciones de recocido simulado, 0.99 como el factor de enfriamiento y un step de 0.1. La segunda configuración utiliza los valores de 50, 1000, 100, 0.99 y 0.1, respectivamente. Estas configuraciones fueron elegidas por las siguientes

Tabla 6.3: Proceso de sintonización

Parámetros	Valores posibles	Valor por defecto
$climb$	{5, 10, 25, 50, 100}	25
$Tmax$	{50, 100, 500, 1000}	100
$iter$	{5, 10, 25, 50, 100}	5
α	{0.8, 0.9, 0.99}	0.9
$step$	{0.1, 0.25, 0.5, 0.75, 1.0}	0.25

razones:

1. La primera no obtiene los mejores resultados pero su tiempo de ejecución es corto y
2. La segunda obtiene soluciones de mejor calidad pero requiere un tiempo de ejecución mayor.

Las Tablas 6.4 y 6.5 muestran los valores obtenidos para el indicador de hipervolumen y los tiempos de ejecución, con las dos configuraciones descritas anteriormente.

6.5. AMOSA contra NSGA-II

Para poder tener un punto de comparación del rendimiento y calidad de las soluciones, se va a realizar la comparación de la propuesta presentada en este trabajo con uno de los algoritmos de optimización multi-objetivo más popularmente utilizado para problemas con dos funciones objetivo: NSGA-II. Para esta comparación, se decidió utilizar la implementación del framework llamado *pymoo* [5], utilizando los parámetros sugeridos por los autores de NSGA-II (Deb et al., 2002). Estos parámetros son $p_c = 0.9$ y $\eta_c = 15$ para el operador de cruza binaria simulado (SBX) y $p_m = 0.9$ y $\eta_m = 20$ para el operador de mutación basado en posición (PBX). Para el tamaño de la población P y el número de generaciones G, se utilizaron dos configuraciones $(P = 5, G = 300)$ y $(P = 50, G = 300)$, estos valores se mantuvieron relativamente pequeños debido al tiempo de ejecución requerido por NSGA-II.

La Tabla 6.4 presenta los resultados para el conjunto de datos del Estado de Hidalgo, donde se puede observar que NSGA-II requiere un tiempo de ejecución mayor

[5] `https://pymoo.org/`

Tabla 6.4: Conjunto de datos de Hidalgo. Se muestran los valores estadísticos de 20 ejecuciones independientes.

Nombre	Parámetros					AMOSA		
	$climb$	$Tmax$	$iter$	α	$step$	Hv		Tiempo de ejecución (horas)
AMOSA - 1	5	100	50	0.99	0.1	mean: 0.9308 std: 0.1813 best: 1.1940 worst: 0.6832		mean: **4.11** std: **0.05** best: **3.97** worst: **4.24**
AMOSA - 2	50	1000	100	0.99	0.1	mean: **1.0846** std: **0.1170** best: **1.1830** worst: **0.7546**		mean: 12.36 std: 0.15 best: 12.11 worst: 12.81

Nombre	Parámetros						NSGA-II		
	P	p_c	η_c	p_m	η_m	G	Hv		Tiempo de ejecución (horas)
NSGA-II - 1	5	0.9	15	0.9	20	300	mean: 0.2665 std: 0.1431 best: 0.6289 worst: 0.0119		mean: 28.86 std: 13.27 best: 6.44 worst: 43.10
NSGA-II - 2	50	0.9	15	0.9	20	300	mean: 0.2663 std: 0.1411 best: 0.6240 worst: 0.0118		mean: 36.01 std: 15.22 best: 6.48 worst: 50.67

al de AMOSA, para encontrar soluciones, y obtiene un peor valor de H_v. Es importante recalcar que NSGA-II también utilizada la versión paralelizada de las funciones objetivo. Igualmente se puede observar cómo NSGA-II fue capaz solamente de encontrar dos soluciones no dominantes entre sí; mientras que AMOSA, en su primera configuración, obtiene mejor convergencia que en su segunda configuración, pero esta última obtiene una mejor distribución, ver Figura 6.2.

La Tabla 6.5 contiene los resultados obtenidos para el conjunto de datos del Estado de Morelos, y se puede observar un comportamiento similar a lo descrito en el Estado de Hidalgo. AMOSA supera a NSGA-II tanto en la calidad de las soluciones como en el tiempo de ejecución. Sin embargo, la Figura 6.3 muestra que la segunda configuración de AMOSA es cercana en convergencia a la primera configuración, y la distribución de las soluciones obtenidas es mejor que en la primera configuración.

Tabla 6.5: Conjunto de datos de Morelos. Se muestran los valores estadísticos de 20 ejecuciones independientes.

Nombre	Parámetros					AMOSA	
	$climb$	$Tmax$	$iter$	α	$step$	Hv	Tiempo de ejecución (horas)
AMOSA - 1	5	100	50	0.99	0.1	mean: 1.081 std: 0.088 best: 1.206 worst: 0.916	mean: **3.52** std: **0.03** best: **3.47** worst: **3.65**
AMOSA - 2	50	1000	100	0.99	0.1	mean: **1.167** std: **0.035** best: **1.197** worst: **1.065**	mean: 10.22 std: 0.15 best: 9.92 worst: 10.60

Nombre	Parámetros						NSGA-II	
	P	p_c	η_c	p_m	η_m	G	Hv	Tiempo de ejecución (horas)
NSGA-II - 1	5	0.9	15	0.9	20	300	mean: 0.567 std: 0.205 best: 0.994 worst: 0.243	mean: 7.38 std: 9.02 best: 4.15 worst: 46.46
NSGA-II - 2	50	0.9	15	0.9	20	300	mean: 0.567 std: 0.206 best: 0.997 worst: 0.243	mean: 5.95 std: 1.90 best: 4.29 worst: 10.09

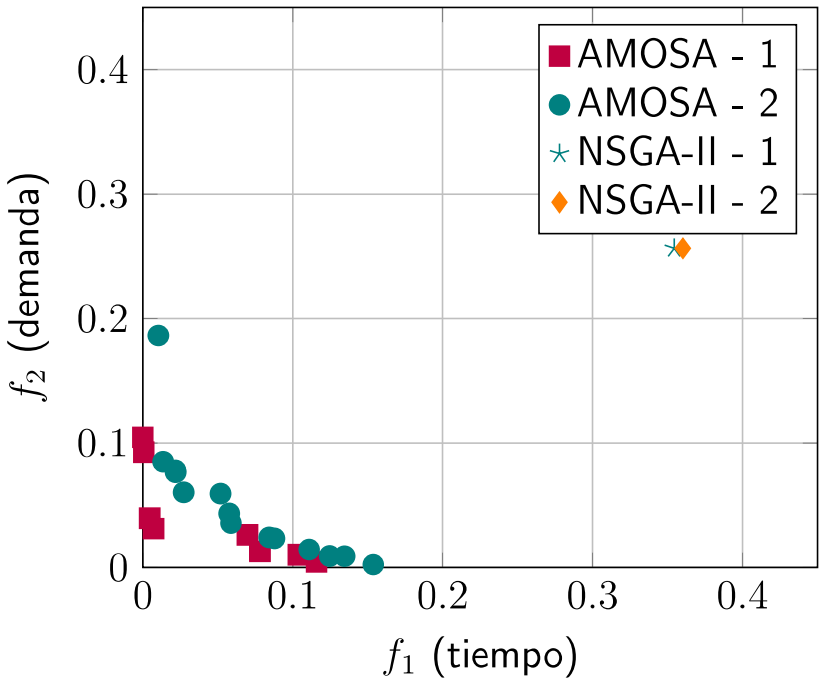


Figura 6.2: Soluciones no dominadas encontradas por todas las ejecuciones de cada algoritmo, con sus dos configuraciones, usando los datos de Hidalgo.

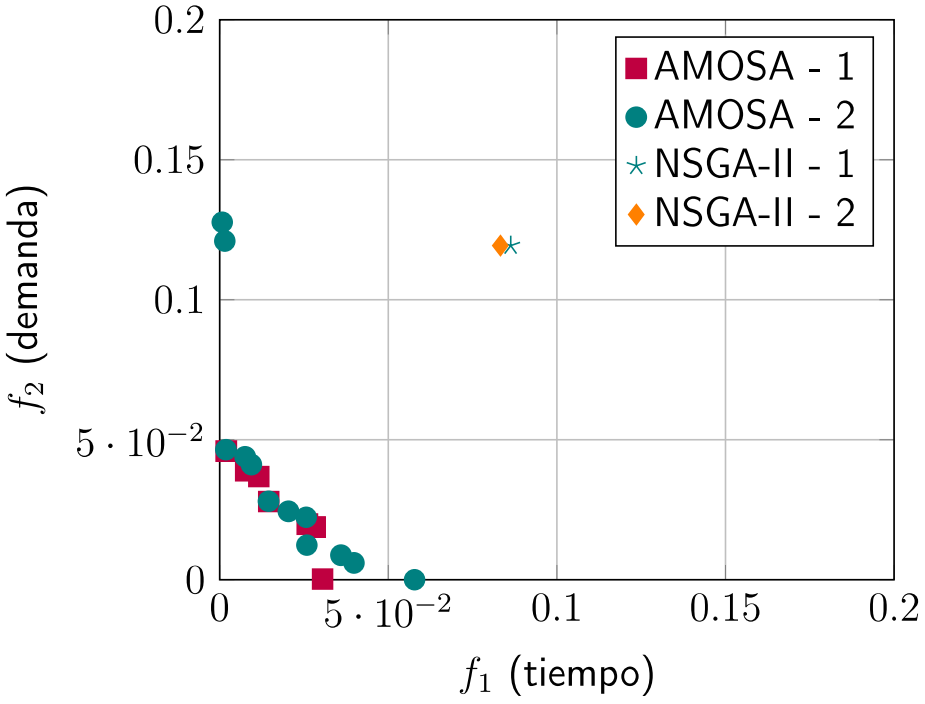


Figura 6.3: Soluciones no dominadas encontradas por todas las ejecuciones de cada algoritmo, con sus dos configuraciones, usando los datos de Morelos.

Capítulo 7

Conclusiones y trabajo futuro

En este trabajo de tesis abordamos un problema relacionado con la logística de entregas. El objetivo es crear áreas de entrega que:

1. Estén bien definidas y no sobrepuestas,
2. Sean similares en el tiempo de servicio que se requiere para recorrerlas y
3. El peso total de los productos que deben entregarse sea similar.

Dicho problema se modeló como un problema de optimización multi-objetivo y para resolverlo se propuso el uso de un algoritmo basado en recocido simulado llamado AMOSA. Pese a que los algoritmos evolutivos suelen ser más utilizados para la resolución de problemas de optimización multi-objetivo, se tuvo la hipótesis de que las metaheurísticas basadas en poblaciones no eran viables para el problema de interés, debido a que realizan múltiples evaluaciones de la función objetivo y, en este problema, el cálculo de dicha función tiene un alto costo computacional.

El estudio experimental demostró que el algoritmo propuesto superó a la configuración recomendada de NSGA-II, un algoritmo evolutivo popular para problemas multi-objetivo con dos objetivos, tanto en la calidad de las soluciones como en los tiempos de ejecución. De esta forma, validamos experimentalmente nuestra hipótesis. Sin embargo, es necesario hacer un estudio más amplio en cuanto a la búsqueda de parámetros en NSGA-II, así como incluir otros algoritmos evolutivos multi-objetivo populares, como lo son SMS-EMOA o MOEA/D.

Por otro lado, en este trabajo se realizó la paralelización a nivel de núcleos de procesamiento (cores) de la función objetivo debido a su alto costo computacional, logrando una reducción de tiempo del 13 %. Finalmente, se establecieron los parámetros de AMOSA usando ParamILS y un conjunto de valores discretos para los parámetros del algoritmo. Fue así como se encontraron dos configuraciones interesantes, la primera obtuvo tiempos de ejecución prácticos, mientras que la segunda

configuración encontró soluciones de mejor calidad, a cambio de un mayor tiempo de ejecución. Esto deja como trabajo a futuro realizar un estudio más profundo sobre la discretización de los valores de los parámetros, con el objetivo de encontrar configuraciones que puedan mejorar los resultados encontrados, tomando como punto de partida las dos configuraciones encontradas. Finalmente, otro punto de estudio interesante en el futuro es el de evaluar esta propuesta bajo otros escenarios y conjuntos de datos que puedan tener características más complicadas que los usados.

Referencias

Ankerst, M., Breunig, M. M., Kriegel, H.-P., y Sander, J. (1999, junio). Optics: Ordering points to identify the clustering structure. *SIGMOD Rec.*, *28*(2), 49–60. Descargado de https://doi.org/10.1145/304181.304187 doi: 10.1145/304181.304187

Bandyopadhyay, S. (2011). Multiobjective simulated annealing for fuzzy clustering with stability and validity. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *41*(5), 682-691. doi: 10.1109/TSMCC.2010.2088390

Bandyopadhyay, S., y Maulik, U. (2001). Nonparametric genetic clustering: comparison of validity indices. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *31*(1), 120-125. doi: 10.1109/5326.923275

Bandyopadhyay, S., y Maulik, U. (2002). An evolutionary technique based on k-means algorithm for optimal clustering in rn. *Information Sciences*, *146*(1), 221-237. doi: https://doi.org/10.1016/S0020-0255(02)00208-6

Bandyopadhyay, S., Maulik, U., y Pakhira, M. K. (2001). Clustering using simulated annealing with probabilistic redistribution. *International Journal of Pattern Recognition and Artificial Intelligence*, *15*(02), 269-285. Descargado de https://doi.org/10.1142/S0218001401000927 doi: 10.1142/S0218001401000927

Bandyopadhyay, S., y Saha, S. (2013). Use of multiobjective optimization for data clustering. En *Unsupervised classification: Similarity measures, classical and metaheuristic approaches, and applications* (pp. 217–243). Berlin, Heidelberg: Springer Berlin Heidelberg. Descargado de https://doi.org/10.1007/978-3-642-32451-2_9 doi: 10.1007/978-3-642-32451-2_9

Bandyopadhyay, S., Saha, S., Maulik, U., y Deb, K. (2008). A simulated annealing-based multiobjective optimization algorithm: Amosa. *IEEE Transactions on Evolutionary Computation*, *12*(3), 269-283. doi: 10.1109/TEVC.2007.900837

Beume, N., Naujoks, B., y Emmerich, M. (2007). Sms-emoa: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, *181*(3), 1653-1669. Descargado de https://www.sciencedirect.com/science/article/pii/S0377221706005443 doi: https://doi.org/10.1016/

j.ejor.2006.08.008

Bezdek, J. C. (1981). *Pattern recognition with fuzzy objective function algorithms*. Springer, Boston, MA.

Brown, D. E., y Huntley, C. L. (1992). A practical application of simulated annealing to clustering. *Pattern Recognition*, *25*(4), 401-412. Descargado de https://www.sciencedirect.com/science/article/pii/003132039290088Z doi: https://doi.org/10.1016/0031-3203(92)90088-Z

Das, S., Abraham, A., y Konar, A. (2008). Automatic kernel clustering with a multi-elitist particle swarm optimization algorithm. *Pattern Recognition Letters*, *29*(5), 688-699. Descargado de https://www.sciencedirect.com/science/article/pii/S0167865507003923 doi: https://doi.org/10.1016/j.patrec.2007.12.002

Deb, K., Pratap, A., Agarwal, S., y Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182-197. doi: 10.1109/4235.996017

Dembélé, D., y Kastner, P. (2003, 05). Fuzzy C-means method for clustering microarray data. *Bioinformatics*, *19*(8), 973-980. Descargado de https://doi.org/10.1093/bioinformatics/btg119 doi: 10.1093/bioinformatics/btg119

Ester, M., Kriegel, H.-P., Sander, J., y Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. En *Proceedings of the second international conference on knowledge discovery and data mining* (p. 226–231). AAAI Press.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers Operations Research*, *13*(5), 533-549. Descargado de https://www.sciencedirect.com/science/article/pii/0305054886900481 (Applications of Integer Programming) doi: https://doi.org/10.1016/0305-0548(86)90048-1

Glover, F. (1989, aug). Tabu Search—Part I. *ORSA Journal on Computing*, *1*(3), 190–206. Descargado de http://pubsonline.informs.org/doi/10.1287/ijoc.1.3.190 doi: 10.1287/ijoc.1.3.190

Glover, F. (1990, feb). Tabu Search—Part II. *ORSA Journal on Computing*, *2*(1), 4–32. Descargado de http://pubsonline.informs.org/doi/10.1287/ijoc.2.1.4 doi: 10.1287/ijoc.2.1.4

Guerreiro, A. P., Fonseca, C. M., y Paquete, L. (2021, jul). The hypervolume indicator. *ACM Computing Surveys*, *54*(6), 1–42. Descargado de https://doi.org/10.1145%2F3453474 doi: 10.1145/3453474

Handl, J., y Knowles, J. (2006). Multi-objective clustering and cluster validation. En Y. Jin (Ed.), *Multi-objective machine learning* (pp. 21–47). Berlin, Heidelberg: Springer, Berlin, Heidelberg. Descargado de https://doi.org/10.1007/3-540

-33019-4_2 doi: 10.1007/3-540-33019-4_2

Handl, J., y Knowles, J. (2007). An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, *11*(1), 56-76. doi: 10.1109/TEVC.2006.877146

Hastie, T., Tibshirani, R., y Friedman, J. (2009). *The elements of statistical learning.* Springer-Verlag, New York. doi: 10.1007/978-0-387-84858-7

He, R., Xu, W., Sun, J., y Zu, B. (2009, 12). Balanced k-means algorithm for partitioning areas in large-scale vehicle routing problem. En (Vol. 3, p. 87 - 90). doi: 10.1109/IITA.2009.307

Heloulou, I., Radjef, M. S., y Kechadi, M. T. (2017, enero). Automatic multi-objective clustering based on game theory. *Expert Syst. Appl.*, *67*(C), 32–48. Descargado de https://doi.org/10.1016/j.eswa.2016.09.008 doi: 10.1016/j.eswa.2016.09.008

Hutter, F., Hoos, H., Leyton-Brown, K., y Stützle, T. (2009, 10). Paramils: An automatic algorithm configuration framework. *J. Artif. Intell. Res. (JAIR)*, *36*, 267-306. doi: 10.1613/jair.2861

J., M. (1967). Some methods for classification and analysis of multivariate observations. En *In fifth berkeley symposium on mathematical statistics and probability* (pp. 281–297). University of California Press.

Jain, A. K., y Dubes, R. C. (1988). *Algorithms for clustering data.* USA: Prentice-Hall, Inc.

Johnson, D. S. (1990). Local optimization and the traveling salesman problem. En *International colloquium on automata, languages, and programming* (pp. 446–461). Srpinger.

Johnson, R. K., y Sahin, F. (2009). Particle swarm optimization methods for data clustering. En *2009 fifth international conference on soft computing, computing with words and perceptions in system analysis, decision and control* (p. 1-6). doi: 10.1109/ICSCCW.2009.5379452

José-García, A., y Gómez-Flores, W. (2016). Automatic clustering using nature-inspired metaheuristics: A survey. *Applied Soft Computing*, *41*, 192-213. Descargado de https://www.sciencedirect.com/science/article/pii/S1568494615007772 doi: https://doi.org/10.1016/j.asoc.2015.12.001

Karp, R. M. (1972). Reducibility among Combinatorial Problems. En R. E. Miller, J. W. Thatcher, y J. D. Bohlinger (Eds.), *Complexity of computer computations* (pp. 85–103). Boston, MA: Springer US. Descargado de https://doi.org/10.1007/978-1-4684-2001-2_9 doi: 10.1007/978-1-4684-2001-2_9

Kennedy, J. (1997, April). The particle swarm: social adaptation of knowledge. En *Proceedings of 1997 ieee international conference on evolutionary computation (icec '97)* (p. 303-308). doi: 10.1109/ICEC.1997.592326

Kirkpatrick, S., Gelatt, C. D., y Vecchi, M. P. (1983, may). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680. Descargado de `https://www.science.org/doi/10.1126/science.220.4598.671` doi: 10.1126/science.220.4598.671

Klein, R. W., y Dubes, R. C. (1989). Experiments in projection and clustering by simulated annealing. *Pattern Recognition*, *22*(2), 213-220. Descargado de `https://www.sciencedirect.com/science/article/pii/0031320389900678` doi: https://doi.org/10.1016/0031-3203(89)90067-8

Koopmans, T. C., y Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica*, *25*(1), 53–76. Descargado 2023-05-23, de `http://www.jstor.org/stable/1907742`

Kriegel, H.-P., Schubert, E., y Zimerk, A. (2017). The (black) art of runtime evaluation: Are we comparing algorithms or implementations? *Knowledge and Information Systems*. doi: 10.1007/s10115-016-1004-2

Krishnapuram, R., y Keller, J. M. (1993). A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, *1*(2), 98-110. doi: 10.1109/91.227387

Lantz, B. (2015). *Machine learning with r*. Packt Publishing.

Law, M. H. C., Topchy, A. P., y Jain, A. K. (2004). Multiobjective data clustering. En *Proceedings of the 2004 ieee computer society conference on computer vision and pattern recognition, 2004. cvpr 2004.* (Vol. 2, p. II-II). doi: 10.1109/CVPR.2004.1315194

Lee, J., y Perkins, D. (2021). A simulated annealing algorithm with a dual perturbation method for clustering. *Pattern Recognition*, *112*, 107713. Descargado de `https://www.sciencedirect.com/science/article/pii/S0031320320305161` doi: https://doi.org/10.1016/j.patcog.2020.107713

Lin, Y., Tang, H., Li, Y., Fang, C., Xu, Z., Zhou, Y., y Zhou, A. (2022, mar). Generating clusters of similar sizes by constrained balanced clustering. *Applied Intelligence*, *52*(5), 5273–5289. Descargado de `https://doi.org/10.1007/s10489-021-02682-y` doi: 10.1007/s10489-021-02682-y

Menchaca-Méndez, A., Montero, E., Flores-Garrido, M., y Miguel-Antonio, L. (2022). An algorithm to compute time-balanced clusters for the delivery logistics problem. *Engineering Applications of Artificial Intelligence*, *111*, 104795. doi: 10.1016/j.engappai.2022.104795

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., y Teller, E. (1953, jun). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. Descargado de `http://aip.scitation.org/doi/10.1063/1.1699114` doi: 10.1063/1.1699114

Montero, E., Riff, M.-C., y Neveu, B. (2014, apr). A beginner's guide to tuning methods. *Appl. Soft Comput.*, *17*, 39–51. Descargado de `https://doi.org/`

10.1016/j.asoc.2013.12.017 doi: 10.1016/j.asoc.2013.12.017

Nonlinear programming ii: Unconstrained optimization techniques. (2019). En *Engineering optimization theory and practice* (p. 273-345). John Wiley Sons, Ltd. Descargado de https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119454816.ch6 doi: https://doi.org/10.1002/9781119454816.ch6

Nonlinear programming i: One-dimensional minimization methods. (2019). En *Engineering optimization theory and practice* (p. 225-271). John Wiley Sons, Ltd. Descargado de https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119454816.ch5 doi: https://doi.org/10.1002/9781119454816.ch5

Russell, S., Norvig, P., y Canny, J. (2003). *Artificial intelligence: A modern approach.* Prentice Hall/Pearson Education. Descargado de https://books.google.com.mx/books?id=KI2WQgAACAAJ

Selim, S. Z., y Alsultan, K. (1991). A simulated annealing algorithm for the clustering problem. *Pattern Recognition*, *24*(10), 1003-1008. Descargado de https://www.sciencedirect.com/science/article/pii/003132039190097O doi: https://doi.org/10.1016/0031-3203(91)90097-O

Sheng, W., y X., L. (2006). A genetic k-medoids clustering algorithm. *Journal of Heuristics*, *12*, 447-466. doi: https://doi.org/10.1007/s10732-006-7284-z

Stathis Sideris. (2005). *Hierarchical_ clustering_ diagrama.* Descargado de https://commons.wikimedia.org/w/index.php?curid=7344806 (Este trabbajo esta licenciado por Atribución-CompartirIgual 3.0 No portada (CC BY-SA 3.0) Para ver una copia de la licencia, vea https://creativecommons.org/licenses/by-sa/3.0/legalcode.)

Storn, R., y Price, K. (1997, 01). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*, 341-359. doi: 10.1023/A:1008202821328

van der Merwe, D. W., y Engelbrecht, A. P. (2003). Data clustering using particle swarm optimization. En *The 2003 congress on evolutionary computation, 2003. cec '03.* (Vol. 1, p. 215-220 Vol.1). doi: 10.1109/CEC.2003.1299577

Voorhees, E. M. (1986). *The effectiveness and efficiency of agglomerative hierarchic clustering in document retrieval* (Tesis Doctoral no publicada). USA. (UMI order no. GAX86-07224)

Wang, N., Wang, J. S., Zhu, L. F., Wang, H. Y., y Wang, G. (2021). A novel dynamic clustering method by integrating marine predators algorithm and particle swarm optimization algorithm. *IEEE Access*, *9*, 3557-3569. doi: 10.1109/ACCESS.2020.3047819

Wolpert, D., y Macready, W. (1997, April). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67-82. doi: 10.1109/4235.585893

Xu, R., y Wunsch, D. C. (2010). Clustering algorithms in biomedical research: A review. *IEEE Reviews in Biomedical Engineering*, *3*, 120-154. doi: 10.1109/RBME.2010.2083647

Zhang, Q., y Li, H. (2007). Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, *11*(6), 712-731. doi: 10.1109/TEVC.2007.892759

Zitzler, E., y Thiele, L. (1998, September). Multiobjective optimization using evolutionary algorithms—a comparative study. En A. E. Eiben (Ed.), *Parallel problem solving from nature v* (pp. 292–301). Amsterdam: Springer-Verlag.

Apéndice A

Visualizaciones

A.1. Visualización de los tiempos producidos por *cProfile*

Tener la información sobre la caracterización de tiempo de las funciones es ventajoso, sin embargo, visualizarlo en una tabla en la terminal o en un archivo de texto no es algo que pueda ser fácil de ver o interpretar, es por eso que para este trabajo, se utilizó un visualizador del perfilamiento de *Python* llamado *Snakeviz*.

A.1.1. Uso de *Snakeviz*

Snakeviz[1] es una librería de *Python* diseñada para tomar los datos producidos por *cProfile* y mostrarlos de una manera interactiva en el navegador. Esta herramienta puede desplegar los datos de perfilación de varias maneras, ver Figuras A.1 y A.2. Este tipo de visualizaciones ayudan a tener una mejor idea sobre qué funciones están consumiendo el mayor tiempo de ejecución así como las funciones que van derivadas de cada una. Igualmente permite visualizar la tabla generada por *cProfile* de una manera interactiva, ver Figura A.3.

[1] `https://jiffyclub.github.io/snakeviz/`

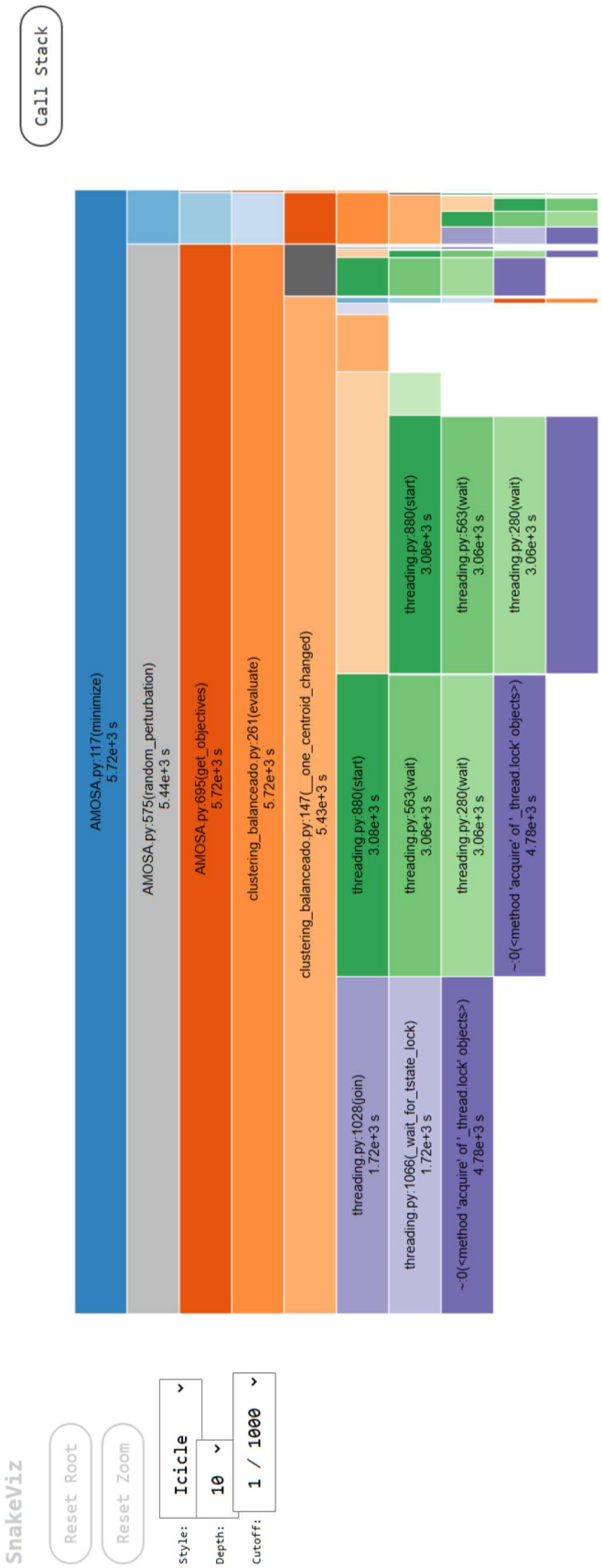


Figura A.1: Visualización en formato Icicle. Cada bloque muestra la función que se llamó, en qué archivo y línea se encuentra, y su tiempo de ejecución. El bloque de arriba representa la función base de donde el siguiente bloque es llamado.

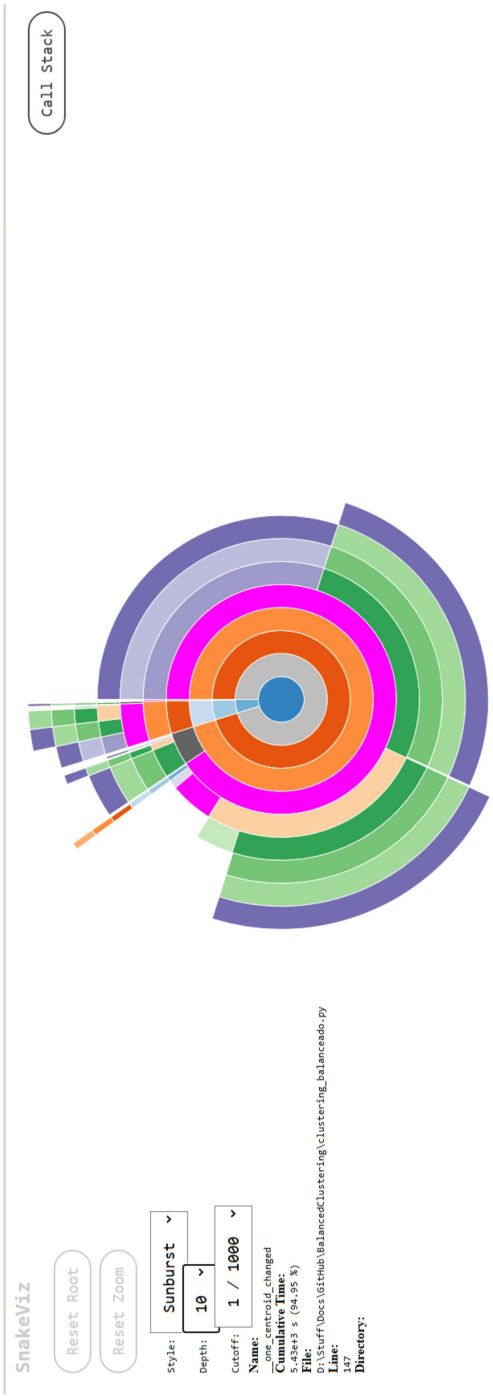
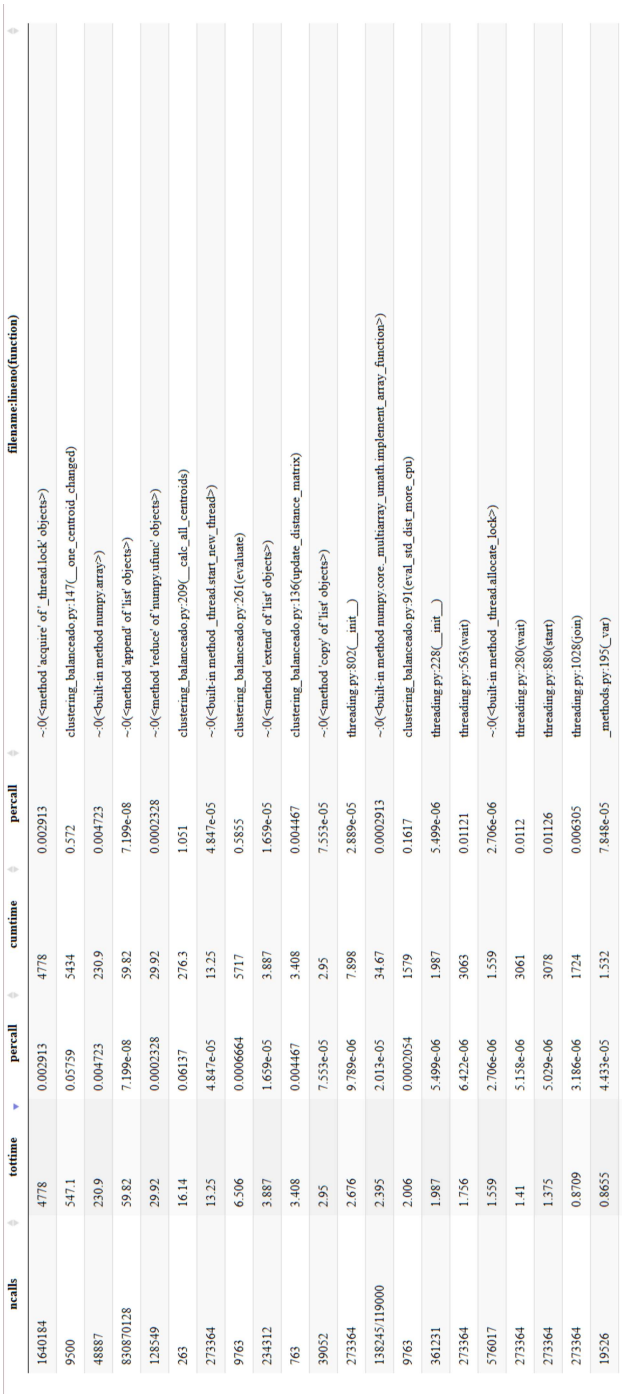


Figura A.2: Visualización en formato Sunburst. En este formato cada función se representa como un arco. La función del centro es la primera que se llama y la longitud del arco es el tiempo de la función. Un arco que casi completa una vuelta, representa una función que consume la mayoría del tiempo de la función que la manda llamar.



ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1640184	4778	0.002913	4778	0.002913	~:0(<method 'acquire' of '_thread.lock' objects>)
9500	547.1	0.05759	5434	0.572	clustering_balanceado.py:147(__one_centroid_changed)
48887	230.9	0.004723	230.9	0.004723	~:0(<built-in method numpy.array>)
830870128	59.82	7.199e-08	59.82	7.199e-08	~:0(<method 'append' of 'list' objects>)
128549	29.92	0.0002328	29.92	0.0002328	~:0(<method 'reduce' of 'numpy.ufunc' objects>)
263	16.14	0.06137	276.3	1.051	clustering_balanceado.py:209(__calc_all_centroids)
273364	13.25	4.847e-05	13.25	4.847e-05	~:0(<built-in method _thread.start_new_thread>)
9763	6.506	0.0006664	5717	0.5855	clustering_balanceado.py:261(evaluate)
234312	3.887	1.659e-05	3.887	1.659e-05	~:0(<method 'extend' of 'list' objects>)
763	3.408	0.004467	3.408	0.004467	clustering_balanceado.py:136(update_distance_matrix)
39052	2.95	7.553e-05	2.95	7.553e-05	~:0(<method 'copy' of 'list' objects>)
273364	2.676	9.789e-06	7.898	2.889e-05	threading.py:802(__init__)
138245/119000	2.395	2.013e-05	34.67	0.0002913	~:0(<built-in method numpy.core._multiarray_umath.implement_array_function>)
9763	2.006	0.0002054	1579	0.1617	clustering_balanceado.py:91(eval_std_dist_more_cpu)
361231	1.987	5.499e-06	1.987	5.499e-06	threading.py:228(__init__)
273364	1.756	6.422e-06	3063	0.01121	threading.py:563(wait)
576017	1.559	2.706e-06	1.559	2.706e-06	~:0(<built-in method _thread.allocate_lock>)
273364	1.41	5.158e-06	3061	0.0112	threading.py:280(wait)
273364	1.375	5.029e-06	3078	0.01126	threading.py:880(start)
273364	0.8709	3.186e-06	1724	0.006305	threading.py:1028(join)
19526	0.8655	4.433e-05	1.532	7.848e-05	_methods.py:195(_var)

Figura A.3: Visualización de la tabla generada por *cProfile*. En la última columna se indica el nombre de la función y el archivo y línea donde se encuentra. La primera columna indica el número de llamadas que se hicieron a la función. La segunda columna el tiempo que se pasó en la función, excluyendo el tiempo de otras funciones. La tercera columna el tiempo por llamada, el cual es calculado dividiendo el $tottime/ncalls$. La cuarta columna es el tiempo que se pasó en la función tomando en cuenta llamadas a funciones dentro de esta. Por ultimo la quinta columna es el tiempo por llamada dividiendo $cumtime/ncalls$.