UNIVERSIDAD NACIONAL AUTÓNOMA

DE MÉXICO

FACULTAD DE CIENCIAS

Metaheurísticas para la asignación de grupos en la Escuela Nacional Preparatoria

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Licenciada en Ciencias de la Computación

PRESENTA:

Alejandra Krystel Coloapa Díaz

TUTOR

Canek Peláez Valdés

Ciudad Universitaria, CDMX, 2020

Hoja de datos del jurado

1. Datos del alumno
   Coloapa
   Díaz
   Alejandra Krystel
   aleexkj@ciencias.unam.mx
   Universidad Nacional Autonoma de Mexico
   Facultad de Ciencias
   Ciencias de la Computación
   311181629

2. Datos del tutor
   Dr.
   Canek
   Peláez
   Valdés

3. Datos del sinodal 1
   Dr.
   David Guillermo
   Romero
   Vargas

4. Datos del sinodal 2
   Dra.
   Claudia Orquídea
   López
   Soto

5. Datos del sinodal 3
   Dra.
   María de Luz
   Gasca
   Soto

6. Datos del sinodal 4
   M. en C.
   Manuel Cristóbal
   López
   Michelone

7. Datos del trabajo escrito
   Metaheurísticas para la asignación de grupos en la Escuela Nacional Preparatoria
   106 p
   2020

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [15]

Agradecimientos

A mis padres, por su apoyo incondicional y por siempre alentarme a soñar más alto. A mi hermana, por acompañarme en *los miércoles de tesis*.

A la M. Ángela Villanueva, por todo lo que me ha enseñado, por su apoyo desde el día uno en la Opción Técnica hasta este momento. Ha sido un enorme placer que nuestros caminos se cruzaran.

A la M. Angélica Martínez Pérez y la dirección de la ENP 6, por brindarme los datos necesarios para la elaboración de este trabajo. A la familia de la ENP 6, que me apoyaron en todo momento resolviendo dudas sobre la asignación de grupos.

A los *tlachaches*, por todo su apoyo durante la carrera, por los viajes y risas compartidas. A Uriel, *my partner in crime*, por siempre motivarme a *salir de la cueva*.

A Canek, por toda su paciencia y apoyo. Gracias por no dejar que abandonara este trabajo.

Índice general

Lista de figuras

Lista de tablas

Lista de definiciones

Lista de acrónimos

AFA Algoritmo de Flujo de Agua. 3, 54–57, 61, 63, 67, 69, 73, 76–78

ENP Escuela Nacional Preparatoria. 2, 3, 42, 44, 79, 80, 82

PAG Problema de Asignación Generalizada. 2, 3, 10, 42–44, 48, 61

PAV Problema del Agente Viajero. 1, 5, 7–10, 15, 17, 18, 39, 55, 81

Introducción

1

1.1 Metaheurísticas para problemas de optimización

La optimización está en todas partes. Casi cualquier problema en ingeniería, ciencia, economía y la vida diaria puede ser formulado como un problema de optimización [6]. Hablamos de optimización cuando buscamos la elección de la mejor alternativa dentro de todas las posibles, probablemente bajo ciertas restricciones, por ejemplo: la elección de la ruta más rápida a un lugar sin pasar por avenidas principales, la elección de inversiones para maximizar ganancias y minimizar riesgos o la calendarización de clases para maximizar el uso de un salón.

Para optimizar necesitamos tres elementos: variables de decisión, función objetivo y restricciones. Las variables representan las alternativas existentes: qué calle tomar, la cantidad a invertir o el salón asignado a una clase. La función objetivo representa lo que buscamos maximizar o minimizar y se define a partir de las variables de decisión, es el costo o ganancia de la elección tomada. Las restricciones son funciones o cotas sobre las variables de decisión, que limitan la función objetivo y las distintas alternativas. Dependiendo del tipo y número de variables y funciones, un problema de optimización se vuelve más difícil de resolver.

Los problemas de optimización se pueden dividir fácilmente en dos categorías: aquellos con variables continuas y aquellos con variables discretas, a los que se denominan combinatorios. En el caso de problemas continuos, se busca un conjunto de números reales o incluso una función; en el caso combinatorio se busca un objeto dentro de una colección finita o contablemente finita, típicamente un entero, conjunto, permutación o gráfica. Estos dos tipos de problemas son muy diferentes y los métodos para resolverlos han divergido bastante [24].

Encontrar una solución a un problema de optimización es una tarea difícil, pues depende del tipo de problema, disponibilidad, recursos y restricciones de tiempo [8]. Los problemas combinatorios son llamativos porque son fáciles de enunciar, pero en general, son muy difíciles de resolver [34]. Mientras que problemas como la ruta más corta en una gráfica o el árbol generador de peso mínimo pueden resolverse en tiempo polinomial, otros, como el Problema del Agente Viajero (PAV) o el Problema de la mochila, pertenecen a la clase de problemas *NP*–completos.

Los métodos para resolver problemas de optimización combinatoria pueden ser clasificados como exactos o aproximados; los métodos exactos tienen la garantía de encontrar la solución óptima en un tiempo acotado, mientras que los métodos aproximados no tienen

ninguna garantía sobre la solución obtenida. Los métodos exactos usan información del problema para determinar qué soluciones son las que deben ser examinadas, sin embargo, esto no es posible para muchos problemas de tipo combinatorio.

Cuando la teoría de la computación mostró que muchos problemas de optimización eran *NP*–duros y probablemente intratables por algoritmos exactos — sin importar el creciente poder de cómputo— los métodos aproximados fueron la única forma de encontrar soluciones, en especial para los combinatorios [31]. La complejidad del problema de interés hace que sea imposible probar con todas las posibles soluciones, por lo que se opta por encontrar soluciones suficientemente buenas en una escala de tiempo aceptable.

Las metaheurísticas son reconocidas ampliamente como uno de los métodos aproximados más prácticos para resolver problemas de optimización combinatoria [37]. A diferencia de los métodos exactos, permiten atacar instancias de gran tamaño de un problema, encontrando soluciones satisfactorias en un tiempo razonable, aunque no hay garantía de encontrar la solución óptima ni acotaciones sobre el tiempo de ejecución [36].

El campo de metaheurísticas se ha sometido a cambios de paradigma que han modificado la forma en que los investigadores consideran el desarrollo de métodos heurísticos. Hubo un cambio de un periodo centrado en métodos, en el que las metaheurísticas se pensaban como algoritmos, a un periodo en el que las metaheurísticas son tomadas como estructuras de alto nivel, colecciones de conceptos e ideas, que ofrecen pautas en cómo resolver un problema de optimización [32].

Con el gran crecimiento de poder de cómputo y plataformas para paralelizar, las metaheurísticas han sido exitosas resolviendo problemas de la vida real con requisitos de tiempo de respuesta rigurosos [10]. Los problemas de optimización que parecían intratables hace unas décadas pueden ser resueltos eficientemente ahora [32].

1.2 Objetivo

En este trabajo me centraré en problemas de optimización combinatoria y el uso de metaheurísticas como marco algorítmico para resolverlos. Trataré una instancia del Problema de Asignación Generalizada (PAG): la asignación de grupos de la Escuela Nacional Preparatoria (ENP). El PAG busca una asignación de tareas (alumnos) a agentes (grupos) de forma que no se excedan los recursos (cupo del grupo) y la ganancia de la asignación sea óptima. La asignación de grupos es un problema combinatorio ya que existen muchas combinaciones de alumnos en grupos pero solo un subconjunto que satisface los criterios buscados, y todavía un subconjunto más pequeño cuya asignación tiene ganancia óptima.

Hay muchas formas de medir la ganancia de una asignación, pues esto puede variar de acuerdo a los criterios que tenga la dirección de cada plantel. Este trabajo está orientado

a la asignación de grupos en la ENP 6 "Antonio Caso", midiendo la ganancia de una asignación de acuerdo a la satisfacción de los alumnos, bajo las restricciones de cupo y balance de los grupos.

1.3 Motivación

Actualmente la ENP 6 tiene un proceso de asignación en el que los alumnos eligen un número fijo de grupos en el orden de su preferencia y posteriormente un algoritmo asigna los grupos, de forma que haya alumnos de todos los promedios en cada grupo y su cupo sea similar. En el pasado este trabajo era realizado por el personal administrativo del plantel y posteriormente fue automatizado. El algoritmo actual procesa las estadísticas de los promedios de los alumnos para determinar un número fijo de lugares por rango de promedio y procede a asignar grupos siguiendo una estrategia *glotona*[1]. Mientras que esta aproximación garantiza el balance de alumnos por promedio en cada grupo, no minimiza el número de alumnos asignados en un grupo que no es de su preferencia.

La inspiración de este trabajo surge al buscar una mejor asignación, por ejemplo: si en lugar de asignar un alumno en su primera opción y otro en su quinta, se puede asignar a ambos en su segunda. La meta es obtener el mayor número de alumnos en sus primeras opciones sin alterar el balance y cupo de los grupos. El desarrollo de un nuevo método de asignación es beneficioso por diversas razones: al tener más alumnos en sus primeras opciones se reduce la carga administrativa de hacer cambios posteriores en la asignación a petición de los alumnos; la satisfacción de los alumnos con los resultados obtenidos podría influir en su desempeño académico al motivarlos asignándoles un grupo de sus opciones; ya que en un algoritmo metaheurístico la búsqueda es guiada por la función de costo, si los criterios que describen una buena asignación cambiaran, los ajustes necesarios en la implementación son mínimos – haciendo que este nuevo método sea robusto.

1.4 Estructura del trabajo

En el segundo capítulo daré conceptos sobre los problemas de optimización combinatoria y en el tercer capítulo discutiré las metaheurísticas vistas como un marco algorítmico para resolverlos. En el capítulo cuatro formalizaré la asignación de grupos como una instancia del Problema de Asignación Generalizada y analizaré distintas formas de medir la ganancia de una asignación. En el quinto capítulo se describirá la implementación del algoritmo metaheurístico "Algoritmo de Flujo de Agua" (*Water flow-like algorithm*) para resolver la asignación de grupos. El capítulo seis mostrará los resultados obtenidos dando

[1] En un algoritmo glotón (*greedy*) siempre se toma la decisión que parece la mejor en ese momento, es decir, se elige un óptimo local con la esperanza de que éste lleve a un óptimo global. Los algoritmos glotones no siempre generan soluciones óptimas, pero para muchos problemas – como el árbol generador de peso mínimo o la ruta más corta– siempre encuentran la solución óptima.

comparativa con el algoritmo actual y el capítulo siete tendrá las conclusiones y trabajo futuro.

Problemas de optimización combinatoria

2

La optimización es una rama de las matemáticas que involucra la maximización o minimización de una o varias funciones bajo ciertas condiciones; en otras palabras: la búsqueda de la mejor solución. En el caso combinatorio buscamos el acomodo, agrupación, orden o selección óptima de objetos discretos, usualmente finitos [34].

Probablemente el problema de optimización combinatoria más conocido es el Problema del Agente Viajero (PAV)[1], en el que dada una lista de ciudades y el costo de viajar entre ellas[2], se busca la ruta más barata que visite cada ciudad exactamente una vez y regrese a la ciudad inicial [21]. El PAV ha sido estudiado por más de cincuenta años y hasta ahora no se conoce un algoritmo de tiempo polinomial que lo resuelva, lo cual sucede con muchos problemas de optimización combinatoria.

En este capítulo se dará la definición de un problema de optimización y términos relacionados, haciendo énfasis en los combinatorios. Se discutirá una visión general de los métodos de optimización y su clasificación.

2.1 Modelos de optimización

Cuando se tiene un problema de la vida real en el que se busca optimizar, lo primero que necesitamos es su *modelo* matemático, es decir, una representación del problema y sus características en términos matemáticos.

El diseño del modelo de un problema de optimización puede ser una tarea difícil, ya que los problemas de la vida real suelen ser muy complicados para capturar todos sus detalles. Se tiene un balance entre la facilidad para resolverlo y la validez del problema: entre más cercano a la realidad sea el modelo se vuelve más difícil de resolver; pero si está muy simplificado, las soluciones obtenidas podrían no ser de utilidad [28].

Un modelo de optimización tiene tres componentes principales: variables de decisión, restricciones y objetivo.

Variables de decisión

Variables $x \in D$ asociadas a cada decisión y sus posibles valores. Una *solución* $\overline{x} = (v_1, \ldots, v_n) \in D_1 \times \ldots \times D_n$ consiste en la elección de valores para cada varia-

[1] Por su nombre en inglés “The travelling salesman problem“ formulado por primera vez en 1930 [21].

[2] Suponiendo costos simétricos, es decir, viajar de la ciudad X a la ciudad Y cuesta lo mismo que viajar de Y a X.

ble de decisión. El conjunto S de todas las posibles soluciones es llamado *espacio de búsqueda*[3].

Restricciones

Pueden ser de dos tipos: igualdad y desigualdad. Las primeras son funciones que restringen los valores, mientras que las segundas definen una relación entre las variables de decisión, por ejemplo $x_1 + x_2 \leq L$. Decimos que una solución $\overline{x}$ es *factible* si satisface todas las restricciones. El subconjunto $F \subseteq S$ de soluciones factibles es denominado *región factible*.

Objetivo

Una función $f : S \to \mathbb{R}$ que asigna un valor real a cada solución y da una relación de orden total sobre S. Esta función evalúa qué tan buena es la elección tomada. Nos referimos a $f(\overline{x})$ como el *costo* de la solución en problemas de minimización, y como la *ganancia* de una solución cuando buscamos maximizar. A f se le denomina *función objetivo*[4].

2.1.1 Clases de modelos

Es común encontrar más de un modelo para el mismo problema; la diferencia entre ellos involucra la eficiencia de los algoritmos que pueden usarse y la precisión de las soluciones obtenidas.

Los modelos más exitosos están basados en programación matemática y programación restringida [36]. Cuando las variables de decisión son continuas, el modelo más común es la programación lineal pues para estos problemas existen algoritmos exactos. Sin embargo, cuando hay variables discretas (que por ende son indivisibles, como el número de personas), los modelos que usen variables continuas son inapropiados [36]. En programación restringida el modelo está basado en describir las propiedades de la solución buscada; este tipo de modelado usualmente es aplicado en problemas de calendarización pero es menos apto para problemas con un gran número de soluciones factibles [36].

Existen problemas para los que no es suficiente describir sus variables de decisión, restricciones y función objetivo, sino que se requieren parámetros adicionales que pueden ser obtenidos analizando o simulando procesos de la vida real [28]. Además, para muchos problemas de aplicación práctica, no siempre se cuenta con la disponibilidad de modelos matemáticos. Por ejemplo, en algunas aplicaciones se debe simular un proceso físico para evaluar la función objetivo, lo cuál es imposible si se usa un modelo analítico [36].

Ejemplo 2.1

Dada una lista $C = \{c_1, \ldots, c_n\}$ de n ciudades y una matriz de distancias $d_{n \times n}$ entre cada

[3] También conocido como espacio de configuraciones o de estados.

[4] También conocida como función de costo, utilidad, aptitud o energía.

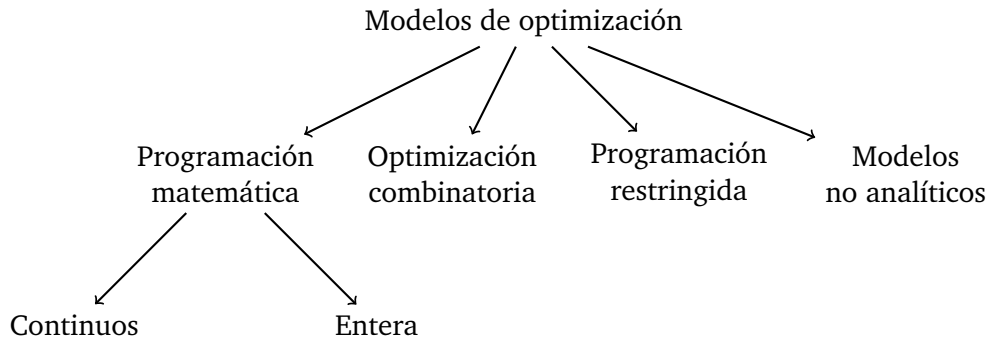


Figura 2.1 – Modelos clásicos de optimización; clasificación no exclusiva [36].

dos ciudades, un posible modelo para el PAV se basa en la permutación de las ciudades, teniendo una variable de decisión que es el orden en que se visitan. Una solución consiste en una permutación de C, por lo que el espacio de búsqueda es el conjunto

$$S_\pi = \{\text{permutaciones } \pi \text{ de } n \text{ objetos}\}.$$

Sea $\pi(j)$ la ciudad en la j–ésima posición, la función objetivo f toma una permutación y le asocia el costo

$$f(\pi) = \sum_{j=1}^{n-1} d_{\pi(j),\pi(j+1)} + d_{\pi(n),\pi(1)}.$$

Siguiendo este modelo no habría restricciones sobre las variables, pues al definir S_π como el conjunto de permutaciones garantizamos que solo se visita una vez cada ciudad.

Usando programación entera podemos dar otro modelo con variables binarias por cada arista en la gráfica no dirigida $G = (V, E)$ asociada a d:

$$x_{ij} = \begin{cases} 1 & (i,j) \in E \text{ está en el ciclo,} \\ 0 & \text{en otro caso,} \end{cases}$$

ya que $d_{i,j} = d_{j,i}$, basta considerar solo las variables x_{ij} con $i < j$. Denotemos a $\mathcal{D}$ como el número de variables. Una solución $\overline{x}$ consiste en la elección de $\mathcal{D}$ aristas.

La función objetivo se puede definir como sigue:

$$f(\overline{x}) = \sum_{(i,j)\in E, i<j} d_{i,j} x_{ij}.$$

$(1,2,3)$	$(2,3,1)$
$(1,3,2)$	$(3,1,2)$
$(2,1,3)$	$(3,2,1)$

(a) Espacio de búsqueda S_π.

$(1,1,1)$	$(0,0,1)$
$(1,1,0)$	$(0,1,1)$
$(1,0,1)$	$(0,1,0)$
$(1,0,0)$	$(0,0,0)$

(b) Espacio de búsqueda $S_{0,1}$.

Figura 2.2 – Instancia de PAV con $C = \{1,2,3\}$. Espacios de búsqueda de acuerdo al ejemplo 2.1.

Este modelo requiere restricciones sobre una solución $\overline{x}$ para evitar que una ciudad se visite más de una vez y que las aristas tomadas formen un solo ciclo:

$$\sum_{j \in V} x_{ij} = 2, \ \forall i \in V \tag{2.1}$$

$$\sum_{i,j \in V, i \neq j} x_{ij} = n, \tag{2.2}$$

por lo que el espacio de búsqueda es el conjunto

$$S_{0,1} = \{\overline{x} = (v_{1,2}, \ldots, v_{1,n}, v_{2,3}, \ldots, v_{n,n-1}) \mid v_i \in \{0,1\}\},$$

donde

$$\mid S_{0,1} \mid = 2^{\mathcal{D}}.$$

La figura 2.2 muestra una instancia del PAV con tres ciudades y los espacios de búsqueda para ambos modelos. En el caso de programación entera, podemos ver que la solución $(1,1,0)$ no es factible pues consta de las aristas $\{(1,2),(1,3)\}$ lo cual no cumple las restricciones definidas. ⊣

2.2 Problemas de optimización

En programación lineal es común enunciar un problema de optimización como sigue:

$$\text{mín } f(x_1, \ldots, x_n),$$

cumpliendo que

$$\begin{aligned} h_k(x_1, \ldots, x_n) \leq 0 \quad & k = 1, \ldots, m, \\ g_j(x_1, \ldots, x_n) = 0 \quad & j = 1, \ldots, p, \text{ y} \\ x_i \in D_i \quad & i = 1, \ldots, n, \end{aligned}$$

donde g, h son restricciones y f es la función objetivo. Sin embargo, podemos dar una definición más general que incluya a casi cualquier modelo:

Definición 2.1
Una *instancia de un problema de optimización*[5] es una tupla $P = (S, f)$ donde S es el espacio de búsqueda y $f : S \to \mathbb{R}^+$ es la función objetivo. Sea F la región factible, el objetivo es encontrar $s^* \in F$ tal que

$$f(s^*) \leq f(s), \forall s \in F.$$

A s^* se le denomina *óptimo global*. ⊣

Aunque se define como problema de minimización, también podemos definirlo como problema de maximización reemplazando $f(s)$ por $-f(s)$. En lo que resta del capítulo se hablará de problemas de optimización refiriéndose a la versión de minimización.

Definición 2.2
Un *problema de optimización* es un conjunto $\mathcal{I}$ de instancias de un problema de optimización. ⊣

Coloquialmente, una instancia de un problema de optimización es un problema concreto, donde las variables son proporcionadas y tenemos información suficiente para encontrar una solución. Un problema de optimización es la colección de instancias con las mismas propiedades y generadas de manera similar [28]. En el caso del PAV, el problema de optimización consiste en la pregunta sobre ciclos hamiltoneanos en cualquier gráfica, mientras que una instancia consta de la lista de ciudades y sus distancias.

2.2.1 Clasificación

Dependiendo del tipo y número de variables de decisión, la naturaleza y número de funciones, tenemos distintos tipos de problemas de optimización. Las categorías más comunes se muestran en la tabla 2.1.

[5] La traducción más adecuada sería *ejemplar*, pero se opta por "instancia" al ser el término más común en la literatura.

Base	*Descripción*
Tipo de variables de decisión	Si son binarias, enteros u objetos discretos, el problema es combinatorio. Si toman valores reales, el problema es continuo.
Naturaleza de la función de costo y restricciones	Se clasifican en lineales cuando todas las funciones son lineales, y no lineales si alguna función es no lineal. Programación geométrica y cuadrática son ejemplos de problemas no lineales.
Número de restricciones	Con o sin restricciones.
Número de funciones objetivo	Con una o más funciones objetivo, a estos últimos se les conoce como *multiobjetivo*.
Naturaleza de las variables de decisión	Son deterministas si las variables de decisión y parámetros están definidos; o estocásticos si alguna variable o parámetro es descrita por una variable aleatoria.

Tabla 2.1 – Tipos de problemas de optimización.

En un problema de optimización combinatoria, las variables de decisión son discretas y el espacio de búsqueda es finito. Ejemplos de este tipo de problemas[6] son el PAV, Problema de la mochila, el *árbol generador de peso mínimo* y el PAG.

2.2.2 Espacio de búsqueda y vecindarios

El espacio de búsqueda está definido implícitamente por las variables de decisión $x \in X$. Cuando el espacio de búsqueda es discreto se produce una *explosión combinatoria*; es decir, hay un crecimiento exponencial del tamaño del espacio de búsqueda en relación a las variables y sus dominios.

Un espacio de búsqueda puede dar relaciones sobre dos soluciones, dada una solución factible $s \in S$ de un problema en particular, es útil definir un conjunto $N(s)$ de soluciones similares.

Definición 2.3
Dado el problema de optimización $P = (S, f)$, un *vecindario* es un mapeo $N : S \to 2^S$ que asigna a cada $s \in S$ un conjunto $N(s) \subseteq S$ de soluciones. ⊣

Un vecindario asocia soluciones que son similares de cierta forma, por lo que necesitamos una manera de determinar similitudes. Un espacio de búsqueda métrico es una forma

[6] Algunas definiciones se pueden encontrar en el apéndice A.

particular de un espacio topológico [28], donde la similitud entre dos soluciones es dada por una métrica.

Definición 2.4
Una *métrica* para un conjunto U es una función $d : U \times U \to \mathbb{R}$ tal que:

- $d(x, y) \geq 0,\ \forall x, y \in U$,
- $d(x, x) = 0,\ \forall x \in U$,
- $d(x, y) = d(y, x),\ \forall x, y \in U$,
- $d(x, z) \leq d(x, y) + d(y, z),\ \forall x, y, z \in U$.

⊣

Ejemplos de métricas son:

- La distancia absoluta:

$$d(x, y) = |x - y| \text{ para } x, y \in \mathbb{R}.$$

- La distancia Manhattan[7]:

$$d(x, y) = |x_1 - y_1| + |x_2 - y_2| \text{ para } x = (x_1, x_2), y = (y_1, y_2) \in \mathbb{R}^2.$$

- La distancia Euclidiana:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \text{ para } x, y \in \mathbb{R}^n.$$

- La distancia de Hamming[8]:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \text{ para } x, y \in \{0, 1\}^n,$$

y su generalización para variables continuas y discretas:

$$d(x, y) = \sum_{i=1}^{n} z_i,$$

[7] También conocida como la métrica del taxista o distancia L_1, mide la diferencia entre coordenadas.
[8] Usada generalmente en teoría de la información, mide el número de elementos en los que x y y difieren.

donde

$$z_i = \begin{cases} 0 & \text{si } x_i = y_i, \\ 1 & \text{si } x_i \neq y_i. \end{cases}$$

Si somos capaces de definir una métrica sobre el espacio de búsqueda, podemos definir los vecindarios como sigue:

Definición 2.5

Un ε–vecindario de $x \in S$ es el conjunto $N_\varepsilon(x) = \{y \in S \mid d(x, y) \leq \varepsilon\}$ de soluciones a una distancia ε de x. ⊣

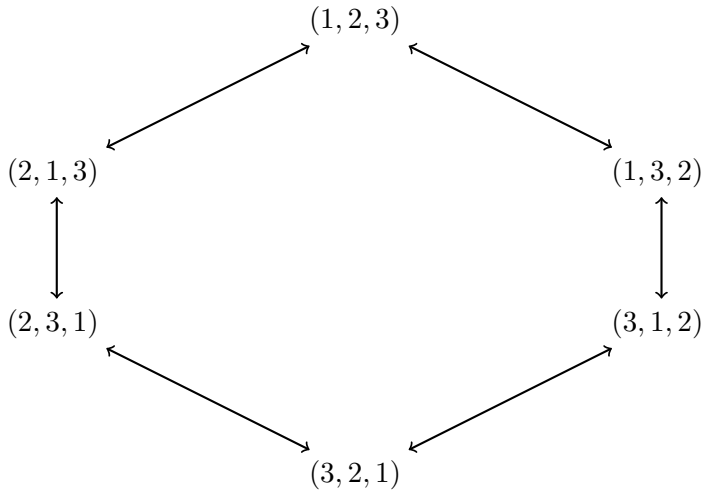


Figura 2.3 – Ejemplo 2.1 (continuación) — Gráfica $(S_\pi, N_2(S))$ del 2–vecindario de una permutación usando la distancia de Hamming: dos soluciones son adyacentes si difieren en a lo más dos ciudades.

La elección del vecindario $N(s)$ dependerá mucho de la estructura de S, pues para el mismo problema puede haber distintos modelos que resulten en vecindarios distintos. Por otra parte, no siempre es posible definir métricas ya que el problema puede carecer de similitudes significativas entre decisiones. Para estos problemas la única opción es que todas las soluciones sean vecinas [28].

2.2.3 Óptimos locales y globales

Encontrar el óptimo global de una instancia de un problema de optimización puede ser muy complicado, pero muy seguido es fácil encontrar una solución que es la mejor dentro de un vecindario. Cuando esto pasa, hablamos de óptimos locales.

Definición 2.6

Dada una instancia de un problema de optimización $P = (S, f)$ y un vecindario N, un *óptimo local* es una solución $\widehat{s} \in S$ tal que

$$f(\widehat{s}) \leq f(s),\ \forall s \in N(\widehat{s}).$$

⊣

Definición 2.7

Dada una instancia de un problema de optimización $P = (S, f)$ y un vecindario N, decimos que N es *exacto* si cada que $s \in S$ es un óptimo local también es un óptimo global. ⊣

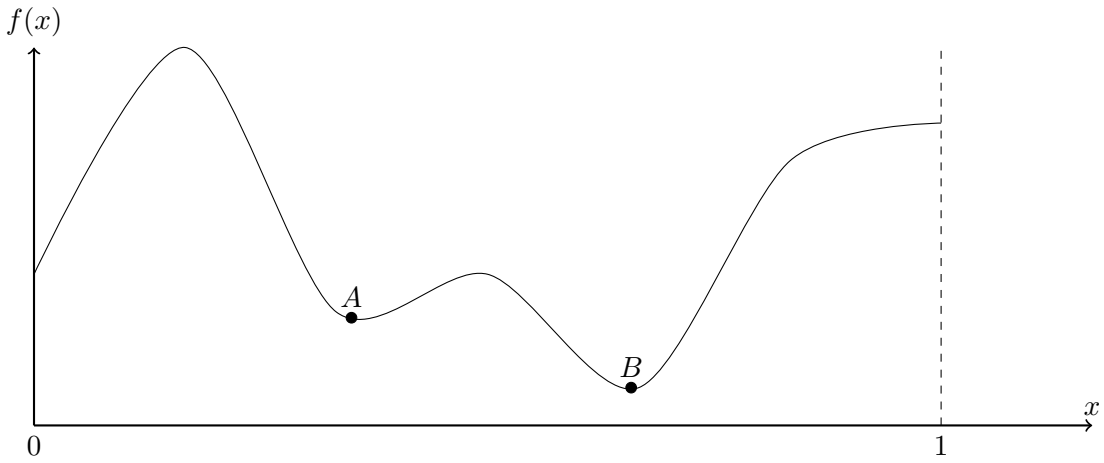


Figura 2.4 – Dada la instancia $([0, 1], f)$ y un ε–vecindario usando la distancia Euclidiana, si ε es lo suficientemente pequeña, tanto A como B son óptimos locales, pero solo B es un óptimo global; si $\varepsilon > 1$, entonces el vecindario es exacto.

El número de óptimos locales dependerá de la elección del vecindario. La modalidad de un problema describe el número de óptimos locales en el mismo: un problema es *unimodal* si tiene un solo óptimo local (que a su vez es el óptimo global); y un problema es *multimodal* si tiene más de un óptimo local, siendo éstos los más difíciles.

Aunque el objetivo es encontrar el óptimo global, muchas veces nos conformamos con una solución cercana a la óptima; lo cual introduce el concepto de solución subóptima.

Definición 2.8

Dada una solución factible x y el óptimo global x^*, decimos que x es cercana a la solución óptima o subóptima (*near-optimal*) si $f(x) - f(x^*) < \varepsilon$. ⊣

2.2.4 Paisaje adaptativo

El concepto de paisaje adaptativo (*fitness landscape*) fue introducido en biología evolutiva por Sewall Wright; el paisaje relaciona el genotipo con el éxito de un individuo.

La analogía en los problemas de optimización es la relación de la distancia entre dos soluciones con la diferencia de su costo.

Definición 2.9
Un paisaje adaptativo es la tripleta (S, f, d) donde S es el espacio de búsqueda, f la función objetivo y d la métrica definida en S. ⊣

Podemos imaginar el paisaje adaptativo como una región montañosa con colinas, mesetas y valles (ver figura 2.5): si vemos el vecindario N inducido por d como una gráfica, donde los nodos representan las soluciones y dos nodos s, s' son adyacentes si $s \in N(s')$ y consideramos la función objetivo como una altitud. En esta analogía, un algoritmo de optimización es un camino en la gráfica del vecindario y su eficiencia dependerá de las características del espacio de búsqueda [34].

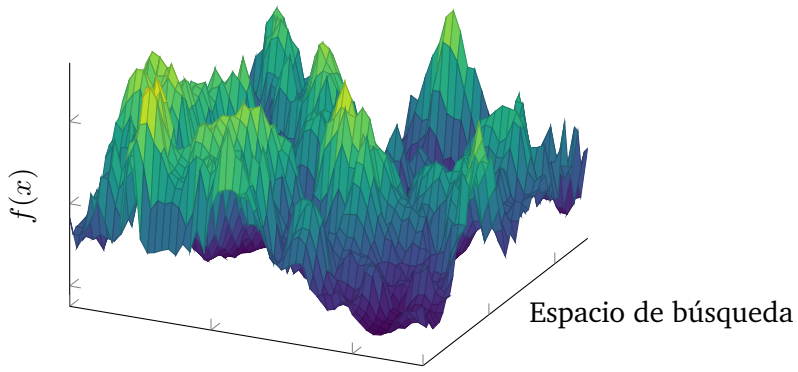


Figura 2.5 – Visualización del paisaje adaptativo de un problema de optimización.

El objetivo de un problema de optimización es encontrar el óptimo global, o en su defecto, soluciones casi óptimas u óptimos locales. Conocer más del problema es relevante pues nos permite explotar características que guíen la búsqueda de una solución óptima.

La *localidad* de un problema describe que tan bien la distancia entre dos soluciones $x, y \in S$ corresponde a la diferencia $| f(x) - f(y) |$, por lo que es un indicio sobre que tan difícil es la instancia a resolver.

La localidad es alta si las soluciones vecinas tienen costo similar [28], cuando esto pasa las soluciones de alta calidad están agrupadas y podemos hacer uso de una solución buena para buscar en su vecindario por soluciones similares, esperando encontrar una mejor. Por otro lado, si la localidad es baja, la calidad de soluciones vecinas difiere mucho y algoritmos que usen el vecindario para mejorar una solución se comportan como búsqueda aleatoria.

El coeficiente de correlación costo-distancia [28, p. 31], la función de *autocorrelación* [28, p. 35] y la función de *correlación de camino aleatorio* [28, p. 35] pueden ser usadas para medir la dificultad de un problema de acuerdo a su localidad. Estas funciones

dan estimados apropiados solo si el espacio de búsqueda es *regular*, es decir, todas las soluciones en el espacio de búsqueda tienen la misma probabilidad de ser visitadas en un camino aleatorio [28].

La *descomponibilidad* de un problema describe que tan bien se puede descomponer el problema original en problemas más pequeños e independientes. La descomponibilidad de un problema es alta si la estructura de la función objetivo es tal que no todas las variables deben considerarse simultáneamente, sino que hay grupos de variables independientes entre ellos; es decir, la función objetivo puede calcularse como $f(x) = \sum_{g \in G_s} f'(g)$ donde G_s es una partición de las variables de decisión.

Algunos algoritmos de optimización descomponen el problema en otros más pequeños que sean más fáciles de resolver, y componen la solución del problema original. Un factor determinante para la descomponibilidad es el tipo y número de variables de decisión.

El análisis del paisaje adaptativo es común al querer más información sobre una clase de problemas, más allá de una instancia particular. Sin embargo, esto puede ser una tarea complicada al tratar con problemas de muchas dimensiones. En la literatura podemos encontrar análisis sobre el paisaje adaptativo de distintos problemas, por ejemplo, Mohammad-H. y col. [20] proponen un análisis para el PAV.

2.2.5 Complejidad

La dificultad de un problema de optimización describe que tan difícil es encontrar una solución óptima de un problema o instancia en específico. La dificultad es definida independientemente del método que se use para resolverlo pues debe ser la misma para todos los posibles métodos de optimización.

Dado un problema bien definido, hacemos uso de algoritmos para encontrar una solución. Un algoritmo es una secuencia finita de instrucciones bien definidas que siempre terminan y que resuelven un problema. Un algoritmo necesita dos recursos importantes para resolver un problema: tiempo y espacio. La complejidad en tiempo (espacio) de un algoritmo es el número de pasos (memoria) requeridos para resolver un problema de tamaño n. En el caso de problemas de optimización combinatoria el tamaño de un problema dependerá de la representación de una gráfica; conjunto de números; familia de conjuntos; etcétera.

La complejidad generalmente se define en términos del análisis del peor caso y con instancias muy grandes, pues esto determinará el límite de aplicabilidad del algoritmo. Nos interesa saber la cota asintótica que caracterice el crecimiento del tiempo como una función del tamaño del problema.

La notación de la O grandota (*Big-O*) usa el análisis asintótico y es una de las herramientas más populares en el análisis de algoritmos.

Definición 2.10
Un algoritmo tiene complejidad $f(n) = O(g(n))$ si existen constantes positivas n_0, c tal que

$$\forall n \geq n_0,\ f(n) \leq cg(n).$$

⊣

En este caso, la función $f(n)$ está acotada por arriba por la función $g(n)$. La notación de la O grandota puede ser usada para describir la complejidad en tiempo y espacio de un algoritmo.

Decimos que un algoritmo es de *tiempo polinomial* si su complejidad en tiempo es $O(p(n))$ donde $p(n)$ es un polinomio en función de n; si su complejidad es de la forma $O(k^n)$, decimos que el algoritmo es de *tiempo exponencial*. En 1960 Edmonds se apropió de la idea de llamar *eficiente* a un algoritmo si es de tiempo polinomial [30]; esto se ha vuelto el estándar en la comunidad científica al referirse a un algoritmo como eficiente.

La complejidad de un problema es equivalente a la complejidad del mejor algoritmo que lo resuelve. Un problema es *tratable* si existe un algoritmo de tiempo polinomial que lo resuelva; a su vez, nos referimos a un problema como *intratable* si su cota mínima es exponencial.

La teoría de la complejidad trata con problemas de decisión, que siempre tienen respuesta sí o no, por ejemplo: *¿es el número Q primo?* Existe otro tipo de problemas denominados *indecidibles* para los que no puede existir un algoritmo que los resuelva, como es el caso del problema del paro (*Halting problem*): "Dado un programa y su entrada, alguna vez termina?", para el que Turing demostró que no existe un algoritmo que responda correctamente todas las instancias de este problema.

2.2.5.1 Clases de complejidad

La teoría de la computación categoriza los problemas de decisión en clases de complejidad, de acuerdo a los recursos necesarios para resolverlos. Una clase de complejidad es un conjunto de problemas donde la cantidad de recursos (tiempo o espacio) requeridos muestran el mismo comportamiento asintótico.

Existen dos clases de complejidad importantes: *P* y *NP*. La clase *P* representa el conjunto de problemas de decisión que pueden resolverse en tiempo polinomial; los problemas en esta clase son relativamente *fáciles* de resolver. La clase *NP* representa el conjunto de problemas de decisión que pueden resolverse en tiempo polinomial de forma no determinista, por lo que $P \subseteq NP$.

Existen problemas que pueden ser *transformados* a otro problema en *NP*, a los que se conoce como *NP*–duros. Informalmente, un problema es *NP*–duro si es tan difícil como

cualquier otro problema en *NP*. Un problema es *NP*–completo si pertenece a *NP* y es *NP*–duro, siendo éstos los más difíciles en *NP*.

Los problemas en *P* son tratables, por lo que existen algoritmos eficientes para resolverlos. No está claro si los problemas *NP*–completos son tratables, pues hasta ahora no se ha encontrado un algoritmo de tiempo polinomial que los resuelva ni se ha demostrado que son intratables. Demostrar si $P \neq NP$ es una de las preguntas más importantes en la teoría de la computación que permanece abierta [9].

2.2.5.2 Tres versiones de un problema

Un problema de optimización tiene tres versiones [24, cap. 15.2]:

- Búsqueda. El problema es encontrar la solución óptima.
- Evaluación. El problema es encontrar el costo de la solución óptima.
- Decisión. El problema es decidir si, dada una cota L, existe una solución con costo menor o igual a L.

Ejemplo 2.2
Para el PAV, sus tres versiones son:

- Encontrar el ciclo hamiltoneano de peso mínimo.
- Encontrar el peso del ciclo hamiltoneano mínimo.
- Dada una lista de ciudades, ¿existe un ciclo hamiltoneano de peso menor o igual a L?

⊣

Dada una instancia de un problema de optimización $P = (S, f)$ definimos $\mathcal{A}_S$ como el algoritmo que dado s y un conjunto de parámetros P, decide si $s \in S$ y $\mathcal{A}_f$ como el algoritmo que dado s y un conjunto de parámetros Q, calcula $f(s)$. Supongamos que $\mathcal{A}_f$ y $\mathcal{A}_S$ son algoritmos de tiempo polinomial.

Ejemplo 2.3
Para el PAV, definimos a P como n y a Q como la matriz de distancias $d_{n\times n}$. $\mathcal{A}_S$ determina si s es una permutación de tamaño n y $\mathcal{A}_f$ calcula la distancia recorrida dada la matriz d. Ambos son algoritmos de tiempo polinomial: $\mathcal{A}_S$ verifica que la permutación corresponda al conjunto $\{1, \ldots, n\}$; por otra parte $\mathcal{A}_f$ calcula el costo sumando las entradas en d. ⊣

[9] Esta pregunta es uno de los siete problemas del milenio [14] propuestos por el *Instituto Clay de Matemáticas* en el año 2000.

Si tenemos un algoritmo que resuelva la versión de búsqueda, podemos resolver la versión de evaluación usando $\mathcal{A}_f$ sobre el óptimo global. A su vez, si tenemos el valor del óptimo global podemos modificar una solución hasta que tenga el mismo costo[10]. Esto implica que la versión de búsqueda tiene la misma complejidad que la versión de evaluación.

Dado el costo c del óptimo global, podemos transformar la versión de evaluación a su versión de decisión respondiendo **Sí** si $c \leq L$ y **No** en otro caso. A su vez, si podemos responder la versión de decisión entonces podemos preguntar $O(\log c)$ veces[11] por un costo c' hasta llegar a c. Esto implica que la versión de evaluación es equivalente a la versión de decisión.

Ya que las tres versiones son equivalentes, podemos usar las clases de complejidad de los problemas de decisión para clasificar los problemas de optimización. Si la versión de decisión de un problema es *NP*-completo, entonces el problema de optimización es *NP*–duro [34]. Esta relación es útil cuando intentamos mostrar la dificultad de un problema de optimización: si se puede probar que un problema de decisión es díficil, también se demuestra que el problema de optimización asociado es díficil. Desde que las clases de complejidad fueron definidas, para casi todo problema de optimización combinatoria se ha probado que su versión de decisión es *NP*–completo o se ha encontrado un algoritmo de tiempo polinomial que lo resuelva [30]. Por otra parte, existen problemas como el problema del isomorfismo de una gráfica para el que no se ha probado que esté en *P* ni que sea *NP*–completo.

Ejemplos de problemas de optimización en *P* son: el árbol generador de peso mínimo; el camino más corto entre dos vértices; problemas de flujo en gráficas y programación lineal. Ejemplos de problemas de optimización en *NP*: problemas de calendarización; asignación; agrupación, como el problema de la mochila y el PAV.

2.3 Métodos de optimización

Podemos clasificar los algoritmos para resolver problemas de optimización en *exactos* y *aproximados*. Los métodos exactos exploran a base de análisis el espacio de búsqueda para encontrar la solución óptima. Los métodos aproximados buscan una solución óptima aunque no tengan garantía de encontrarla.

Los métodos exactos son relativamente nuevos con la introducción del campo de Investigación de Operaciones alrededor de la Segunda Guerra Mundial [32]. Hasta 1970 el desarrollo de métodos de optimización se centraba en métodos exactos; esto cambió cuando se volvió claro que muchos problemas de la vida real eran *NP*–completos y

[10] No existe un método general para lograr esto, pero variaciones de programación dinámica son usadas [24].

[11] Bajo la suposición de que el costo es un entero con logaritmo acotado por un polinomio en relación al tamaño del problema.

los métodos exactos no podrían resolverlos. Desde entonces el desarrollo de métodos aproximados ha ganado más importancia.

El algoritmo seleccionado para optimizar dependerá del tipo de problema; la naturaleza del algoritmo; la calidad de la solución deseada; los recursos disponibles; y la experiencia del desarrollador [39].

2.3.1 Métodos exactos

Para problemas de optimización continuos, podemos encontrar muchos métodos basados en el análisis matemático. Los problemas lineales pueden ser resueltos con el algoritmo *Simplex* (Dantzig) y *métodos de punto interior*. Existen algoritmos inspirados en Simplex para problemas no lineales como el algoritmo de *Nelder y Mead*.

Los métodos exactos para resolver problemas de optimización combinatoria son basados en enumeración. Métodos comunes son el algoritmo A* y acercamientos de ramificación y corte (*branch and bound*) que descartan partes del espacio de búsqueda donde la solución óptima no puede ser encontrada. Programación dinámica y métodos *de planos cortantes* son otras alternativas para problemas combinatorios.

Aunque existen alternativas para problemas de optimización combinatoria, los problemas de la vida real suelen tener un espacio de búsqueda muy grande incluso descartando regiones donde no se puede encontrar un óptimo. Por si fuera poco, algunos de estos métodos tienen complejidad exponencial por lo que solo pueden ser aplicados para instancias pequeñas de problemas *NP*–duros, pues entre más grande sea el espacio de búsqueda, los problemas se vuelven intratables.

Profundizar en métodos exactos sale del alcance de este trabajo, pero una descripción más detallada puede ser encontrada en [28], [30] y [39].

2.3.2 Métodos aproximados

Pueden dividirse en dos: algoritmos de aproximación y algoritmos heurísticos. Los algoritmos de aproximación encuentran soluciones con calidad demostrable y tiempo de ejecución acotado; mientras que las heurísticas son métodos que encuentran soluciones buenas en tiempo razonable: es decir, no tienen garantía sobre la calidad de la solución ni una cota sobre el tiempo de ejecución.

2.3.2.1 Algoritmos de aproximación

Son algoritmos que encuentran soluciones a una distancia fija del óptimo global.

Definición 2.11
Un algoritmo es una ε–aproximación si

- tiene complejidad polinomial,

- para cualquier instancia del problema encuentra una solución a tal que

$$a \leq \varepsilon * s \quad \text{si } \varepsilon \geq 1,$$
$$\varepsilon * s \leq a \quad \text{si } \varepsilon \leq 1,$$

  donde s es el óptimo global y ε es una constante o está en función del tamaño del problema.

⊣

Existen problemas que no pueden ser aproximados por un factor constante, por lo que encontrar una solución subóptima es tan difícil como encontrar la óptima [34]. Estos algoritmos suelen estar muy lejos del óptimo global, por lo que no son muy útiles en problemas de la vida real.

2.3.2.2 Métodos heurísticos

Los métodos heurísticos son algoritmos que requieren del conocimiento del problema para la búsqueda de una solución. Pueden clasificarse en dos: metaheurísticas y heurísticas específicas.

Una heurística se considera una *regla de dedo*[12] que explota trucos, simplificaciones y conocimiento sobre el problema [28]. Son diseñadas para resolver un problema en particular, por lo que suelen encontrar buenas soluciones de manera eficiente.

Las metaheurísticas se definen como un marco algorítmico para resolver problemas de optimización, mezclando estrategias para explorar el espacio de búsqueda de forma eficiente. Han probado ser útiles en resolver un gran número de problemas complejos de optimización de la vida real [36]. Su gran aplicabilidad ha llevado a que métodos aproximados tengan más relevancia en la solución de problemas de optimización, pues permiten reducir la diferencia entre la realidad y el modelo, permitiendo resolver problemas más relevantes [28]. El precio: no hay garantía sobre las soluciones obtenidas.

[12] De la expresión en inglés "rule of thumb" definida como un método basado en la experiencia y sentido común, que si bien puede ser correcto no es científicamente preciso [19].

Metaheurísticas

3

Las metaheurísticas son parte de los métodos aproximados para resolver problemas de optimización, siendo el más usado para los problemas combinatorios. Son atractivas por diversas razones:

- Simplicidad y robustez. Pueden manejar problemas muy complejos incluso si propiedades matemáticas del problema no están a la mano y aún así obtener buenas soluciones.
- Eficiencia. Encuentran soluciones en tiempo razonable y, en muchos casos, su eficiencia no disminuye mucho si la estructura del problema o los parámetros cambian.
- Flexibilidad. Sus implementaciones suelen tener varios parámetros para controlar la búsqueda, los cuales tienen que ser configurados con cuidado para obtener buenas soluciones.

Muchas de las metaheurísticas desarrolladas en los últimos años son inspiradas por la naturaleza, y las estrategias de búsqueda dependen mucho de la filosofía de cada una. Aunque en realidad no sepamos cómo la naturaleza resolvería un problema de optimización *NP*–duro, si tuviera ese propósito, las similitudes entre los objetivos de la naturaleza y problemas de optimización han resultado en buenas heurísticas, no tanto por la metáfora en la que se basan sino por la traducción y adaptación a algoritmos adecuados [29].

En este capítulo se discutirá la definición de metaheurística comparando con términos relacionados; elementos de diseño y pautas para su implementación; se discutirá el uso de metáforas en el diseño de metaheurísticas y una clasificación. Aunque no se incluye una descripción de las metaheurísticas mencionadas, se pueden consultar en [6] y [9].

3.1 Metaheurística, heurística e *hyper*-heurística

Las heurísticas son técnicas basadas en la experiencia para la resolución de problemas; involucran el arte de descubrir nuevas estrategias [6] a base de prueba y error, conocimiento previo, reglas de dedo, una suposición educada, intuición e incluso sentido común.

Las heurísticas para problemas de optimización pueden ser constructivas o iterativas: generando una solución agregando componentes de forma glotona o mejorando una solución, explotando características del problema hasta llegar a un óptimo. El diseño de

un algoritmo heurístico es considerado un arte pues la eficacia de éste dependerá mucho de la experiencia y conocimiento del problema.

Una gran desventaja de estos algoritmos es que pueden quedar atrapados en óptimos locales, por lo que se desarrollaron versiones más generales con técnicas para escapar de óptimos locales y que también pudieran ser usados en otros problemas. A estas nuevas versiones se les conocía como "heurísticas modernas" [32], a las que Glover llamó por primera vez *metaheurísticas* en 1986 [11].

La palabra *metaheurística* se deriva de la composición de dos palabras: *heurística* —del griego *heuriskein*— y el prefijo *meta* que significa "más allá", "de alto nivel". Aunque no existe una definición oficial, en la literatura podemos encontrar algunas opciones:

> Una metaheurística se define formalmente como un proceso iterativo generador que guía una heurística subordinada combinando inteligentemente diferentes conceptos de exploración y explotación del espacio de búsqueda, usando estrategias de aprendizaje para estructurar información con el objetivo de encontrar eficientemente soluciones subóptimas.
>
> — Osman y col. [23].

> Las metaheurísticas típicamente son estrategias de alto nivel que guían una heurística subyacente específica del problema, para mejorar su rendimiento. El objetivo principal es evitar las desventajas de una mejora iterativa permitiendo escapar de óptimos locales.
>
> — Stützle [34].

> Una metaheurística es un conjunto de conceptos que pueden ser usados para definir métodos heurísticos que pueden ser aplicados a un amplio conjunto de problemas. En otras palabras, una metaheurística puede ser vista como una estructura general para algoritmos que puede ser aplicada a diferentes problemas de optimización con relativamente pocas modificaciones para adaptarla a un problema en específico.
>
> — Metaheuristics Network [17].

> Las metaheurísticas son métodos que orquestran una interacción entre procedimientos de mejora local y estrategias de alto nivel para crear un proceso capaz de escapar de óptimos locales y realizar una exploración del espacio de búsqueda de forma robusta.
>
> — Gendreau y col. [9].

Por lo anterior, consideramos a las *metaheurísticas* como una estructura algorítmica de alto nivel para explorar el espacio de búsqueda de forma eficiente y eficaz, son un conjunto de ideas, conceptos y operadores usados para el diseño de un algoritmo aproximado cuyas estrategias dependerán del problema a resolver. Es importante resaltar que aunque en teoría esta estructura de alto nivel puede aplicarse a cualquier problema de optimización, la eficiencia del algoritmo dependerá de explotar el conocimiento del problema, elegir las estrategias adecuadas y combinarlo de la mejor manera.

Es difícil dar una distinción entre heurísticas y metaheurísticas pues están fuertemente relacionadas y en la literatura no existe una convención. A su vez, el término *metaheurística* ha sido usado tanto para el marco algorítmico como para una implementación en específico [31]. En este trabajo nos referiremos como heurística a una estrategia específica para un problema y como metaheurística al marco algorítmico y la implementación.

En la literatura es común encontrar el término "metaheurísticas híbridas", que mezclan estrategias de otros métodos de optimización con las metaheurísticas, con el objetivo de mejorar la eficiencia de la búsqueda. Sin embargo, dada la tendencia de considerar a las metaheurísticas como el marco algorítmico, el término va desapareciendo pues este marco permite el uso de cualquier estrategia para explorar el espacio de búsqueda [31].

Las *hyper*-heurísticas fueron mencionadas en 1997 como heurísticas para elegir otras heurísticas: una metodología automatizada para seleccionar o generar heurísticas para resolver problemas de optimización. La idea de automatizar el diseño de heurísticas existe desde 1960, la tendencia actual es intentar generar nuevas heurísticas adecuadas a un problema o clase de problemas [9].

3.2 Historia

Mucho ha cambiado desde la primera vez que el término *metaheurística* fue mencionado por primera vez. Sörensen y col. [32] proponen cinco periodos en la historia de las metaheurísticas descritos a continuación.

3.2.1 Periodo Pre-teórico (hasta 1940)

La mente humana resuelve problemas –que pueden ser modelados como problemas de optimización– heurísticamente, el cerebro produce una solución sin garantizar que sea la mejor (cuando el humano cazaba era más importante hacerlo rápido que de forma óptima, pues la presa podía escapar). Tiene la capacidad de usar estrategias heurísticas: cuando nos enfrentamos a un problema nuevo, lo primero que intentamos es buscar problemas similares que hemos resuelto en el pasado, usando la experiencia.

3.2.2 Periodo temprano (1940 – 1980)

El concepto de optimización con métodos heurísticos apareció en la literatura a inicios de 1940, para finales de 1960 ya era un concepto común en las Ciencias de la Computación [28]. El inicio de la Investigación de Operaciones marcó la época en la que la gente empezó a pensar en principios generales de resolución de problemas.

En 1945 Polya publicó su libro *Cómo resolverlo* (*How to solve it*) en el cual se enuncian estrategias para resolver problemas como analogía, inducción y uso de problemas auxiliares. Polya propone buscar problemas similares para los que ya existan técnicas y adaptarlas, encontrar una generalización o un subproblema para el que ya existan métodos [31]. Aunque estas estrategias no resuelven ningún problema ni pueden ser llamadas algoritmos, son estrategias de alto nivel que siguen siendo usadas hoy en día.

Varias ideas surgieron en esta época como los algoritmos constructivos y glotones y el método de aproximación de *Vogel*. La noción de estrategias generales para la resolución de problemas de optimización llevó a las heurísticas al siguiente periodo.

3.2.3 Periodo centrado en métodos (1980 – 2000)

En este periodo las metaheurísticas despegan. Una línea de investigación cobró vida: la evolución. Aunque el propósito era estudiar la evolución y no problemas de optimización, hubo investigadores que desarrollaron algoritmos inspirados en la evolución (llamados algoritmos evolutivos) para optimización y aprendizaje de máquina. Muchos artículos fueron publicados durante este periodo con nuevos métodos basados en metáforas.

La búsqueda de una heurística genérica para la optimización comenzó. Se buscaban métodos que pudieran resolver cualquier problema eficientemente sin requerir información específica del problema, llamados métodos de *caja negra*. Se creía intuitivamente que existía un algoritmo universal de búsqueda y mucha gente dedicó esfuerzo en diseñar tal algoritmo. Sin embargo, en 1997 surgen los teoremas de que no hay almuerzo gratis (*No-free lunch theorems*) para la optimización, que establecen que ningún algoritmo de búsqueda es mejor que otro en encontrar un valor extremo de una función cuando son promediados sobre el conjunto de todas las posibles funciones; es decir, todos los algoritmos tienen la misma eficiencia cuando son evaluados sobre el conjunto de todas las funciones [6].

En 1995 la investigación en metaheurísticas había crecido tanto que tuvo lugar su propia serie de conferencias: *Metaheuristics International Conference*.

3.2.4 Periodo centrado en un marco algorítmico (2000 – actual)

En este periodo la noción sobre las metaheurísticas crece, son descritas como estructuras y no como métodos. La popularidad de las metaheurísticas híbridas crece al inicio de los

2000, pues los investigadores solían restringirse al uso de una sola metaheurística; algunas combinaciones fueron más populares que otras, como las *mateheurísticas* (*matheuristics*).

Actualmente se desarrollan algoritmos heurísticos usando la experiencia y conocimiento sobre los métodos que funcionan para cierto tipo de problemas y sobre cuáles probablemente no. Se han publicado algoritmos que combinan operadores eficientes de otros ya existentes, cuidadosamente modificados para resolver un problema en específico.

3.2.5 Periodo científico (futuro)

Se propone este periodo donde las metaheurísticas se vuelvan ciencia en lugar de arte, pues ha sido difícil que se tomen en serio [31], ya que su desarrollo es guiado principalmente por experiencia y no teoría. Para resolver problemas de optimización de la vida real, las heurísticas siguen y probablemente seguirán siendo el único método viable, por lo que se propone una mejora en los protocolos de investigación.

Cuando un nuevo algoritmo es publicado, se hace énfasis en su eficiencia –en especial si obtiene mejores resultados que algoritmos existentes– pero no se demuestra si un componente nuevo introdujo la mejora, si fue una configuración de parámetros o algún *truco* en la implementación, por lo que la investigación se vuelve una competencia [31]. La investigación debería tener mayor calidad, buscar qué métodos funcionan mejor para ciertos problemas, especialmente si se encuentra una explicación de por qué funcionan tan bien.

3.3 El uso de metáforas

Durante muchos años hubo publicaciones que presentaban un método "novedoso" para optimización, basado en una métafora sobre un proceso que muchas veces no tenía relación a la optimización: el salto de ranas, la refracción de luz, el flujo del agua en el mar, una orquestra tocando, el movimiento de galaxias, imperios, murciélagos, aves, hormigas, abejas, moscas y prácticamente cualquier otra especie de insectos [31].

El Recocido Simulado (*Simulated Annealing*) fue una de las primeras metaheurísticas basadas en metáforas que simula el proceso de recocido del acero, el cual consiste en calentar y luego enfriar lentamente el material para variar sus propiedades físicas. La analogía conlleva disminuir el costo de una solución con una serie de *movimientos* bajo cierta probabilidad que disminuye con el tiempo (temperatura); la idea de permitir movimientos que pudieran empeorar una solución fue *novedoso*. Glover cuestiona el uso de la metáfora [11] en cuanto a si un umbral flexible (*adaptive threshold*) obtendría mejores resultados que usar el parámetro de temperatura.

Seguramente la metáfora más exitosa fue la naturaleza: algoritmos evolutivos, Redes Neuronales, Autómatas Celulares, Computación Cuántica, etcétera. Los algoritmos evolutivos se basan en el principio de la supervivencia del más apto: exploran soluciones sobre el

tiempo y las de mejor calidad se preservan [6]. Metaheurísticas basadas en la naturaleza incluyen los Algoritmos Genéticos, Colonia de Hormigas y Enjambre de Partículas, además de existir muchas más basadas en procesos físicos, químicos, comportamiento social y fenómenos biológicos.

En la segunda mitad de 1990 se volvió claro que las metaheurísticas basadas en metáforas no necesariamente darían buenos resultados. Metaheurísticas que explotan las características del problema son superiores a estos métodos de caja negra – aunque esto no detuvo el creciente número de publicaciones basadas en metáforas, incluso cada vez más exóticas como gotas de agua, canto de los pájaros y la armonía musical [31].

La historia de la ciencia está llena de descubrimientos basados en el poder de las metáforas. Una buena metáfora es simple, se relaciona bien con otras metáforas y explica la evidencia [18]; nos permiten abstraer conceptos y entender una idea en términos de otra, pero si la llevamos muy lejos o en la dirección incorrecta puede engañarnos.

El problema con las metaheurísticas basadas en metáforas es que suelen ofuscar la terminología – por lo que características en común con otras metaheurísticas ya existentes pasan desapercibidas– y distraen de los métodos que realmente son innovadores; un ejemplo es la metaheurística de Armonía Musical (*Harmony Search*), que resultó ser un caso particular de estrategias evolutivas [31]. El punto de interés de cualquier metaheurística yace en los pasos computacionales que sigue (comparado con otros métodos en la literatura) y el tipo de búsqueda que esto ocasiona, dejando de lado la belleza de la metáfora en que se basa [29].

Nuevas metaheurísticas deberían usar la terminología de optimización, no de la metáfora; y proveer la relación entre las estrategias usadas con la estructura del problema, pues como se ha mencionado, explotar las características del problema es lo que hará que la implementación sea eficiente.

3.4 Clasificación

Hay muchas formas de clasificar los algoritmos metaheurísticos, dependiendo las características que se tomen en cuenta. Blum y col. [2] proponen la siguiente clasificación:

- **Inspiración** – De acuerdo al origen del algoritmo, pueden clasificarse en metaheurísticas inspiradas o no por la naturaleza. Ejemplos de algoritmos basados en la naturaleza son Algoritmos Genéticos y Colonia de Hormigas; y no inspirados en naturaleza son Búsqueda Tabú y Búsqueda Local Iterada. Esta clasificación no es muy relevante pues uno puede cuestionar la inspiración de un componente en particular, haciendo que un algoritmo pertenezca a ambas clases.
- **Función objetivo** – De acuerdo al uso que se le da a la función objetivo, se clasifican en dinámicas y estáticas; las primeras modifican la función objetivo durante la

búsqueda como Búsqueda Local Guiada y las segundas usan la misma función durante la búsqueda.

- **Número de vecindarios** – Con uno o más vecindarios. La mayoría de los algoritmos metaheurísticos usan un solo vecindario, mientras que otras como la Búsqueda con Vecindad Variable (*Variable Neighborhood Search*) usa un conjunto de vecindarios que le da la posibilidad de diversificar la búsqueda intercambiando de vecindario.

- **Uso de memoria** – Se clasifican en metaheuristícas con y sin memoria. Las segundas realizan un proceso de Markov[1], pues basan el siguiente movimiento únicamente en el estado actual de la búsqueda; cuando se usa memoria diferenciamos en dos: uso a corto y largo plazo; el primero involucra mantener registro de las decisiones tomadas recientemente y el segundo involucra parámetros de la búsqueda. El término *programación con memoria adaptativa* ha sido usado para referirse a algoritmos que usan algún tipo de memoria [34].

- **Número de soluciones** – De acuerdo al número de soluciones consideradas al mismo tiempo se clasifican en poblacionales (más de una solución) y de trayectoria. Las metaheurísticas poblacionales consideran un conjunto de soluciones que comparten información de la búsqueda. Algoritmos Genéticos y Colonia de Hormigas son ejemplos clásicos de esta categoría. En las metaheurísticas de trayectoria se realiza una exploración del espacio de búsqueda intercambiando una solución por otra en su vecindario, son llamadas así pues el proceso describe una trayectoria en el espacio de búsqueda. Recocido Simulado, Búsqueda Tabú y Búsqueda de Vecindad Variable son los algoritmos más famosos de esta categoría.

En lo que resta de este capítulo se analizarán los componentes de diseño de una metaheurística, haciendo diferencia entre las metaheurísticas de trayectoria y poblacionales.

3.5 Intensificación y Diversificación

Una metaheurística se diseña incluyendo estrategias para intensificación y diversificación[2]. Intensificación se refiere a los mecanismos y parámetros usados para enfocar la búsqueda en regiones prometedoras; la diversificación comprende las estrategias usadas para explorar regiones nuevas, buscando escapar de óptimos locales. Mientras más uso se dé a la función objetivo, mayor la intensificación; si se usa algún factor aleatorio en la búsqueda o alguna otra función, la diversificación aumenta.

[1] Proceso estocástico en donde las probabilidades del siguiente estado dependen solo del estado actual o el estado anterior inmediato, y no del recorrido que llevó al estado actual.

[2] También conocidos como explotación y exploración, aunque son usados con un significado más restrictivo [2].

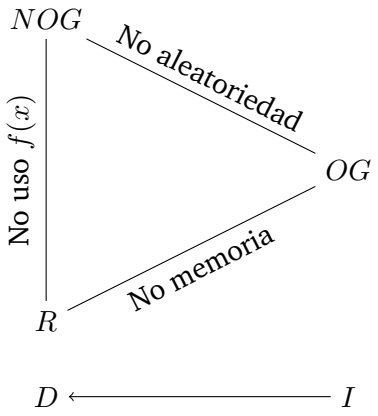


Figura 3.1 – Marco $I\&D$. Todos los componentes I&D se pueden ubicar en el triángulo: a mayor uso de la función objetivo, mayor será la intensificación; si se incluye otra función o un factor de aleatoriedad aumenta la diversificación.

El balance de estas estrategias es crucial en el diseño de una metaheurística: si tiene mucha intensificación y poca diversificación, la búsqueda se enfoca en una sola región del espacio y se corre el riesgo de quedar atrapada en un óptimo local; a su vez, si tiene poca intensificación y mucha diversificación, la búsqueda se acerca a ser aleatoria. Una metaheurística será exitosa para un problema de optimización si puede dar un balance entre intensificación y diversificación de manera óptima. La búsqueda debe realizarse de forma inteligente para explorar regiones con soluciones de buena calidad y moverse a regiones no exploradas cuando sea necesario.

La intensificación y diversificación suelen verse como fuerzas opuestas, sin embargo, en su definición original se menciona que no lo son, pues cada una tiene características de la otra. Blum y col. [2] definen a un componente $I\&D$ como cualquier función o componente algorítmico que tiene un efecto de intensificación o diversificación en la búsqueda. Ejemplos de estos componentes son los operadores de búsqueda, factores de probabilidad, uso de listas tabú o cambios en la función objetivo; es decir, un componente $I\&D$ es un operador, acción o estrategia de algoritmos metaheurísticos.

El espacio de todos los componentes $I\&D$ se describe como un triángulo cuyas esquinas corresponden a componentes extremos (figura 3.1): **OG** corresponde a los componentes basados en la función objetivo, **NOG** corresponde a los componentes basados en una o más funciones que no son la función objetivo y **R** corresponde a los componentes que usan aleatoriedad. La esquina OG corresponde a componentes con máxima intensificación y mínima diversificación, mientras que las esquinas NOG-R corresponden a componentes con máxima diversificación y mínima intensificación. Se propone este marco como una vista unificada de intensificación y diversificación que permita analizar distintos algoritmos metaheurísticos de acuerdo a su diseño. Algunos ejemplos de componentes $I\&D$ son [2]:

- En Búsqueda Tabú se usan listas para evitar ciertas soluciones que ya han sido consideradas o que no cumplen alguna regla; la restricción de las soluciones que se consideran tiene un efecto en la diversificación mientras que la elección de la mejor solución que no esté en las listas tiene un efecto de intensificación. Este componente puede ubicarse entre OG y NOG.

- En Recocido Simulado se tiene un criterio de aceptación de acuerdo a la temperatura, esto involucra la función objetivo y un factor aleatorio; si la temperatura es baja, la diversificación disminuye y la intensificación aumenta. Este componente puede ubicarse entre las tres esquinas.

- En la Colonia de Hormigas la actualización del valor de las feromonas altera la distribución de probabilidad usada para explorar el espacio de búsqueda; este componente se centra en la intensificación, y la diversificación depende de la forma en que se actualizan las feromonas. Este componente se ubica entre OG y NOG.

- En la Búsqueda Local Iterada se selecciona una solución aleatoria dentro del vecindario. Aunque pareciera que solo tiene un efecto de diversificación, al usar un vecindario, la función objetivo está implícitamente usada; por lo que tiene un efecto de intensificación al seleccionar una solución que no es muy diferente a la actual (mientras no sea un vecindario aleatorio).

- En la búsqueda de vecindad variable, se opta por alternar de vecindario cuando la búsqueda se atora en una región, lo cual introduce un efecto de diversificación.

Probablemente la mayoría de los componentes $I\&D$ básicos que se usan en los algoritmos metaheurísticos tienen un efecto de intensificación y diversificación a la vez [2]. El balance adecuado es necesario para que un algoritmo metaheurístico sea eficiente, la manera en que estos componentes actúan dependerá del *horizonte de optimización*: estrategias a corto plazo tienen un fuerte efecto de intensificación, si el horizonte aumenta, estrategias de diversificación entran en acción [2]. La manera más simple de coordinar la intensificación y diversificación es agregar un mecanismo de reinicio, así cada vez que la búsqueda se *atore*, se puede iniciar en otra región del espacio. Otras estrategias más avanzadas son Oscilación Estratégica (*Strategic Oscillation*) y Cadenas de Eyección (*Ejection Chain*) [2].

Las metaheurísticas poblacionales mantienen la diversificación cuidando la similitud de las soluciones actuales: si éstas se vuelven muy similares muy rápido se puede activar un mecanismo para corregirlo, esto involucra un factor aleatorio (introduciendo soluciones nuevas de manera aleatoria) o uso de memoria (evitando introducir soluciones muy similares o que ya han sido consideradas) [7]; la intensificación está en el uso de operadores para reemplazar soluciones por mejores en el vecindario. En las metaheurísticas de

trayectoria, la intensificación involucra estrategias de búsqueda local, guiando la búsqueda hacia regiones prometedoras; la diversificación se centra en permitir movimientos hacia regiones no exploradas generando soluciones nuevas o permitiendo introducir componentes nuevos.

3.6 Elementos de diseño

Podemos identificar algunos principios de diseño que surgen al adecuar el modelo de un problema de optimización en la implementación de un algoritmo metaheurístico: la representación de las soluciones (en relación a las variables de decisión), los operadores de búsqueda (a partir del vecindario) y la función objetivo.

Un análisis del paisaje adaptativo del problema de optimización dará una aproximación de cómo la representación, operadores de búsqueda y función objetivo se comportarán durante la búsqueda [36]. El análisis permite identificar los componentes $I\&D$ adecuados, por ejemplo, si tenemos un paisaje muy plano se debe favorecer la construcción de soluciones más que a la exploración del vecindario [7].

3.6.1 Representación

La representación de una solución es una pregunta fundamental en las metaheurísticas [7], pues influirá en cómo se evalúa la función objetivo y los operadores de búsqueda a usar. La elección de la representación debe seguir los principios de inclusión, conectividad y eficiencia:

1. **Inclusión**. Todas las soluciones asociadas al problema de optimización deben poder representarse, es decir, el espacio de búsqueda generado por la representación debe incluir todas las soluciones factibles del problema. Algunas representaciones crean espacios de búsqueda más sencillos sacrificando el principio de inclusión, lo cual aunque reduce la complejidad de la búsqueda, puede dejar fuera soluciones de alta calidad.

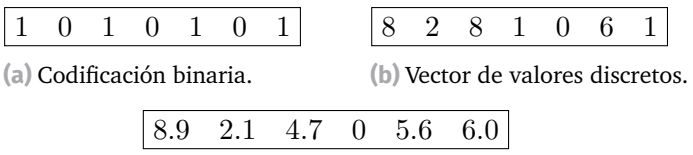
2. **Conectividad**. Debe existir un camino entre cualesquiera dos soluciones, es decir, se debe poder obtener cualquier solución a través de una serie de modificaciones de otra. Esto incluye permitir movimientos a soluciones no factibles, pues puede ser la única forma de llegar a una solución óptima.

3. **Eficiencia**. Debe ser fácil obtener una solución y transformarla a otra. La representación debe ser fácil de manipular para disminuir la complejidad de la función objetivo y los operadores de búsqueda. En general, la representación de una solución puede absorber algunas restricciones haciendo que la evaluación sea más fácil.

3.6.1.1 Representaciones lineales

Pueden ser vistas como cadenas sobre un alfabeto. También se conocen como representaciones *indirectas* o codificadas, pues se requiere una función que mapee la representación a una solución del problema. A continuación se describen las más comunes (figura 3.2):

- **Binarias**. Vector de valores binarios: $\{0,1\}^l$. Usadas para problemas cuyas variables de decisión denotan la presencia o ausencia de un elemento [36] como el problema de la mochila y el problema de satisfacibilidad.
- **Enteras**. Vector de valores enteros: $\mathbb{Z}^l$. Son una generalización de los vectores binarios, donde cada variable toma un valor sobre $\mathbb{Z}$. Estas representaciones son usadas en problemas de optimización combinatoria como el problema de asignación.
- **Reales**. Vector de valores reales: $\mathbb{R}^l$. Usados en problemas de optimización continuos y problemas de parametrización.
- **Permutaciones**. Muchos problemas de planeación y enrutamiento son considerados problemas de permutación, pues una solución puede verse como una permutación π de un conjunto σ.



(a) Codificación binaria. **(b)** Vector de valores discretos.

(c) Vector de valores reales.

Figura 3.2 – Algunas representaciones lineales comunes.

La función que mapea una solución a una representación tiene distintas posibilidades:

- **Uno a uno**. Una solución es codificada en una sola representación y cada codificación representa una única solución.
- **Uno a muchos**. Una solución puede ser representada por varias codificaciones, esta redundancia incrementará el tamaño del espacio de búsqueda y podría afectar la eficiencia de la metaheurística. Por ejemplo, en problemas de agrupación dos soluciones podrían representar los mismos grupos.
- **Muchos a uno**. Una codificación representa varias soluciones, si parte de la información de las soluciones no está propiamente representada; como consecuencia se tiene un espacio de búsqueda más pequeño, lo cual puede mejorar la eficiencia de la metaheurística pero también influirá en la calidad de la solución encontrada.

3.6.1.2 Representaciones no lineales

Son representaciones *directas* relacionadas a estructuras más complejas, en general son gráficas, conjuntos o árboles. Estas representaciones no requieren una función para transformarse a una solución pues son más apegadas al modelo, y por ende, dependientes del problema.

Las gráficas son usadas en problemas de redes, circuitos o donde existen relaciones sobre elementos; si usamos una lista de adyacencias pueden verse como representaciones lineales. Los conjuntos son usados en problemas de agrupación y selección, teniendo longitud variable. Los árboles son usados en problemas que involucran jerarquía y, en algunos casos, pueden codificarse con una representación lineal.

3.6.2 Función objetivo

La función objetivo guiará la búsqueda hacia soluciones buenas del espacio de búsqueda, si ésta está mal definida puede llevar a soluciones no factibles o de baja calidad sin importar los componentes usados [36].

Ya que la función objetivo tiene un rol importante en el paisaje adaptativo, si ésta genera mesetas será difícil escapar de óptimos locales. Para esto, se puede agregar un componente que discrimine dos soluciones que tengan el mismo costo en la versión original [12], es decir, alisar (*smoothing*) el paisaje (figura 3.3). Para algunos problemas, agregar más información a la función objetivo resulta en una mejora significativa en la calidad de las soluciones encontradas [36].

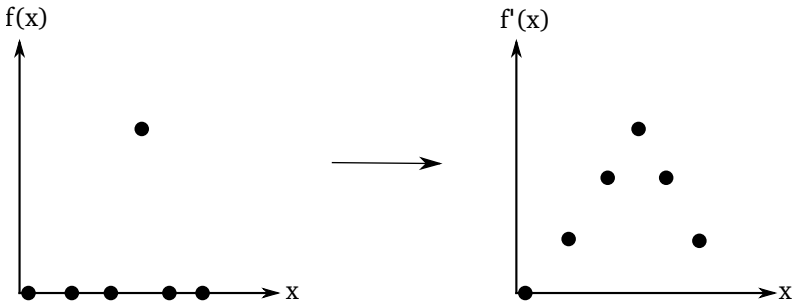


Figura 3.3 – Representación del paisaje adaptativo en una dimensión – modificación de la función objetivo para mejorar el paisaje.

Otro factor importante al adaptar la función objetivo es que ésta debe poder calcularse de forma eficiente, de no ser posible, se puede hacer uso de una función sustituta que aproxime la función objetivo.

3.6.3 Operadores de búsqueda

Una solución s' vecina de s ($s' \in N(s)$) es obtenida aplicando un operador de búsqueda que modifica a la solución s. La forma en que obtenemos una solución vecina depende

de la representación usada y el vecindario elegido para el problema de optimización. Los operadores de búsqueda se basan en la intuición de que si tenemos una buena solución, es muy probable que encontremos una mejor cercana a ella; es decir, soluciones buenas tienen una estructura en común. La eficiencia de estos operadores dependerá del paisaje adaptativo, por lo que la elección del vecindario es un factor determinante.

Podemos dar la estructura de un vecindario de distintas maneras. El vecindario trivial es hacer que todas las soluciones sean vecinas; sin embargo, esto hace que buscar en el vecindario de una solución sea tan difícil como el problema original. Es importante definir el vecindario de tal forma que encontrar un óptimo local pueda realizarse en poco tiempo [12], haciendo una pequeña modificación usando una métrica, generando soluciones *cercanas* para no dar *brincos* en el espacio de búsqueda.

Podemos clasificar los operadores de búsqueda en dos: operadores de búsqueda local y operadores de recombinación:

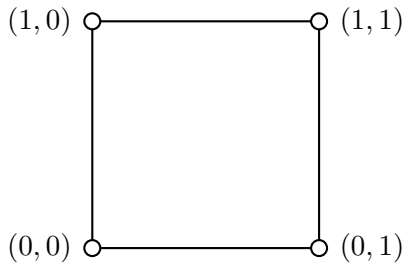
- **Operadores de búsqueda local**. Estos operadores dependen del paisaje adaptativo, pues su objetivo es guiar la búsqueda a soluciones de mejor calidad dentro del vecindario. En las representaciones lineales, la distancia de Hamming (figura 3.4a) es a menudo usada pues da una relación útil entre soluciones; para obtener soluciones vecinas se puede cambiar el valor i del vector (figura 3.4b): cambiar 0 a 1, incrementar o decrementar un valor discreto, cambiar un valor real bajo cierta distribución; aplicar una máscara o intercambiar valores. Para el caso de las permutaciones se debe tener cuidado, pues la alteración de una solución puede resultar en otra que no sea una permutación.

- **Operadores de recombinación**. Estos operadores se basan en la descomponibilidad del problema: solucionar problemas más pequeños y combinarlos, por lo que si la mezcla de dos soluciones no resulta en una mejor, no son muy eficientes. La combinación de dos soluciones x, y debe satisfacer [28]

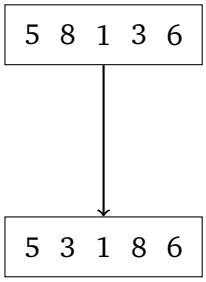
  $$d(x, y) \geq \text{máx}(d(x, z), d(y, z))$$

  donde z es la solución resultante; esto indica que la solución obtenida debe ser similar a las soluciones usadas para generarla. Para las representaciones lineales se tienen operadores estándar (figura 3.4c): *crossover* de n puntos, uniforme, aritmético y geométrico [36].

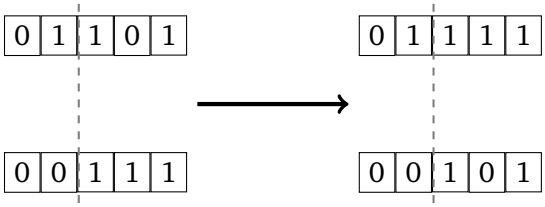
En una metaheurística poblacional se debe mantener la diversidad de las soluciones, de otra forma se comporta como una metaheurística de trayectoria al no introducir componentes nuevos al mezclar dos soluciones, por lo que operadores que modifiquen una solución o estrategias que prohíban un conjunto de soluciones muy similares son necesarias.



(a) Vecindario de una representación binaria.



(b) Inversión de la subsecuencia $8, 1, 3$.



(c) *crossover* de un punto, se obtienen dos soluciones nuevas mezclando fragmentos de otras dos.

Figura 3.4 – Ejemplos de operadores de búsqueda para representaciones lineales. a) Operador de búsqueda local que cambia un valor binario en cierta posición. b) Operador de búsqueda local que invierte una subsecuencia de un vector discreto. c) Operador de recombinación.

Para las representaciones no lineales, los operadores de búsqueda deben respetar la métrica sobre el espacio de búsqueda y se definen de acuerdo al problema.

3.6.4 Manejo de restricciones

En algunas ocasiones, encontrar una solución factible a un problema de optimización es un problema *NP*–duro [7], por lo que generar una solución factible es una tarea difícil. En estos casos necesitamos incluir estrategias que lidien con las soluciones no factibles.

3.6.4.1 Rechazo

Es la estrategia más simple, donde una solución no factible se descarta. Este acercamiento se recomienda cuando la región no factible es pequeña pues no explota información de las soluciones descartadas.

3.6.4.2 Penalización

La idea de esta estrategia es transformar un problema quitando sus restricciones al agregar un factor de penalización en relación a las restricciones violadas. Una nueva función objetivo f' se define como

$$f'(s) = f(s) + \rho(s),$$

donde f es la función objetivo original y $\rho(s)$ es una función que cuantifica la infactibilidad de la solución s. Si el factor de penalización es muy alto –en un problema de minimización– la búsqueda evitará soluciones no factibles y, en el caso de regiones disjuntas, no se visitarán regiones factibles, por lo que la búsqueda puede converger a una solución no óptima; por otro lado, si la penalización es muy baja, se perderá tiempo explorando soluciones no factibles e incluso puede resultar en una solución no factible.

Existen al menos tres alternativas para definir la penalización [4]:

- Usar una penalización estática, sin importar qué tan mala sea la solución:

  $$\rho(s) = K,$$

  con K lo suficientemente grande para desviar la búsqueda de regiones no factibles. Esta alternativa no distingue entre soluciones no factibles y por ende, no explota información de soluciones que puedan estar cerca de una región factible.

- Cuantificar la inviabilidad de la solución, dependiendo del número de restricciones que no cumpla:

  $$\rho(s) = \sum_{i=1}^{m} w_i \alpha_i,$$

  donde m es el número de restricciones, a_i es un valor booleano que indica si la restricción i es violada y w_i es la penalización asociada a la restricción i.

- Cuantificar el costo de reparar la solución, incluyendo la distancia a una región factible:

  $$\rho(s) = \sum_{i=1}^{m} w_i d_i^k,$$

  donde d_i es la distancia a una región factible para la restricción i y k es un parámetro de búsqueda.

Coello [4] lista las siguientes pautas para calcular la penalización:

- Funciones de penalización que midan la distancia hacia una solución factible son más eficientes que aquellas que están en función del número de restricciones violadas.
- Para problemas con pocas restricciones y pocas soluciones factibles, penalizaciones basadas en el número de restricciones violadas no suelen generar buenas soluciones.
- Las funciones de penalización pueden ser construidas eficientemente a partir de la distancia máxima/esperada a una solución factible.

- Entre más precisa sea la penalización, mejores serán las soluciones encontradas.

Sin embargo, estas pautas pueden ser difíciles de seguir debido a que la distancia de una solución a la región factible muchas veces no puede ser estimada con precisión.

Al incorporar el factor de penalización a la función objetivo se tienen tres alternativas:

- **Penalización estática**. El factor de penalización no depende del estado de la búsqueda y permanece constante durante ella. En general, esta estrategia requiere más parámetros para determinar el factor de penalización adecuado.
- **Penalización dinámica**. El factor de penalización incrementa durante la búsqueda, teniendo un efecto fuerte de diversificación al inicio, que se convierte en intensificación al final de la búsqueda [4] [36].
- **Penalización adaptativa**. El factor de penalización depende de la información acumulada de la búsqueda para mejorar la eficiencia, en esta versión la magnitud de los coeficientes w_i se actualiza dependiendo de las soluciones encontradas hasta el momento; por ejemplo, puede disminuir si se han encontrado soluciones factibles y aumentar si la búsqueda pierde mucho tiempo en regiones no factibles.

3.6.4.3 Reparación

Estas estrategias consisten en algoritmos que transformen una solución no factible a una que sí lo sea. Son usadas cuando los operadores de búsqueda pueden generar soluciones que no sean factibles. En general estas estrategias son glotonas, buscando transformar una solución a otra con el menor costo posible y modificándola lo menos que se pueda. La eficiencia de esta estrategia dependerá de la existencia de un algoritmo de reparación.

3.6.4.4 Preservación

Estas estrategias se centran en los operadores de búsqueda, haciendo que dada una solución factible solo se permitan movimientos a otra solución factible. En el caso de las permutaciones, por ejemplo, se puede usar una codificación de llave aleatoria [36] que garantice la transformación de una permutación a otra. Ya que para algunos problemas encontrar soluciones factibles es muy difícil, la implementación de estas estrategias dependerá de la representación y operadores elegidos.

3.6.5 Parámetros

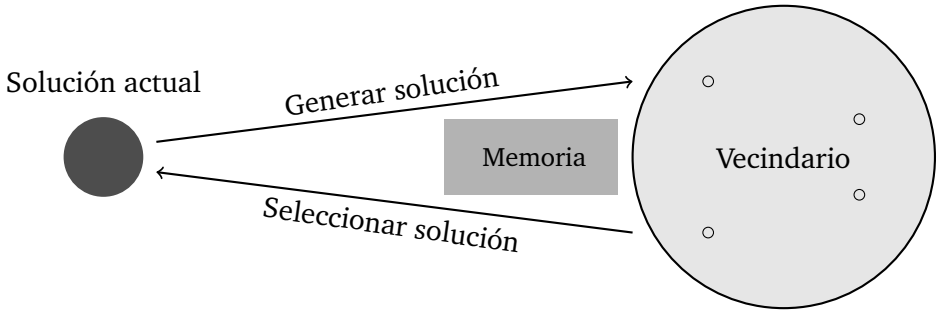
Un parámetro de búsqueda es un valor de entrada al procedimiento que es controlado por el usuario y que tiene un efecto en el comportamiento de la búsqueda, y por ende, en su eficiencia. Una configuración de parámetros, por más minuciosa que sea, no compensará una mala definición de las soluciones, vecindario o función objetivo [7]. La configuración

de valores para los parámetros que puede tener una metaheurística es una tarea laboriosa, pues se requiere de una cuidadosa inicialización ya que los valores ideales dependen de la instancia a resolver y del tiempo de ejecución esperado.

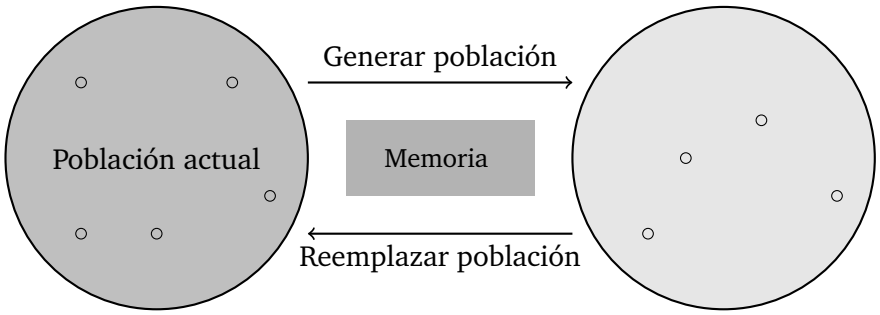
Existen dos estrategias para configurar los parámetros de búsqueda: *online* y *offline*; en la primera, los parámetros se controlan y actualizan dinámicamente durante la ejecución mientras que en la segunda, los valores se establecen de antemano. Las metaheurísticas que configuran sus parámetros automáticamente durante la búsqueda se conocen como auto-adaptativas.

3.7 Estructura general

Las metaheurísticas comparten una estructura similar (figura 3.5): inician generando una o más soluciones, a partir de las cuáles se itera la búsqueda hasta alcanzar una condición de paro. En cada iteración se exploran nuevas soluciones en el vecindario actual, probablemente usando memoria, y posteriormente se selecciona la o las soluciones que conformarán el nuevo estado. Este proceso lo podemos abstraer en tres etapas: generación de soluciones iniciales, estrategias de movimiento (exploración y selección de nuevas soluciones) y la evaluación de la condición de paro.



(a) Metaheurística de trayectoria – A partir de la solución actual se explora el vecindario y se reemplaza por otra de este conjunto. Tanto la exploración como selección pueden hacer uso de memoria.



(b) Metaheurísticas poblacionales – Dado un conjunto de soluciones se usan operadores de recombinación para generar un nuevo conjunto que reemplace el actual.

Figura 3.5 – Estructura de una metaheurística [36].

3.7.1 Soluciones iniciales

La forma en que las soluciones iniciales –el punto de partida de la búsqueda– son obtenidas es una decisión importante. Se tienen dos alternativas: soluciones aleatorias y soluciones construidas por algoritmos glotones. En el caso de las metaheurísticas de trayectoria, se suele iniciar de una solución generada por una estrategia glotona, pues se tiene la idea de que al partir de una buena solución se llegará a una mejor –aunque como se ha discutido, esto dependerá de las características del paisaje adaptativo; en las metaheurísticas poblacionales se suele iniciar con soluciones aleatorias para mantener la diversidad de la población.

Si no se tiene mayor conocimiento del problema o de las propiedades que comparten las soluciones de alta calidad, es recomendable generar las soluciones de manera aleatoria bajo una distribución uniforme sobre el espacio de búsqueda [28].

3.7.2 Estrategias de movimiento

Las estrategias de movimiento definen cómo una solución es intercambiada por otra durante el proceso de búsqueda. Los operadores de búsqueda son utilizados para generar el vecindario de una solución.

En las metaheurísticas de trayectoria se selecciona una solución del vecindario para reemplazar la solución actual. Un vecindario muy grande puede mejorar la calidad de la solución encontrada, pues en cada paso se consideran más soluciones; sin embargo, esto requiere de mayor tiempo para generar y evaluar todas las soluciones vecinas [36]. La mayoría de las metaheurísticas de trayectoria usa vecindarios pequeños y en el caso de usar uno más grande, existen estrategias heurísticas para explorarlos como cadenas de eyección (*ejection chain*), transferencia cíclica (*cyclic transfer*) y abanico secuencial (*sequential fan*). Para seleccionar una solución vecina se tienen distintas estrategias:

- **Mejora más significativa (*best improvement*)**. El mejor vecino (la solución de mayor calidad) es seleccionado, esto implica una exploración exhaustiva del vecindario.
- **Primera mejora (*first improvement*)**. Se selecciona la primera solución vecina que sea mejor a la solución actual, por lo que se realiza una exploración parcial del vecindario y en el peor caso (si ninguna solución es mejor) se realiza una exploración completa.
- **Selección aleatoria**. Una solución aleatoria es seleccionada dentro del vecindario, incluso si no mejora la solución actual.

Ya que estas estrategias tienden a quedar atrapadas en un óptimo local, se introducen estrategias de diversificación cómo un umbral de aceptación (donde la solución es aceptada

bajo cierta probabilidad), intercambiar de vecindario o modificar la función objetivo (lo cuál tiene impacto en cuál es la mejor solución del vecindario).

En las metaheurísticas poblacionales se genera una nueva población de distintas formas:

- **Basado en evolución**. La nueva población es compuesta por soluciones generadas a partir de operadores de búsqueda, tomando como base las soluciones actuales.
- **Basado en memoria**. La memoria[3] es construida a partir de las soluciones actuales y será usada para generar nuevas soluciones. Las metaheurísticas basadas en memoria se conocen como Programación con Memoria Adaptativa[4].

Una vez el nuevo conjunto de soluciones es creado, se seleccionan soluciones que conformen la nueva población. La estrategia tradicional es reemplazar por completo el conjunto de soluciones pero otras estrategias incluyen una selección *elitista*: las mejores soluciones son preservadas en la población.

3.7.3 Condición de paro

Si la estrategia de búsqueda guía hacia una mejor solución, se puede detener cuando hay varias iteraciones sin encontrar una mejora. Si la búsqueda puede continuar tanto como queramos, se suelen introducir criterios como un máximo número de movimientos o tiempo de ejecución. También se puede introducir un parámetro de búsqueda que indique cuando terminar, como en Recocido Simulado donde la búsqueda se detiene una vez se llega a cierta temperatura.

3.8 Análisis de eficiencia

La eficiencia de una metaheurística dependerá de varios factores: el tiempo de ejecución, calidad de soluciones obtenidas, robustez, escalabilidad para paralelizar, ajuste automático de parámetros, etcétera. El primer problema al querer medir la eficiencia de una metaheurística son los insumos. Para problemas teóricos se tienen bibliotecas que generan instancias estándar para problemas muy conocidos, como es el caso del PAV[5] y SAT[6]; para problemas de la vida real obtener los insumos es difícil, pues suelen ser privados, costosos de generar o por temas de confidencialidad no se pueden obtener. Alternativamente se pueden generar instancias aleatorias, pero para los problemas de la vida real éstas pueden estar muy lejos de la estructura que realmente se presenta y por ende, la efectividad de la implementación puede variar al probar con una instancia real [36].

[3] Taillard y col. [35] analizan las metaheurísticas de acuerdo al uso de memoria en cada etapa.
[4] En inglés: *Adaptive memory programming (AMP)*.
[5] http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html
[6] https://www.cs.ubc.ca/ hoos/SATLIB/benchm.html

En genenal, se recomienda dividir en dos el conjunto de instancias a usar: un subconjunto que será usado para la configuración de parámetros y un segundo para evaluar la eficiencia del algoritmo [36]. Se tienen tres indicadores principales de eficiencia:

- **Tiempo**. Para los métodos exactos la eficiencia está en términos de su complejidad en tiempo; para las metaheurísticas este no es un indicador adecuado pues no da la historia completa de la búsqueda [36]. El tiempo suele medirse en términos de uso de CPU, lo cual varía de acuerdo a la arquitectura de la computadora donde se realiza la ejecución, por lo que otra medida aceptada es el número de evaluaciones de la función objetivo.
- **Robustez**. Es la insensibilidad contra las variaciones de parámetros o instancias, entre menos variación haya, mayor robustez. El conjunto de parámetros debe ser el mismo para todas las instancias a menos de que se use un ajuste automático, pues de otra manera el tiempo necesario para calibrar los parámetros debe tomarse en cuenta para medir la eficiencia.
- **Calidad de soluciones**. Se mide la distancia o desviación de la solución encontrada a una de las siguientes opciones [36]:

    1. *Óptimo global*. Para esto es necesario saber de antemano el costo del óptimo global, lo cual puede ser desconocido en muchos problemas, aunque en algunos casos se puede obtener una estimación.
    2. *Cota superior/inferior*. Se usan cotas justas como alternativa al costo del óptimo global. En general, esta medida es más fácil de obtener.
    3. *Mejor solución conocida*. Existen múltiples recursos con instancias de algunos problemas teóricos y la mejor solución obtenida hasta el momento.
    4. *Implementación actual*. Se usa la solución obtenida por un método implementado actualmente, que sirve de referencia para medir la calidad de una solución.

Los resultados obtenidos durante la experimentación de la implementación de un algoritmo metaheurístico suelen ser analizados estadísticamente, obteniendo medidas como la mediana, el promedio, mínimo, máximo, etcétera; por ejemplo: el porcentaje de éxito, que es el número de ejecuciones exitosas entre el número de experimentos. Los resultados obtenidos suelen reportarse por medio de gráficas como una gráfica de cajas *(Box plot)* para mostrar la distribución de los resultados, o una gráfica de dispersión *(Scatter plot)* para mostrar la relación de varios indicadores como la calidad de soluciones contra tiempo o el tiempo contra robustez.

Por otra parte, la reproducibilidad de los resultados es muy importante. Un algoritmo metaheurístico debe estar bien explicado y los parámetros usados deben saberse. Existen

programas como `Paradiseo`[7] para la ejecución de metaheurísticas con el fin de mejorar su reproducibilidad, reusabilidad y extensión.

[7] http://paradiseo.gforge.inria.fr/

Asignación de grupos en la ENP 6

4

El problema de la asignación de grupos en la Escuela Nacional Preparatoria Plantel 6 "Antonio Caso" (ENP 6) consiste en asignar alumnos a grupos de acuerdo a una lista de preferencias proporcionada por ellos. Se busca una asignación balanceada, evitando grupos de *excelencia*, y donde cada alumno sea asignado a un grupo de su preferencia. Hasta hace unos años el trabajo de asignar alumnos en la ENP 6 era realizado por el personal administrativo. Actualmente se tiene un proceso donde los alumnos eligen grupos en orden de preferencia y un algoritmo asigna a los alumnos siguiendo una estrategia glotona: se ordena a los alumnos por promedio y situación escolar, en cada paso se asigna un alumno al grupo de mayor preferencia que tenga cupo para su rango de promedio, de no encontrar ninguno, se selecciona alguno de los disponibles.

La estructura de este problema es la misma que un problema de asignar empleados a proyectos, estudiantes a dormitorios, tareas a trabajadores y trabajos a máquinas [25], por lo que se formalizará usando el Problema de Asignación Generalizada.

En este capítulo se dará una descripción sobre el Problema de Asignación Generalizada y algunas de sus variantes, para formalizar la asignación de grupos como una instancia de este problema; se discutirá el modelo de la asignación de grupos, analizando la función usada para medir la calidad de ésta tomando como base la asignación de grupos en la ENP 6.

4.1 El Problema de Asignación Generalizada

Dado un conjunto de n tareas y m agentes, donde cada agente i tiene una capacidad limitada b_i y cada tarea j requiere cierta capacidad a_{ij} al ser asignada al agente i, el Problema de Asignación Generalizada (PAG) busca una asignación de tareas a exactamente un agente con el menor costo y sin exceder las capacidades. Se suele formular como un problema de programación entera:

$$\text{mín} \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{4.1}$$

cumpliendo que

$$\sum_{j=1}^{n} a_{ij}x_{ij} \leq b_i, \quad i = 1, \ldots, m, \tag{4.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad j = 1, \ldots, n, \tag{4.3}$$

donde c_{ij} es el costo de asignar la tarea j al agente i; x_{ij} son variables binarias que indican si la tarea j es asignada al agente i; y los coeficientes c_{ij}, b_i y a_{ij} son positivos. La restricción 4.2 asegura que la capacidad de cada agente sea respetada y 4.3 garantiza que una tarea sea asignada a un solo agente.

El primer problema de esta forma fue estudiado por De Maio y col. [5] en un problema de transportación, en su modelo los requerimientos de capacidad eran independientes del agente. Ross y col. [27] introdujeron el término *Problema de Asignación Generalizada* para esta clase de problemas.

4.1.1 Casos especiales y extensiones

Cuando la capacidad de todos los agentes y costos de todas las tareas es 1, el problema se reduce al problema de asignación, que puede resolverse en tiempo polinomial ([30, cap. 17]). Cuando el costo y capacidad a ocupar de las tareas no varía entre agentes, el problema se reduce al Problema de Mochilas Múltiples (*Multiple Knapsack Problem*).

En la literatura podemos encontrar extensiones de este problema, propuestas para describir con más detalle problemas de la vida real [26]:

- **PAG con recursos múltiples (PAGMR)**. Un agente consume una variedad de recursos por cada tarea asignada.
- **PAG multi-nivel (PAGM)**. Los agentes pueden realizar tareas con distinta eficiencia y las tareas se asignan a un agente en un nivel específico, lo que implica distintos costos y capacidades.
- **PAG dinámica (PAGD)**. Se considera que las tareas tienen una expiración por lo que el orden en la asignación es importante.
- **PAG de cuello de botella**. La penalización máxima de asignar una tarea a un agente es minimizada.
- **PAG estocástico**. La cantidad de recursos usados al asignar una tarea o las tareas a asignar no se conocen de antemano.
- **PAG multiobjetivo**. Las asignaciones tienen varios atributos a considerar, por ejemplo: costo y tiempo.

- **Problema de asignación multiple generalizada**. Cada tarea se asigna a r_j agentes en lugar de uno.

4.1.2 Métodos

El PAG es un problema de optimización *NP*–duro, y el problema de decisión asociado es *NP*–completo [26]. Se han propuesto tanto métodos exactos como aproximados [16] [26]; dentro de los métodos exactos, la solución óptima para el PAG es obtenida usando técnicas de ramificación y corte; para instancias muy grandes del PAG se han usado metaheurísticas como Recocido Simulado y Búsqueda Tabú, además de existir aproximaciones heurísticas y metaheurísticas híbridas.

4.2 Asignación de grupos

Durante el periodo de reinscripciones los alumnos de la ENP 6 registran grupos, en orden de su preferencia, para cursar el siguiente año escolar. Para cada año hay alrededor de 18 grupos, de los cuales se suelen elegir 5; para cada uno, los alumnos de quinto año seleccionan el idioma que desean cursar de ser asignados en él, mientras que los alumnos de sexto año seleccionan las optativas de área a cursar.

No todos los grupos ofrecen los mismos idiomas y optativas, lo cual tiene un impacto en la demanda. Si se asignara a todos los alumnos en un grupo de su preferencia se podrían generar grupos sin alumnos y otros con demasiados –lo cuál no es pedagógico–, por lo que se impone un límite de capacidad en cada grupo. Esta restricción no solo afecta el cupo de las asignaturas de tronco común, sino que también existe un cupo mínimo para los idiomas y optativas que ofrece un grupo; es decir, todos los idiomas y optativas disponibles deben tener alumnos asignados.

Además de las restricciones de capacidad, la política del plantel es tener una asignación *balanceada*: todos los grupos deben tener el mismo número de alumnos por promedio, es decir, el mismo número de alumnos con promedio entre 7 y 8, 8 y 9, etcétera. Debido a estas restricciones puede que no exista una asignación en la que todos los alumnos estén en un grupo de su preferencia.

El objetivo es encontrar una asignación justa que maximice la satisfacción de los alumnos, cumpliendo las restricciones de capacidad y balance de cada grupo. El balance en el grupo es de suma importancia, se debe buscar un grupo heterogéneo ya que "grupos con diversidad de alumnos tienden a resultar en un mejor aprendizaje" [13].

4.2.1 Trabajo relacionado

Para describir el modelo de optimización se definen las variables binarias:

$$x_{sg} = \begin{cases} 1 & \text{si el alumno } s \text{ es asignado al grupo } g, \\ 0 & \text{en otro caso,} \end{cases} \tag{4.4}$$
$$\forall s = 1, \ldots, N, \quad \forall g = 1, \ldots, G,$$

donde N es el total de alumnos y G es el total de grupos disponibles.

Se tienen las siguientes restricciones:

$$\sum_{g=1}^{G} x_{sg} = 1, \quad s = 1, \ldots, N, \tag{4.5}$$

$$\sum_{s=1}^{N} x_{sg} \leq C_g, \quad g = 1, \ldots, G, \tag{4.6}$$

la igualdad 4.5 es equivalente a 4.3 que garantiza que un alumno sea asignado a un solo grupo; ya que la capacidad b_i de un grupo está dada por el cupo del mismo y un alumno toma solo un lugar ($a_{sg} = 1$), la desigualdad 4.6 es equivalente a 4.2 que establece el límite de alumnos que pueden asignarse en un grupo ($b_i = C_g$).

Agustín Blas y col. [1] atacan el problema de asignar alumnos a laboratorios de su preferencia y proponen maximizar una función que mide la calidad de una asignación en términos de cuántas preferencias de alumnos fueron satisfechas:

$$\text{máx} \left(\sum_{s} \sum_{g} x_{sg} P_{sg} + K_2 \left(1 - \frac{\sigma}{\tau} \right) \right), \tag{4.7}$$

donde

$$\sigma = \frac{\sum_{i=1}^{G} \left(\sum_{s} x_{si} - \frac{N}{G} \right)^2}{G}, \tag{4.8}$$

$$\sum_{g=1}^{G} P_{sg} = 100, \quad s = 1, \ldots, N. \tag{4.9}$$

La preferencia del alumno s por el laboratorio g se denota P_{sg}; entre más grande sea, mayor es la preferencia por el grupo. La restricción 4.9 sirve para limitar los valores que pudiera tener P_{sg}. El valor σ definido en 4.8 es la varianza del número de alumnos en

cada grupo respecto a la media de alumnos por grupo, donde τ y K_2 son parámetros de entrada que indican la tolerancia a la varianza y el peso de la restricción de capacidad, respectivamente.

Powell y col. [25] analizan distintas funciones de costo para un problema de asignación de alumnos a grupos que maximice la diversidad, creando grupos diversos para que los alumnos "tengan la experiencia y desafío de trabajar con personas que no sean como ellos". Se considera una matriz de atributos A de variables binarias a_{ik} que indican si el alumno i exhibe el atributo k; los atributos reflejan información demográfica, experiencia o historia académica derivada de la información que se tiene registrada para cada alumno. En su modelo todos los grupos tienen la misma capacidad ($C_g = S, \forall g$).

Como parte de su estudio Powell y col. proponen varias funciones objetivo, pues el problema puede verse como maximizar diversidad entre grupos o minimizar la diferencia entre grupos:

$$(Z_1)\ \text{máx} \quad \sum_g \sum_i \sum_{j>i} d_{ij} x_{ig} x_{jg}, \tag{4.10}$$

$$(Z_2)\ \text{máx} \quad \sum_k \sum_g c_{gk}(n_g - c_{gk}), \tag{4.11}$$

$$(Z_3)\ \text{mín} \quad \sum_k \sum_g \|c_{gk} - q_k\|, \tag{4.12}$$

$$(Z_4)\ \text{mín} \quad \sum_k \sum_g \sum_{h>g} \|c_{gk} - c_{hk}\|, \tag{4.13}$$

$$(Z_5)\ \text{mín} \quad \max_k \left\{ \max_g c_{gk} - \min_g c_{gk} \right\}, \tag{4.14}$$

donde

$$\begin{aligned} d_{ij} &= \sum_k \|a_{ik} - a_{jk}\|, \\ n_g &= \sum_i x_{ig}, \\ c_{gk} &= \sum_i a_{ik} x_{ig}, \text{ y} \\ q_k &= \frac{\sum_i a_{ik}}{G}. \end{aligned}$$

En 4.10 se toma d_{ij} como la diferencia entre los alumnos i y j por lo que Z_1 representa una medida de la diversidad total. La función en 4.11 usa c_{gk} que cuenta el número de alumnos del grupo g que exhiben el atributo k y n_g como el número de alumnos en el

grupo g, por lo que Z_2 representa la diversidad entre grupos. Ya que la forma ideal de hacer grupos similares es que los valores c_{gk} sean iguales para todos los grupos, 4.12 se basa en esta idea y define q_k como el promedio de alumnos con el atributo k por grupo y Z_3 representa la desviación total absoluta. Una variante de Z_3 es una forma más convencional, sumando la desviación al cuadrado: $(c_{gk} - q_k)^2$. La función 4.13 es similar a Z_3 sin tener que calcular q_k, minimizando la diferencia entre cada par de grupos. En 4.14 se minimiza la máxima diferencia entre pares de grupos, pudiendo encontrar una mejor solución pero ignorando asignaciones que no sean extremas, por lo que funciona mal cuando se necesita hacer desempates o considerar un segundo factor.

Para usar los componentes de diversidad y preferencias, Hubscher [13] propone usar una función objetivo de la forma:

$$f(x) = g(x) + \sum_i \alpha_i h_i(x), \tag{4.15}$$

donde $g(x)$ es la función para un criterio general de asignación y $h_i(s)$ son funciones para criterios más específicos, con sus respectivos pesos α_i de acuerdo a su importancia. Dentro de los criterios considerados, se propone una estrategia para asignar un conjunto de alumnos de forma distribuida entre los grupos. Sea R el conjunto de alumnos que se busca distribuir, se define H_{ij} como sigue:

$$H_{ij} = \begin{cases} -1 & \text{si } i \in R, j \in R \text{ y } i \neq j, \\ 0 & \text{en otro caso,} \end{cases} \tag{4.16}$$

de forma que se puede agregar un componente $h(x)$:

$$h(x) = \sum_g \sum_i \sum_{j>i} x_{ig} x_{jg} H_{ij}, \tag{4.17}$$

que penalizará la asignación si alumnos en R pertenecen al mismo grupo.

Por otra parte, Chiarandini y col. [3] discuten el manejo de preferencias en la asignación de alumnos a proyectos para obtener una asignación pareto–óptima [1]: en donde no se puede mejorar la asignación de un alumno sin empeorar la de otro. Para asignar proyectos

[1] La optimalidad de Pareto es un concepto de eficiencia usado en las ciencias sociales que define a un estado como pareto–óptimo si y solo si no existe una alternativa que mejore la situación de al menos un individuo sin hacer que empeore la de los demás.

se tienen dos aproximaciones: siguiendo un orden utilitario que, aunque puede mejorar la satisfacción en general, algunos alumnos quedan en desventaja; o un orden igualitario maximizando el número de alumnos con grupos en cada preferencia (maximizar los que tienen su primera opción, luego la segunda, etcétera). En el orden utilitario se propone minimizar la siguiente función:

$$f(x) = \sum_{h=1}^{\Delta} w_h R_h, \tag{4.18}$$

donde R_h es el número de alumnos que fueron asignados en un grupo de preferencia h, Δ es el número de preferencias y w_h es el peso que tiene la asignación a grupos de preferencia h. La estrategia más simple para w_h es la función identidad, mientras que una exponencial es:

$$w_h = -2^{\text{máx}[K-h,0]}, \quad h = 1, \ldots, \Delta,$$

donde K es un parámetro de entrada de acuerdo al número de preferencias. Esta alternativa es más drástica, pues penalizará asignaciones a grupos que son de menor preferencia pudiendo ocasionar una asignación injusta.

Para hacer que una asignación sea justa se debe garantizar que el alumno que recibe la peor asignación sea asignado a un grupo cuya preferencia es tan alta como sea posible. Para lograrlo se propone usar el orden lexicográfico comparando dos asignaciones respecto a la peor preferencia dada, si los valores difieren se opta por la asignación con mejor preferencia y si son iguales se busca la segunda peor asignación. Otra alternativa es resolver $\Delta - 1$ problemas donde se maximice la asignación a proyectos de preferencia h.

4.2.2 Modelo para la asignación de grupos

El problema de asignación de grupos puede verse como una instancia particular del PAG como se muestra en la tabla 4.1. Sea N el número de alumnos y G la oferta de grupos, se usan las mismas variables de decisión definidas en 4.4 así como las restricciones 4.5 y 4.6 definidas sobre ellas.

Sea Δ el número de preferencias y P_{ij} la preferencia del alumno i por el grupo j, tal que $P_{ij} = h$ si j es la h–ésima opción del alumno i; y $P_{ij} = \Delta + 1$ si j no está dentro de las preferencias del alumno. Definimos el costo c_{ij} de asignarle el grupo j al alumno i como sigue:

PAG	*Asignación de grupos*
tareas	alumnos
agente	grupo
c_{ij}	preferencia del grupo
b_i	C_g
a_{ij}	1

Tabla 4.1 – El problema de asignación de grupos como una instancia del PAG.

$$c_{ij} = \omega(P_{ij}), \tag{4.19}$$

donde ω es un factor de peso asociado a la preferencia h. Para lidiar con alumnos asignados a un grupo fuera de sus opciones se tienen las siguientes alternativas:

- Agregar la restricción
$$\sum_{i=1}^{N}\sum_{j=1}^{G} x_{ij}P_{ij} \leq \Delta,$$
garantizando que todos los alumnos se asignen a un grupo dentro de sus opciones.
- Definir $\omega(\Delta + 1) \to \inf$ para que soluciones con alumnos asignados fuera de sus opciones no sean consideradas al minimizar – a menos que no exista una asignación donde todos queden en un grupo de su preferencia.

Si bien la primera alternativa es atractiva, la restricción es muy fuerte: se obtiene un modelo que probablemente no tenga solución.

Para manejar la restricción del balance de grupos se usará la función definida en 4.13, donde el atributo k representa la situación escolar del alumno. Sea R el número de atributos a considerar y c_{gk} el número de alumnos en el grupo g cuya situación escolar es k, la desigualdad

$$\sum_{g}\sum_{h>g} \|c_{gk} - c_{hk}\| \leq \sigma, \quad k = 1, \ldots, R,$$

restringe la diferencia de alumnos con atributo k en cada grupo. El número de atributos a considerar es descrito en el apéndice B. En lo que resta del capítulo, los atributos k son

considerados un rango de promedio, es decir, se tienen 6 atributos de acuerdo al promedio del alumno: menor o igual a 5, entre 5 y 6, etcétera.

Sean I_g y O_g el conjunto de idiomas y optativas disponibles en el grupo g respectivamente, definimos las variables binarias I_{jgx} y O_{jgx} que indican si el alumno j cursa la asignatura x en el grupo g. El cupo mínimo de estas asignaturas se define como sigue:

$$\sum_{j=1}^{N} \sum_{i \in I_g} x_{jg} I_{jgi} \geq K_1, \ g = 1, \ldots, G, \tag{4.20}$$

$$\sum_{j=1}^{N} \sum_{o \in O_g} x_{jg} O_{jgo} \geq K_2, \ g = 1, \ldots, G, \tag{4.21}$$

donde K_1 y K_2 son el cupo mínimo para optativas e idiomas, respectivamente.

El modelo resultante se muestra a continuación.

Definición 4.1

Sea N el número de alumnos; G el número de grupos disponibles; Δ el número de opciones registradas; I_g y O_g las asignaturas de idioma y optativas de área que ofrece el grupo g, respectivamente; se definen las siguientes variables de entrada:

$$p_{ik} = \begin{cases} 1 & \text{si el alumno } i \text{ tiene promedio en el rango } k, \\ 0 & \text{en otro caso,} \end{cases}$$

$$P_{ij} = \begin{cases} h & \text{si } j \text{ es la } h\text{–ésima opción del alumno } i, \\ \Delta + 1 & \text{en otro caso,} \end{cases}$$

$$I_{igj} = \begin{cases} 1 & \text{si } j \text{ es el idioma a cursar por el alumno } i \text{ en el grupo } j, \\ 0 & \text{en otro caso,} \end{cases}$$

$$O_{igj} = \begin{cases} 1 & \text{si } j \text{ es una optativa seleccionada por el alumno } i \text{ en el grupo } j, \\ 0 & \text{en otro caso.} \end{cases}$$

La asignación de grupos tiene como objetivo

$$\text{mín} \sum_{i=1}^{N} \sum_{j=1}^{G} \omega(P_{ij}) x_{ij}, \tag{4.22}$$

cumpliendo que

$$\sum_{g=1}^{G} x_{sg} = 1, \quad s = 1, \ldots, N, \tag{4.23}$$

$$\sum_{s=1}^{N} x_{sg} \leq C_g, \quad g = 1, \ldots, G, \tag{4.24}$$

$$\sum_{g} \sum_{h>g} \|c_{gk} - c_{hk}\| \leq \sigma, \quad k = 5, \ldots, 10, \tag{4.25}$$

$$\sum_{j \in N} \sum_{i \in I_g} x_{jg} I_{jgi} \geq K_1, \quad g = 1, \ldots, G, \tag{4.26}$$

$$\sum_{j \in N} \sum_{o \in O_g} x_{jg} O_{jgo} \geq K_2, \quad g = 1, \ldots, G, \tag{4.27}$$

donde

$$x_{ij} = \begin{cases} 1 & \text{si el alumno } i \text{ es asignado al grupo } j, \\ 0 & \text{en otro caso,} \end{cases} \tag{4.28}$$

$$c_{gk} = \sum_{i=1}^{N} p_{ik} x_{ig}, \tag{4.29}$$

y ω, σ, K_1, K_2 y C_g son parámetros del problema:

- $\omega(h)$ es el peso asociado a la preferencia h.
- σ es la diferencia de cupo máxima de alumnos por promedio entre grupos.
- K_1 es el cupo mínimo para idiomas por grupo.
- K_2 es el cupo mínimo para optativas por grupo.
- C_g es el cupo del grupo g.

⊣

En el modelo propuesto se tienen NG variables y $N + 3G + 5$ restricciones, por ejemplo: para una instancia con 1,000 alumnos y 18 grupos se tienen 18,000 variables y 1,059 restricciones, por lo que se vuelve complicado atacar el problema con métodos exactos. Por otra parte, el espacio de búsqueda es de tamaño

$$\frac{N!}{C_g!^G},$$

con el mismo ejemplo, se trata de alrededor de 5.48×10^{1251} combinaciones de alumnos en grupos. En este trabajo se usará un método aproximado para resolver el problema de asignación, usando una variación del modelo propuesto:

- Las variables de decisión se particionan en R subconjuntos N_k de acuerdo a los promedios de los alumnos, donde N_k corresponde a los alumnos cuyo promedio está en el rango k. La restricción 4.24 queda en función de k:

  $$\sum_{s \in N_k} x_{sg} \leq \frac{C_g |N_k|}{G}, \quad g = 1, \ldots, G;\ k = 1, \ldots, R.$$

  Esto permite descomponer la asignación en R asignaciones, cada una con menos variables, sin perder el balance del grupo dado por la restricción 4.25.

- Las restricciones de cupo para los idiomas y optativas se convierten en un factor de penalización de la función de costo:

  $$f(\overline{x}) = \sum_{j=1}^{G} \left(\sum_{i=1}^{N} \omega(P_{ij}) x_{ij} + \alpha_1 h_1(\overline{x}, g) + \alpha_2 h_2(\overline{x}, g) \right)$$

  siendo α_i el factor de penalización y h_i de la forma:

  $$\sum_{a \in S_g} \text{mín}\{M - \sum_{i=1}^{N} x_{ig} S_{iga}, 0\}$$

  donde S_g, M, S_{iga} corresponden a I_g, K_1, I_{iga} para idiomas y O_g, K_2, O_{iga} en el caso de las optativas. Estas funciones miden el número de alumnos faltantes para cumplir con el cupo mínimo en cada idioma u optativa del grupo g, de acuerdo a las desigualdades 4.26 y 4.27.

El modelo adaptado se muestra a continuación.

Definición 4.2

Sean N el número de alumnos; G el número de grupos disponibles; Δ el número de

opciones registradas; R el número de rangos de promedios; I_g y O_g las asignaturas de idioma y optativas de área que ofrece el grupo g, respectivamente; se definen las siguientes variables de entrada:

$$N_k = \{i \mid \text{ el alumno } i \text{ tiene promedio en el rango } k\}, \tag{4.30}$$

$$P_{ij} = \begin{cases} h & \text{si } j \text{ es la } h\text{–ésima opción del alumno } i, \\ \Delta + 1 & \text{en otro caso,} \end{cases} \tag{4.31}$$

$$I_{igj} = \begin{cases} 1 & \text{si } j \text{ es el idioma a cursar por el alumno } i \text{ en el grupo } j, \\ 0 & \text{en otro caso,} \end{cases} \tag{4.32}$$

$$O_{igj} = \begin{cases} 1 & \text{si } j \text{ es una optativa seleccionada por el alumno } i \text{ en el grupo } j, \\ 0 & \text{en otro caso.} \end{cases} \tag{4.33}$$

La asignación de grupos tiene como objetivo

$$\text{mín} \sum_{j=1}^{G} \left(\sum_{i=1}^{N} \omega(P_{ij}) x_{ij} + \alpha_1 h_1 + \alpha_2 h_2 \right), \tag{4.34}$$

cumpliendo que

$$\sum_{g=1}^{G} x_{sg} = 1, \quad s = 1, \ldots, N, \tag{4.35}$$

$$\sum_{s \in N_k} x_{sg} \leq \frac{C_g |N_k|}{G}, \quad g = 1, \ldots, G;\ k = 1, \ldots, R, \tag{4.36}$$

donde

$$x_{ij} = \begin{cases} 1 & \text{si el alumno } i \text{ es asignado al grupo } j, \\ 0 & \text{en otro caso,} \end{cases} \tag{4.37}$$

$$h_1 = \sum_{a \in I_g} \text{mín}\{K_1 - \sum_{i=1}^{N} x_{ig} I_{iga}, 0\}, \tag{4.38}$$

$$h_2 = \sum_{a \in O_g} \text{mín}\{K_2 - \sum_{i=1}^{N} x_{ig} O_{iga}, 0\}, \tag{4.39}$$

y ω, α_i, K_i y C_g son parámetros del problema:

- $\omega(h)$ es el peso asociado a la preferencia h.
- K_1 es el cupo mínimo en idiomas por grupo y α_1 el peso de la penalización.
- Análogamente, K_2, α_2 son el cupo mínimo por optativa y el peso de la penalización.
- C_g es el cupo del grupo g.

⊣

En el siguiente capítulo se discutirá el Algoritmo de Flujo de Agua para atacar el problema de asignación, debido a los buenos resultados que ha tenido en problemas de agrupación, además de ser un método novedoso por la introducción de una población de tamaño variable. Ya que se desconocen las características del paisaje adaptativo para este modelo, una metaheurística con tamaño fijo de soluciones no tendría la garantía de realizar una búsqueda eficiente al mismo tiempo que evita óptimos locales o una convergencia prematura [22].

Algoritmo de Flujo de Agua y su implementación

5

El Algoritmo de Flujo de Agua (AFA) es una metaheurística poblacional inspirada por la trayectoria de un flujo de agua en un terreno, donde una solución es un flujo y el terreno se modela a partir del espacio de búsqueda del problema de optimización. El movimiento de un flujo de agua hacia una posición de menor altura equivale a la búsqueda de un óptimo. A diferencia de otras metaheurísticas poblacionales, el AFA tiene un conjunto de soluciones de longitud variable, iniciando con solo una solución y agregando o quitando soluciones de acuerdo al estado de la búsqueda.

El concepto de la población de longitud variable ataca dos problemas:

- Reduce el número de búsquedas redundantes, las cuáles suelen incrementar el costo computacional.
- El ajuste del tamaño de población: la parametrización *online* necesita más información del problema a resolver; mientras que hacerlo *offline* –a prueba y error por una persona– es tardado y propenso a errores.

La primera versión del AFA fue desarrollada por F.-C. Yang y col. en 2007 para resolver un problema de agrupación de objetos (*Bin-packing problem*), conocido por ser *NP*–duro. Ha sido adaptado y aplicado a diferentes problemas de optimización combinatoria, incluyendo asignación de horarios a enfermeras, el PAV [33] y líneas de producción [22], obteniendo buenos resultados.

Otros algoritmos metaheurísticos inspirados en la dinámica del agua son Formación de ríos (*River Formation Dynamics*), Gotas de Agua Inteligentes (*Intelligent Water Drops*) y el Algoritmo del Ciclo Hidrológico (*Hydrological Cycle Algorithm*), que son semejantes a Colonia de Hormigas [29].

En este capítulo se discutirá la estrategia de búsqueda del AFA propuesta por F.-C. Yang y col., dando la estructura general del algoritmo, parámetros requeridos y una explicación de los operadores de búsqueda usados. Posteriormente se darán los detalles de implementación para resolver el problema de asignación de grupos usando el modelo descrito en 4.2.

5.1 Algoritmo de Flujo de Agua

En el AFA se considera un problema de minimización donde las soluciones son modeladas como flujos de agua con cierta masa M y velocidad V; la función objetivo da la altitud de un flujo y el movimiento por el terreno equivale al movimiento en el vecindario de una solución. Los comportamientos adoptados por el AFA son listados en la tabla 5.1.

Dada la energía cinética MV de una solución, ésta se mueve a una nueva ubicación para recorrer el espacio de búsqueda, es decir, es reemplazada por una o más soluciones vecinas. La ley de la conservación de energía es usada para asignar la masa y velocidad a soluciones derivadas de otra: la masa se divide proporcionalmente de acuerdo a la calidad de las soluciones y la velocidad se calcula tomando en cuenta la mejora del costo, por lo que soluciones mejores ganarán más velocidad para explorar regiones prometedoras del espacio de búsqueda. Un caso especial es cuando la solución no mejora, es decir, se necesita un *salto*; en esta situación, si la velocidad del flujo es suficiente, el flujo de agua se mueve y de otra manera se estanca. Cuando varios flujos llegan a la misma región, se unen para reforzar la búsqueda.

Para evitar quedar en óptimos locales, se introduce una estrategia de reinicio, imitando la evaporación y lluvia: las soluciones perderán masa gradualmente hasta que cierta cantidad se acumule y se introduzcan nuevas soluciones que amplíen la exploración del espacio de búsqueda.

Flujo de agua real	*Algoritmo de Flujo de Agua*
El agua fluye a regiones de menor altura.	La búsqueda se moverá a soluciones de menor costo en cada iteración.
El impulso del flujo lleva al agua a moverse a través de terrenos difíciles, dividiéndose si encuentra terreno accidentado.	Hay un operador de división que generará una o más soluciones a partir de una, dependiendo de su calidad.
Los flujos de agua se unen en uno solo cuando se encuentran en la misma ubicación.	Existe un operador de unión que mezcla varias soluciones en la misma región para evitar una búsqueda redundante o eliminar flujos estancados.
Un flujo de agua está sujeto a la evaporación y el agua evaporada regresa al terreno a través de la lluvia.	Existen operadores de evaporación y lluvia que eliminan soluciones de la población y que introducen soluciones nuevas, respectivamente.

Tabla 5.1 – Principios adoptados en el AFA.

5.1.1 Estructura general

El AFA comienza con una solución aleatoria, a la que se aplican los operadores de búsqueda en cierto orden, modificando el tamaño de la población en cada iteración hasta

llegar a un límite de iteraciones. La figura 5.1 muestra el orden de aplicación de los operadores.

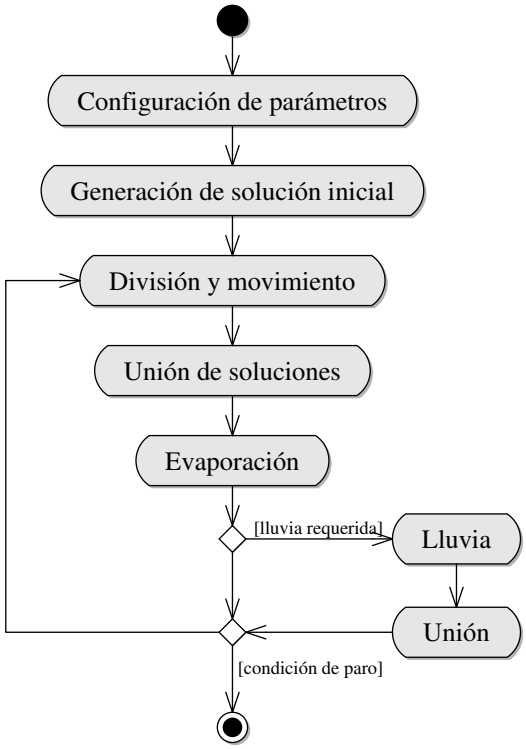


Figura 5.1 – Estructura general del Algoritmo de Flujo de Agua.

La representación, función objetivo y el vencidario a usar dependerán de la implementación para un problema de optimización, por lo que solo se discutirán los operadores de búsqueda y parámetros requeridos.

5.1.2 Operadores de búsqueda

La característica principal del AFA es el operador de división, que se encarga de diversificar la búsqueda usando una estrategia evolutiva; cuando se tienen soluciones similares el operador de unión reduce el número de soluciones quitando aquellas con costo similar o que no hayan mejorado. Para balancear la intensificación y diversificación, los operadores de evaporación y precipitación reinician el estado de la búsqueda para evitar una convergencia prematura.

5.1.2.1 División

Sea $S = \{s_1, \ldots, s_N\}$ las soluciones de la iteración, el número n de soluciones generadas a partir de la solución $s_i \in S$ es determinado por su impulso $M_i V_i$:

$$n = \text{mín}\left\{\text{máx}\left\{1, \left\lfloor \frac{M_i V_i}{T} \right\rfloor\right\}, \overline{n}\right\}, \tag{5.1}$$

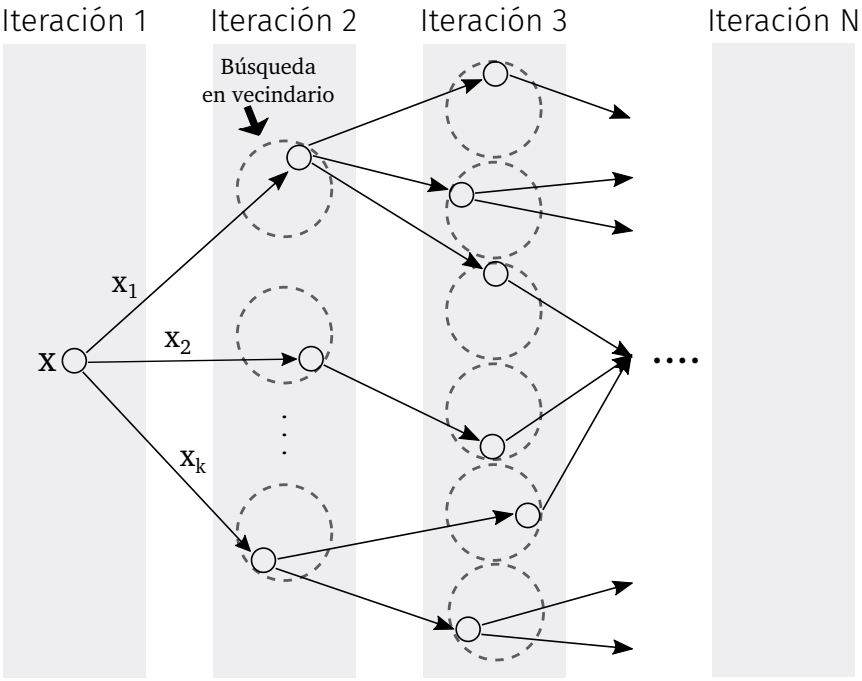


Figura 5.2 – Mecanismo de división, movimiento y unión de los flujos de agua [40].

donde T es el impulso mínimo requerido para dividirse y $\overline{n}$ es el número máximo de soluciones que pueden generarse. Si $V_i = 0$ no se generarán soluciones nuevas ni habrá movimiento de la solución; si $n = 1$ la solución actual se moverá a otra región sin dividirse; y si $n > 1$ se generarán nuevas soluciones que reemplacen a s_i.

Para definir el movimiento del flujo, se obtienen n soluciones de $N(s_i)$ y se ordenan de acuerdo a la función objetivo, por lo que se trata de un operador de búsqueda local.

Cuando la solución actual se divide en otras n soluciones $u_{i1}, \ldots, u_{in}$, la masa original M_i se distribuirá proporcionalmente a la calidad de cada solución:

$$M_{ik} = \left(\frac{n + 1 - k}{\sum_{r=1}^{n} r} \right) M_i, \quad k = 1, \ldots, n. \tag{5.2}$$

Por ejemplo, si $M_i = 5$ y $n = 3$ las masas asignadas a cada solución son 2.5, 1.7 y 0.8 respectivamente. La velocidad de cada solución dependerá de cuánto mejora su costo respecto a s_i:

$$V_{ik} = \begin{cases} \sqrt{V_i^2 + 2g\delta_{ik}} & \text{si } V_i^2 + 2g\delta_{ik} > 0, \\ 0 & \text{en otro caso,} \end{cases} \tag{5.3}$$

donde g es la aceleración gravitacional y

$$\delta_{ik} = f(s_i) - f(u_{ik}) \tag{5.4}$$

denota la variación de la altitud –mejora dada por la función objetivo f– de la solución generada. Cuando $V_i^2 + 2g\delta_{ik} < 0$, el impulso de la nueva solución ha sido agotado: la nueva solución no presenta una mejora significativa, por lo que u_{ik} representa un óptimo local. Si el impulso es suficiente, el flujo generado podría llegar a ser de mayor costo que s_i aunque éste peredería velocidad.

Después de haber actualizado cada solución, se reemplaza la población actual por las soluciones generadas quedando susceptible al operador de unión.

5.1.2.2 Unión

Cuando más de dos soluciones están en la misma región, las masas y velocidades se acumulan para crear una nueva solución y reforzar la búsqueda. Al disminuir el número de soluciones se previene una búsqueda redundante y ayuda a soluciones estancadas a moverse. Dos soluciones están en la misma región si tienen el mismo costo o representación, aunque se podría implementar un radio de diferencia para considerar dos soluciones iguales.

Dada la población actual, para cada dos soluciones s_i y s_j que estén en la misma región, la segunda se une a la primera y es eliminada. Esto se hace hasta que no haya soluciones con el mismo costo.

La masa resultante de la unión de dos soluciones es:

$$M_i = M_i + M_j, \tag{5.5}$$

y la velocidad:

$$V_i = \frac{M_i V_i + M_j V_j}{M_i + M_j}, \tag{5.6}$$

siguiendo la ley de conservación de movimiento. La unión de flujos se realiza después de la división y lluvia para eliminar soluciones redundantes.

5.1.2.3 Evaporación

En cada iteración una solución está sujeta a la evaporación, por lo que cada t iteraciones las soluciones que no han sido modificadas por los operadores anteriores son eliminadas. Este operador sigue una estrategia de reinicio para evitar que la búsqueda se estanque.

La masa resultante de cada solución es actualizada como sigue:

$$M_i = \left(1 - \frac{1}{t}\right) M_i, \tag{5.7}$$

es decir, disminuye por un factor de $1/t$.

5.1.2.4 Precipitación

Cada t iteraciones este operador introduce nuevas soluciones a la población actual para incrementar la amplitud de la búsqueda. Se consideran N soluciones nuevas, duplicando el tamaño de la población. Sea M_0 la masa y V_0 la velocidad de la solución inicial, en el momento que este operador actúa, la masa acumulada por la evaporación es $M_0 - \sum_{k=1}^{N} M_k$ que es distribuida como sigue:

$$\overline{M}_i = \left(\frac{M_0}{\sum_{k=1}^{N} M_k} - 1\right) M_i, \quad i = 1, \ldots, N, \tag{5.8}$$

mientras que la velocidad de cada solución es inicializada en V_0. La solución $\overline{s_i}$ es obtenida a partir de $N(s_i)$; a diferencia de la división, las soluciones obtenidas deben diversificar la búsqueda por lo que se sugieren modificaciones aleatorias a $\overline{s_i}$.

En el caso de que todas las soluciones reciban velocidad cero después del operador de división, éstas permanecerán hasta que gradualmente se eliminen por el operador de evaporación. Para evitar el gasto de iteraciones en la renovación de soluciones, se propone una lluvia precipitada forzando la evaporación y la generación de N soluciones que reemplacen la población actual. Las soluciones introducidas son modificaciones aleatorias a las que ya se tenían y la masa M_0 se distribuye proporcionalmente a la masa original de las soluciones que se reemplazan:

$$\overline{M}_i = \left(\frac{M_i}{\sum_{k=1}^{N} M_k}\right) M_0, \quad i = 1, \ldots, N. \tag{5.9}$$

En ambos casos, se introducen nuevas soluciones que pueden tener el mismo costo que soluciones ya existentes, por lo que se aplica el operador de unión a la población resultante.

5.1.3 Parámetros

Los siguientes parámetros son requeridos para la búsqueda:

- Límite de iteraciones P en el que la búsqueda se detiene.
- Fuerza gravitacional g usada para calcular la velocidad de soluciones generadas.

- Impulso mínimo T usado para calcular el número de soluciones generadas a partir de otra.
- Número de iteraciones t para que haya lluvia.
- Masa M_0 y velocidad V_0 de la primera solución.
- Número máximo de soluciones $\overline{n}$ obtenidas de una solución.

F.-C. Yang y col. sugieren una masa y velocidad inicial tal que $2T \leq M_0V_0 \leq 3T$; para evitar una abrumadora regeneración de soluciones proponen $t < \frac{P}{20}$; y para evitar que la población incremente demasiado sugieren $\overline{n} = 3$.

5.2 Diseño orientado a objetos

Para modelar la asignación y el AFA se tienen las siguientes clases (ver figura 5.3):

- `Solucion` – interfaz para una solución a un problema de optimización, define los métodos asociados a la función de costo y vecindario.
- `FlujoAgua<T>` – implementación de un flujo de agua y sus operadores de búsqueda como se definen en 5.1.2. La clase es genérica para un tipo `T` que implemente la interfaz `Solucion`.
- `Algoritmo` – implementación del algoritmo en sus cuatro fases como se muestra en la figura 5.1 y los parámetros descritos en 5.1.3, así como atributos para indicar la iteración actual, el generador de números aleatorios usado y la población actual. Los atributos `velocidad_minima` y `masa_minima` son usados como cotas inferiores para determinar si un flujo tiene velocidad o masa cero, respectivamente.
- `Estudiante` – representación de un estudiante y sus opciones de grupos ordenadas por preferencia. El método `obtener_preferencia` equivale a la variable P_{ij} (4.31); mientras que `obtener_idioma` y `obtener_optativa` abstraen las variables I_{igj} (4.32) y O_{igj} (4.33). Ambos métodos devuelven un tipo suma: en el caso de no poder obtener un idioma u optativa para el grupo dado, se regresa `None`.
- `Grupo` – representación de un grupo y sus parametros: cupo, idiomas y optativas que ofrece. Estos atributos mantienen el estado actual de una asignación, indicando el número de alumnos asignados en cada idioma y optativa.
- `Asignacion` – implementación de una asignación como solución del PAG. Una asignación es representada por el vector `asignacion` que contiene el grupo de cada alumno por cada rango de promedio, y el vector `descriptor` que contiene el número de alumnos asignados a un grupo con preferencia i. El atributo `estudiantes` es

una referencia a los estudiantes considerados, en el mismo orden que `asignacion`, que se usa para actualizar el costo de la solución. Este atributo es equivalente a los subconjuntos N_k (4.30). El atributo `grupos` representa el estado de la asignación en cada grupo, de tal forma que obtener la penalizaciones de cupo para idiomas y optativas sea eficiente.

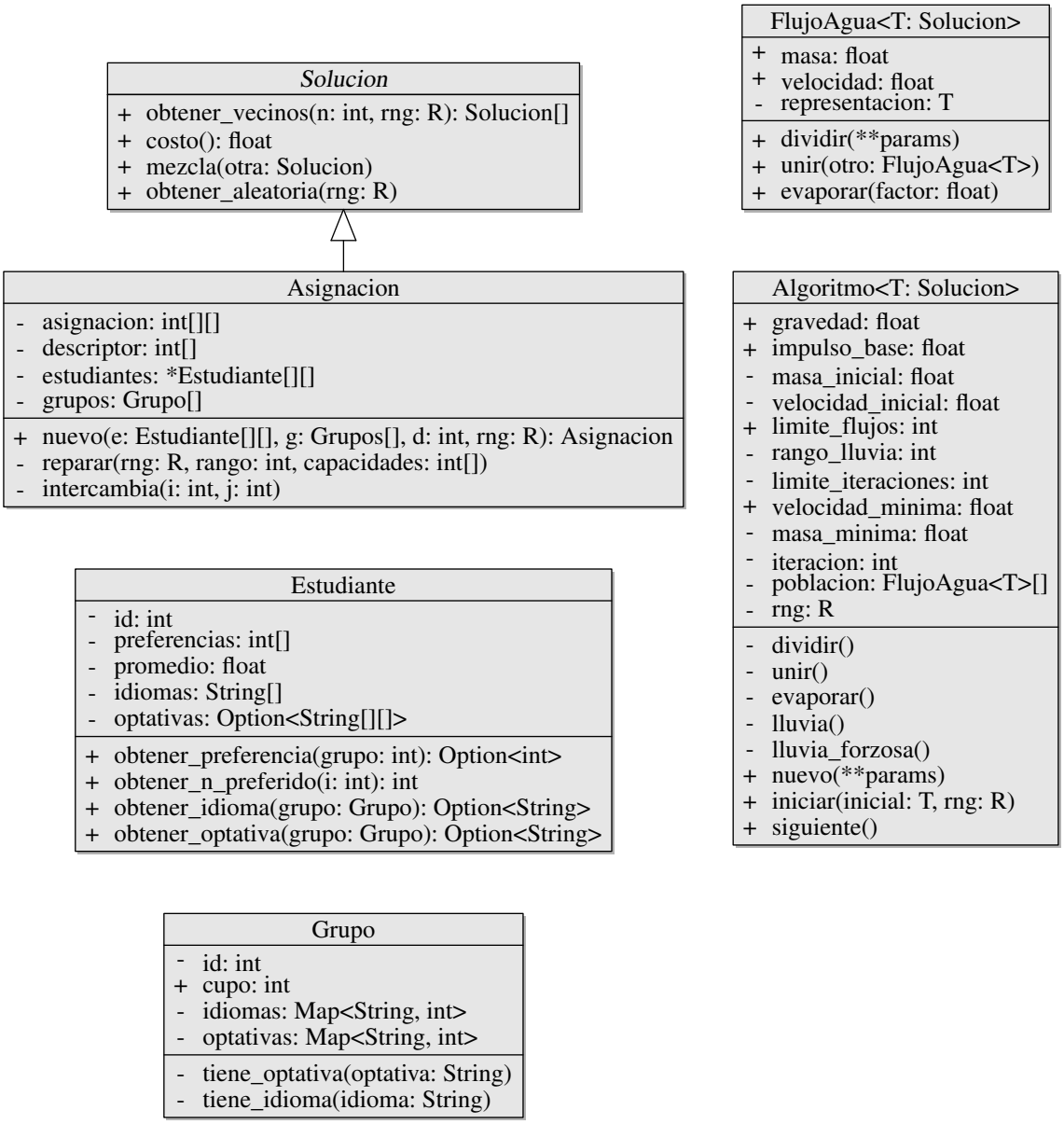


Figura 5.3 – Definición de la clases para el Algoritmo de Flujo de Agua. `R` es el tipo de un generador de números aleatorios y `**params` se usa para abreviar los parámetros requeridos de una función.

5.3 Implementación

A continuación se describe la implementación del modelo 4.2 para el AFA, discutiendo la representación seleccionada y los operadores de búsqueda.

5.3.1 Representación de soluciones

Sea R el número de rangos de promedios a manejar; $N_1, \ldots, N_R$ la partición de alumnos por rango de promedio; y G el número de grupos disponibles; se usa una representación lineal entera donde una solución para N_i es un vector de enteros $\overline{x_i} = \{x_1, \ldots, x_{|N_i|}\}$ tal que $x_j \in [1, G]$ es el grupo asignado al alumno j, por lo que la restricción 4.35 se cumple. El vector $\overline{x_i}$ corresponde al atributo `asignacion`.

Cada que se genera una solución desde cero se usa una estrategia de reparación para balancear el cupo de cada vector $\overline{x_k}$:

1. Sea C_{gk} el cupo del grupo g para el rango k, se identifican los grupos g_i cuya capacidad es mayor a $\lfloor C_{gk} \rfloor$.
2. Se selecciona aleatoriamente un alumno de estos grupos y se asigna a uno que tenga capacidad disponible, dando prioridad a un grupo que sea de su preferencia o en su defecto, tomando un grupo aleatorio dentro de los disponibles.

El paso dos se repite hasta que la restricción 4.36 se cumpla. Con esta definición, la representación usada es un mapeo uno a uno y cumple el principio de inclusión, pues todas las soluciones factibles pueden representarse adecuadamente.

Ejemplo 5.1

Sea $N_i = \{1, \ldots, 10\}, G = 5, C_{gi} = 2\ \forall g$, consideremos las siguientes preferencias:

	alumno									
g	1	2	3	4	5	6	7	8	9	10
1	1	-	1	1	-	-	-	-	-	-
2	2	1	-	2	1	1	1	-	-	1
3	3	2	2	3	2	2	-	1	3	2
4	-	3	-	-	3	3	2	2	2	3
5	-	-	3	-	-	-	3	3	1	-

Dada la solución $\overline{x} = (1, 2, 1, 1, 2, 2, 2, 3, 5, 2)$, donde los grupos 1 y 2 tienen alumnos de más, la siguiente secuencia de asignaciones $x_i \to g$ lleva a una solución factible:

$$
\begin{aligned}
3 \to 3, &\quad (1, 2, \mathbf{3}, 1, 2, 2, 2, 3, 5, 2),\\
6 \to 4, &\quad (1, 2, 3, 1, 2, \mathbf{4}, 2, 3, 5, 2),\\
7 \to 4, &\quad (1, 2, 3, 1, 2, 4, \mathbf{4}, 3, 5, 2),\\
2 \to 5, &\quad (1, \mathbf{5}, 3, 1, 2, 4, 4, 3, 5, 2),
\end{aligned}
$$

en cada paso se procura asignar un grupo dentro de las preferencias del alumno considerado; en el último intercambio esto no es posible y al alumno 2 se le asigna un grupo que no está dentro de sus opciones. ⊣

5.3.2 Vecindario

El vecindario se define usando la distancia de Hamming: para obtener un vecino se intercambia el grupo de dos alumnos en un rango k. Para el operador de división se consideran N intercambios aleatorios, uno para cada alumno. Para el operador de lluvia se invierte una subsecuencia en un vector N_k aleatorio.

Sea Δ el número de opciones registradas, cada solución x tiene asociado el vector $\delta = (a_1, \ldots, a_\Delta, a_{\Delta+1})$ donde

$$
a_h = \sum_{i=1}^{N} \sum_{j \in \{g \mid P_i g = h\}} x_{ij},
$$

es decir, a_h es el número de alumnos asignados a un grupo de preferencia h en x. El vector δ corresponde al atributo `descriptor`. Análogamente, un grupo tiene los atributos `idiomas` y `optativas` que indican el número de alumnos asignados a un idioma u optativa m en el grupo g.

Al realizar un intercambio de grupos, los vectores `asignacion` y `descriptor` son actualizados, así como los vectores `idiomas` y `optativas` de los grupos afectados. Esta operación preserva las restricciones 4.35 y 4.36, pues no se duplica la asignación de grupo para un alumno ni se agregan más alumnos a un solo grupo.

Ejemplo 5.2

Tomando la asignación del ejemplo anterior, tenemos:

- alumnos en su primera opción: 1, 4, 5, 8, 9, 10,
- alumnos en su segunda opción: 3, 7,
- alumnos en su tercera opción: 6,
- alumnos en ninguna de sus opciones: 2,

denotado con $\delta = (6, 2, 1, 1)$; consideremos los posibles intercambios para el alumno 2:

opción	*alumno*	δ
a	1	$(5, 2, 1, 2)$
b	3	$(6, 2, 2, 0)$
c	4	$(5, 2, 1, 2)$
d	5	$(6, 2, 1, 1)$
e	6	$(6, 2, 1, 1)$
f	7	$(6, 1, 3, 0)$
g	8	$(5, 3, 2, 0)$
h	10	$(6, 2, 1, 1)$

donde el alumno 9 no es considerado por tener el mismo grupo. Las opciones d, e y h no implican una mejora pues mantienen la misma cantidad de alumnos en sus opciones; las opciones a y c empeoran la asignación al mover a un alumno fuera de sus opciones sin mejorar la asignación del alumno 2; las opciones b, f y g son buenas soluciones candidatas pues tienen a todos los alumnos dentro de sus opciones. ⊣

5.3.3 Función de costo

Podemos ver a la función 4.34 de la forma

$$f(x) = f'(x) + f_1(x) + f_2(x)$$

donde f' es el costo de la asignación dado por las preferencias de los alumnos, y f_i son las penalizaciones del cupo de los idiomas y optativas. Con esta representación $f'(x)$ es equivalente a

$$\sum_{h=1}^{\Delta+1} w_h a_h,$$

donde w_h es un costo constante asociado a la preferencia h. Análogamente, ya que una solución tiene el cupo de cada idioma $d \in I_g$ y optativa $t \in O_g$ para todo grupo g, al iterar el vector `grupos` se puede calcular h_i (4.38, 4.39), teniendo lo necesario para obtener f_1 y f_2.

5.3.4 Parámetros

De los parámetros listados en 4.2, se tiene:

1. $K_1 = 12$ y $K_2 = 15$ bajo criterios del plantel; mientras que α_1 y α_2 son parámetros de entrada.
2. C_g es parte de la configuración de una instancia de asignación y es dada por el plantel.

En el siguiente capítulo se muestra la experimentación realizada para obtener valores adecuados para w_h y α_i, así como el efecto de la elección de la solución inicial y la estrategia de movimiento.

5.3.5 Complejidad

La complejidad en espacio de la representación de una solución se desglosa en:

- $O(N)$ del vector `asignacion`,
- $O(bG)$ del vector `grupos`, donde b está dado por las asociaciones `idiomas` y `optativas`,
- $O(\Delta)$ del vector `descriptor`.

Si bien actualizar los atributos `asignacion`, `idiomas` y `optativas` afectados por el intercambio de grupos es constante, generar una nueva solución implica *clonar* la representación original. Por lo tanto, la complejidad en tiempo de generar una asignación estará dada por la complejidad en espacio de la representación. Para evitar que la complejidad en tiempo de la exploración del vecindario aumente, se consideran distintos intercambios sin generar una nueva representación.

La complejidad en tiempo de calcular f' es $O(\Delta)$; mientras que obtener f_1 y f_2 toma $O(bG)$, donde b es el número de optativas e idiomas disponibles.

Resultados

6

En este trabajo se contó con los insumos provistos por la ENP 6, que consisten en los registros anonimizados de reinscripción de los años 2015 a 2019. Con esta información se pudieron crear 15 instancias del problema de asignación: una por cada grado (3), por cada ciclo escolar (5). Los insumos relacionados al año 2019 se usaron para la experimentación y el resto para el análisis de la eficiencia de la implementación del AFA.

Se usó una cota justa y la implementación actual para comparar la calidad de las soluciones obtenidas, pues para las instancias usadas no se conocía un óptimo global. La cota inferior se obtuvo analizando la demanda de cada grupo para calcular el mínimo número de alumnos que queda fuera de sus opciones: se asignaron alumnos a los grupos menos demandados y se sumó el cupo faltante de cada uno. Con esta cota se mide la calidad de una solución en términos de cuántos alumnos quedaron asignados en grupos fuera de sus opciones, mientras que la comparación con la implementación actual sirve para medir el balance de los grupos.

En las siguientes secciones se muestran los distintos experimentos realizados para calcular los valores de los parámetros del problema. Las figuras mostradas reflejan la asignación para alumnos de quinto año, que representan la instancia más compleja debido al número de grupos y su demanda. El apéndice B muestra la partición de alumnos considerada.

6.1 Experimentación

La implementación se realizó en el lenguaje de programación Rust[1], cuyos resultados son analizados por un programa escrito en Python[2] para la generación de reportes. Cada ejecución reporta el costo, masa y velocidad de la población, el número de soluciones, y el descriptor de la mejor solución en cada iteración. La implementación puede encontrarse en el apéndice C.

En lo que resta de la sección, la instancia usada fue conformada por 1029 alumnos, 18 grupos, un cupo máximo de 58 y 5 preferencias. En la figura 6.1 se observa la demanda de cada grupo en cada preferencia por cada rango de promedio.

[1] www.rust-lang.org
[2] www.python.org

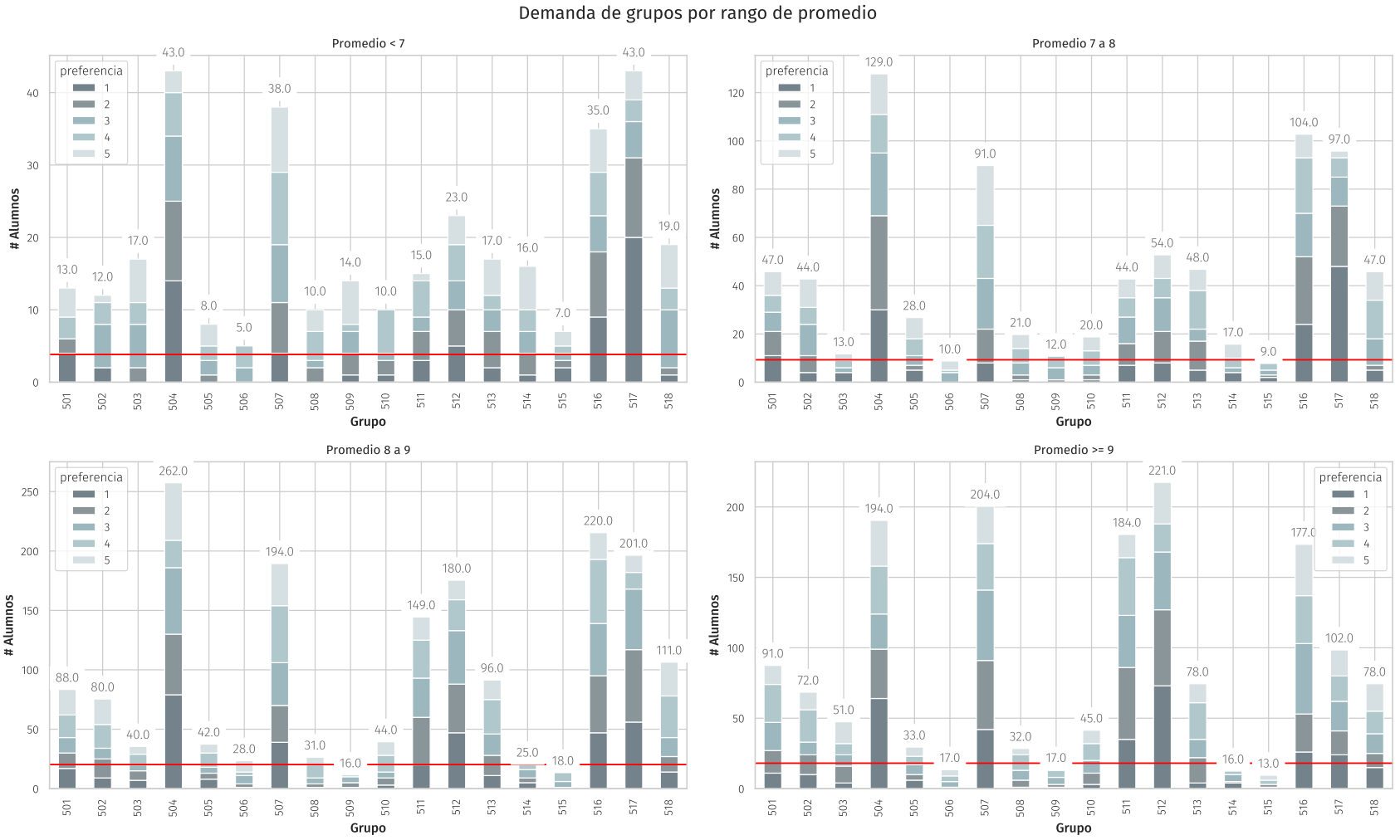


Figura 6.1 – Número de alumnos que solicita un grupo en cada preferencia. La línea roja indica el cupo mínimo de los grupos.

Los parámetros $T, \overline{n}, M_0$ y V_0 del AFA (sección 5.1.3) se basan en [40] y [38]; la velocidad y masa mínima V_{min}, M_{min} toman valores lo suficientemente pequeños, ya que por manejo de punto flotante es muy difícil que una solución llegue a masa o velocidad cero; para t se consideran 100 iteraciones inicialmente; y la gravedad toma valor de 16 dada una experimentación previa. Estos parámetros se listan en la tabla 6.1.

Parámetro	*Valor*
g	16.0
T	20
M_0	8
V_0	5
$\overline{n}$	3
V_{min}	0.01
M_{min}	0.01
t	100

Tabla 6.1 – Parámetros base para experimentación.

6.1.1 Solución inicial

Se tienen tres alternativas para la solución inicial:

1. (GA) A cada alumno se le asigna un grupo de manera aleatoria.
2. (OA) A cada alumno se le asigna un grupo aleatorio dentro de sus opciones.
3. (PO) A cada alumno se le asigna su primera opción.

Se realizaron 25 ejecuciones para cada configuración; con un límite de iteraciones P de 2000 y deteniendo la búsqueda si en 500 iteraciones la mejor solución no cambia. No se considera una penalización de idiomas ($\alpha_1 = 0$) y los pesos w_h se configuran siguiendo una estrategia exponencial: $w_1 = 0.0, w_2 = 0.0, w_3 = 4.0, w_4 = 8.0, w_5 = 16.0, w_6 = 32.0$, donde una solución que mejore el número de alumnos en sus últimas opciones ganará más velocidad que una que solo mantenga la distribución de preferencias.

La figura 6.2 muestra los resultados obtenidos, donde la variante OA tiene un mejor comportamiento que GA al tener el 50 % de sus resultados de menor costo en un rango inferior, mientras que el 50 % superior sigue siendo menor al costo máximo obtenido por GA y PO. En lo que resta del capítulo se utilizó la variante OA para generar la solución inicial.

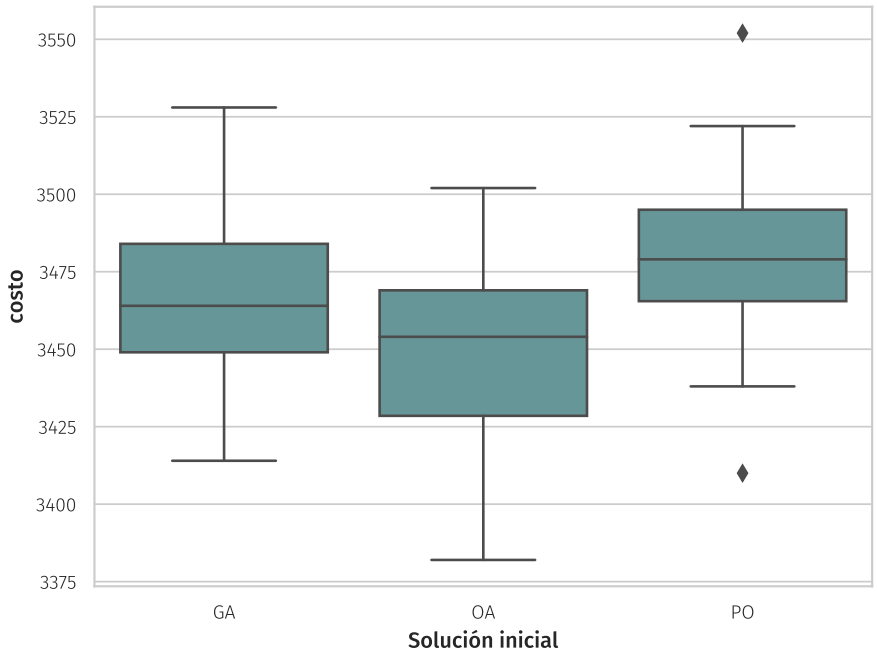


Figura 6.2 – Gráfica de cajas de los costos de las mejores soluciones encontradas en 25 ejecuciones para cada variante de solución inicial.

6.1.2 Función de costo

Hay dos factores que influyen en la velocidad obtenida por una solución nueva: la gravedad y la diferencia de costo. A partir de la configuración de la sección anterior, se experimentó con distintos valores para w_h y ${\alpha_1}^3$ basados en potencias de dos. Se obtuvieron los siguientes resultados:

- Cuando el valor de w_6 (penalización de alumnos fuera de sus opciones) incrementa, la solución se acerca más a la cota mínima usada; sin embargo, si $w_6 > \alpha_1$ la búsqueda prefiere asignaciones que disminuyan alumnos fuera de sus opciones aunque esto implique que el cupo mínimo no se cumpla, pues la penalizacion es menor. Por lo tanto, el peso α_i debe ser mayor o igual a w_6.

- Si w_1, w_2, w_3 son muy similares, al no haber distinción entre primera, segunda y tercera opción, el número de alumnos en su primera opción no es optimizado. Se obtienen mejores resultados si $w_1 \neq w_2 \neq w_3$.

- Si w_5 y w_6 son muy cercanos, el número de alumnos en su última opción y fuera de sus opciones no es distinguido completamente. Para orientar la búsqueda a asignaciones que se disminuyan el número de alumnos fuera de sus opciones, el peso w_6 debe ser significativamente mayor que w_5.

[3] Ya que se trata de una instancia de quinto año, no hay asignaciones de optativas y $\alpha_2 = 0$.

Se realizaron 15 ejecuciones con tres alternativas:

1. $\omega_1 = (0.0, 0.0, 4.0, 8.0, 16.0, 32.0)$ y $\alpha_1 = 32.0$ agregando la penalización α_1 a los pesos de la sección anterior,
2. $\omega_2 = (0.0, 0.5, 1.0, 8.0, 64.0, 256.0)$ y $\alpha_1 = 256.0$,
3. ω_2 y $\alpha_1 = 0.0$ para evaluar el impacto del cupo mínimo en idiomas en las preferencias de los alumnos,

donde $\omega_i = (w_1, \ldots, w_6)$; ya que la función de costo se calcula de forma distinta en cada variante, se obtuvo un promedio del número de alumnos en su i–ésima opción para compararlas. En la figura 6.3 se muestra la distribución obtenida por la mejor solución en cada ejecución de cada variante y la distribución obtenida por el algoritmo actual. Se puede observar que la variante 3 – sin cupo mínimo de idiomas– se acerca mas a la cota mínima calculada, obteniendo en promedio 39 alumnos fuera de sus opciones. Con los pesos ω_1 se obtienen buenos resultados, sin embargo, al agregar la distinción entre alumnos en su primera y segunda opción, ω_2 mejora la distribución. En lo que resta del capítulo se utiliza ω_2.

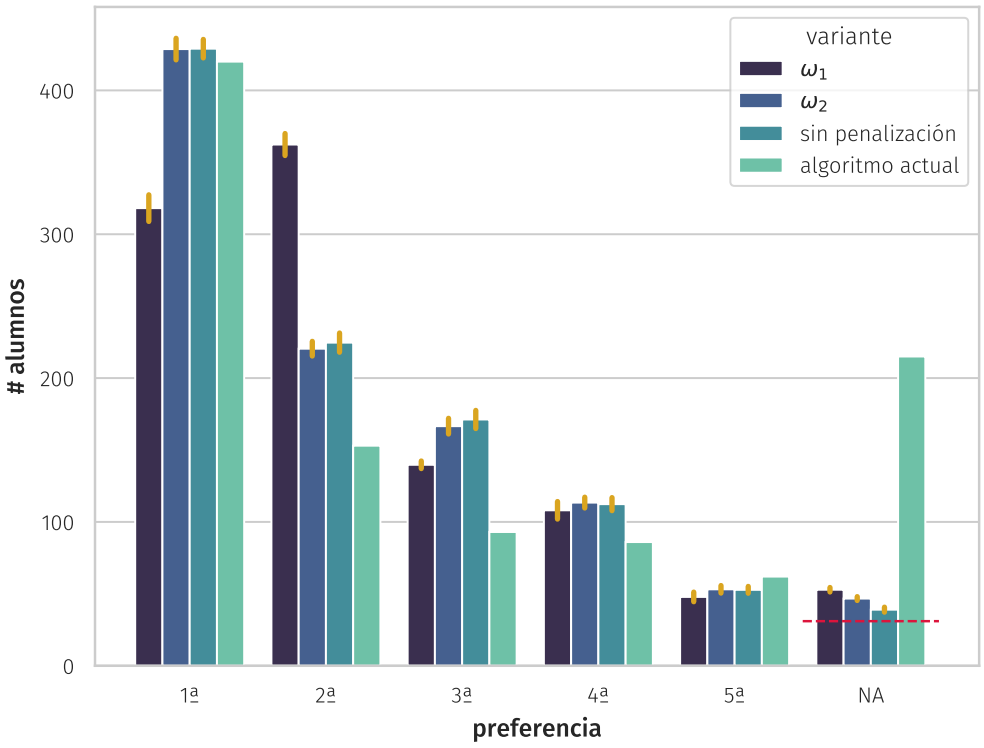


Figura 6.3 – Distribución de alumnos asignados al grupo de su i–ésima opción usando distintos costos. Cada barra muestra su desviación estándar. La cota mínima calculada de alumnos fuera de sus opciones es 31, indicada por la línea punteada.

En cuanto a la gravedad, se experimentó con valores en el rango $g \in [1, 95]$ sin encontrar una mejora significativa (figura 6.4). En el rango $g \in [40, 60]$ se obtuvieron buenos resultados, en espcial cuando $2 \leq \Delta < 4$, pues al haber menos opciones los pesos w_h son mas pequeños y un valor alto en g ayuda a intensificar la búsqueda.

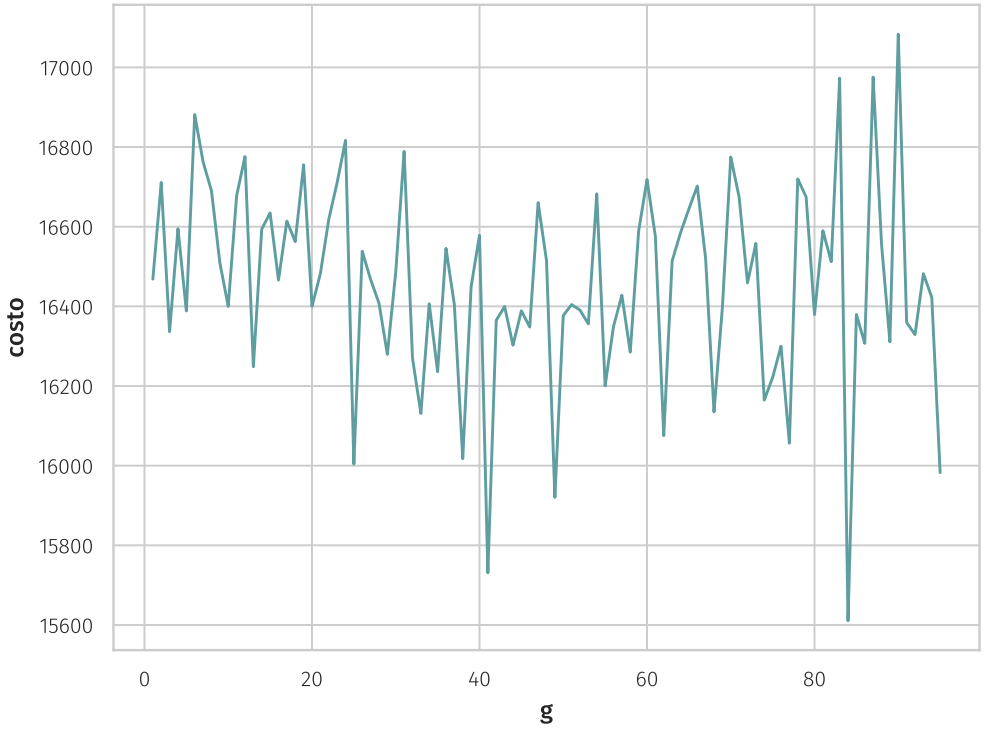


Figura 6.4 – Costo de la mejor solución obtenida con distintos valores para g.

6.1.3 Parámetros del AFA

Retomando los parámetros de la tabla 6.1 se probó con otros valores para T, M_0 y V_0 sin tener gran impacto, debido a que el impulso necesario para intensificar la búsqueda recae en la gravedad y la función de costo. Nourmohammadi y col. [22] proponen otros valores para estos parámetros cuando las instancias con las que trabajan son de gran tamaño.

Para valores distintos de t se encontró que en el rango $[1, 50]$ la búsqueda no obtiene buenos resultados: entre más pequeño su valor, más alumnos quedan fuera de sus opciones. En los experimentos realizados se encontraron buenas soluciones con P alrededor de 3,000; siguiendo las sugerencias del algoritmo original, se propone un valor de $t \in [75, 125]$ y un límite de iteraciones de $3,000$. La tabla 6.2 muestra algunos resultados con distintos valores de t usando $P = 3,000$ y deteniendo la búsqueda si en 700 iteraciones el óptimo global no cambia.

t	*Iteraciones*	*Tiempo de ejecución*	$\delta = (a_1, \ldots, a_\Delta, a_{\Delta+1})$	*Población máxima*
25	1,737	736	(291, 235, 246, 152, 47, 58)	240
100	3,000	1,235	(445, 215, 167, 101, 54, 47)	269
125	2,672	1,035	(438, 218, 162, 113, 51, 47)	264
200	2,648	1,060	(438, 217, 175, 100, 52, 47)	272

Tabla 6.2 – Resultados obtenidos con distintos valores de t. Se muestra el número de iteraciones en que la búsqueda se detuvo, el tiempo de ejecución en segundos, la distribución de alumnos en cada preferencia y la población maxima alcanzada durante la búsqueda.

6.1.4 Eficiencia

En la instancia usada se tienen $R = 5$ vectores de asignación. Como parte de la experimentación se usaron rangos más granulares, por ejemplo: alumnos con promedio entre 9.5 y 10. Sin embargo, no se encontraron mejores resultados al hacer este cambio; esto genera una interrogante sobre una representación que incluya a todos los alumnos, permitiendo explorar soluciones no factibles – donde el balance de los grupos no se cumple– e introduciendo una penalización sobre el número de alumnos por promedio.

Por otra parte, el operador de unión compara todas las soluciones en cada iteración para eliminar soluciones repetidas; la comparación es realizada a partir del vector de asignación, lo cual es de complejidad $O(N)$. Se intentó basar la comparación de acuerdo al descriptor de una solución, sin embargo, esto reducía la amplitud de la búsqueda y se encontraban soluciones inferiores. La implementación resultante compara dos soluciones a partir del *hash* de sus vectores de asignación.

En la figura 6.5 se muestra el comportamiento de la búsqueda. Se puede observar que en alrededor de las primeras 700 iteraciones el óptimo global y local son iguales, debido al número de movimientos de alumnos en cada preferencia; en las siguientes iteraciones la mejor solución no presenta muchos movimientos de alumnos en cada preferencia, mientras que la diferencia entre un óptimo local y global aumenta debido a la diversificación de la búsqueda.

6.2 Comparativa

Con los resultados obtenidos en la sección anterior, se compararon las asignaciones obtenidas por el AFA y el algoritmo actual. Los parámetros finales se listan en la tabla 6.3.

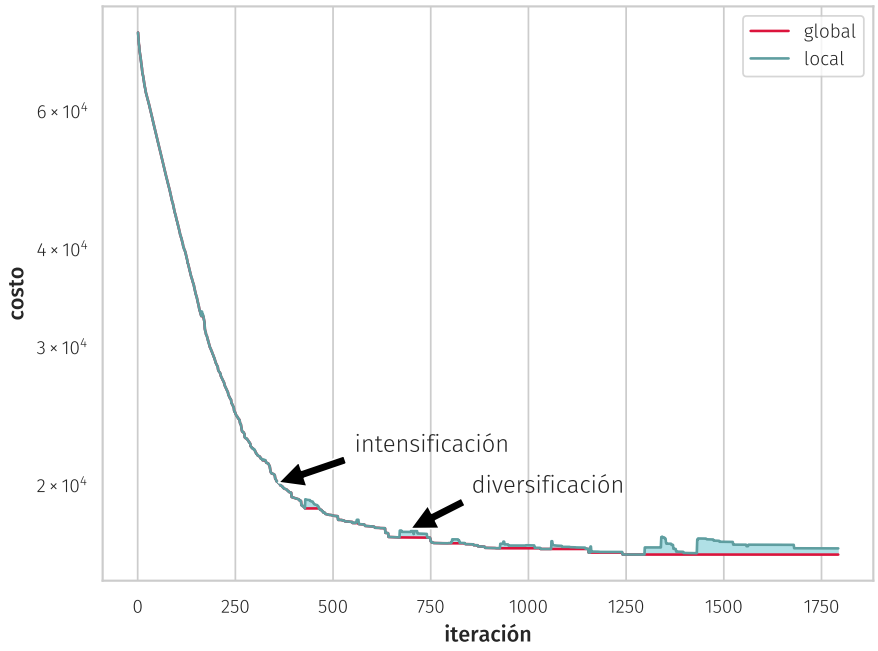
La figura 6.6 muestra la distribución de promedios obtenida, donde la implementación actual mejora el balance de los grupos; la figura 6.7 muestra las preferencias por el grupo asignado, donde es notoria la mejora del número de alumnos fuera de sus opciones; y

Parámetro	*Valor*	*Descripción*
g	56.0	Factor para el impulso de una solución.
T	20	Impulso mínimo para explorar el vecindario.
M_0	8	Masa inicial.
V_0	5	Velocidad inicial.
$\overline{n}$	3	Máximo número de soluciones que se generan a partir de otra.
V_{min}	0.01	Velocidad considerada como cero.
M_{min}	0.01	Masa considerada como cero.
ω	(0.0, 0.5, 1.0, 8.0, 64.0, 256.0)	Pesos para la función de costo.
α_1	256.0	Penalización del cupo mínimo para idiomas.
α_2	256.0	Penalización del cupo mínimo para optativas.
P	3,000	Límite de iteraciones.
t	125	Rango de iteraciones para que haya lluvia.

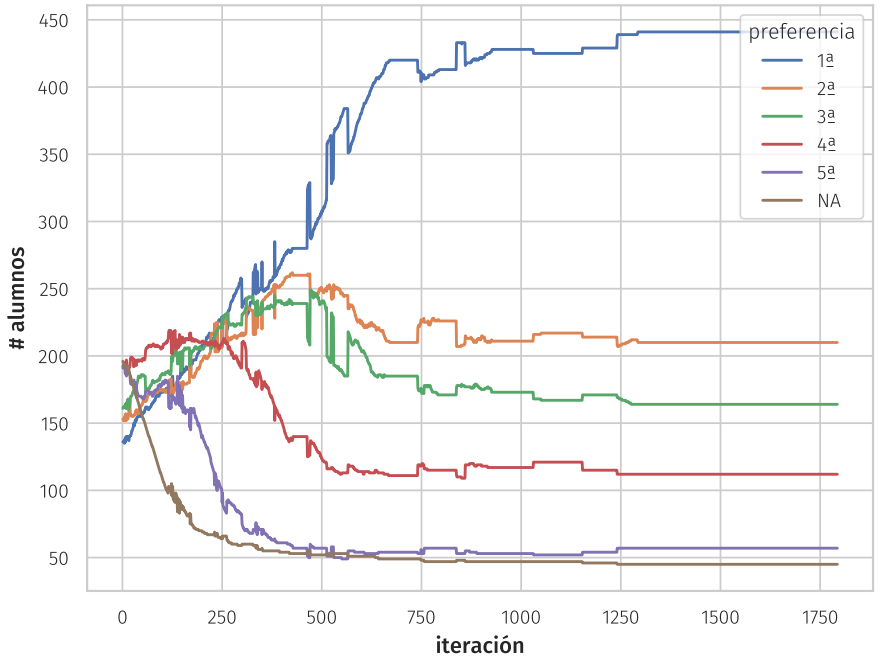
Tabla 6.3 – Parámetros finales usados para la asignación de grupos.

la figura 6.8 muestra el cupo de cada idioma en cada grupo, donde se tiene una mejor distribución debido al cupo mínimo establecido.

Usando los mismos parámetros para todas las ejecuciones, se llegó a asignaciones que cumplen con la cota mínima (o se acercan mucho) de alumnos fuera de sus opciones; cuando se trata de una instancia pequeña (menos de 500 alumnos) se tienen menos de 1,000 iteraciones; en el caso de sexto año, donde hay instancias con menos de cinco grupos, se logró mejorar el número de alumnos dentro de sus primeras tres opciones. Las instancias más complejas son las de quinto año en el turno matutino, debido a la concentración de la demanda en pocos grupos; en estos casos se disminuyó el número de alumnos fuera de sus opciones. Los resultados obtenidos pueden consultarse en el apéndice D.

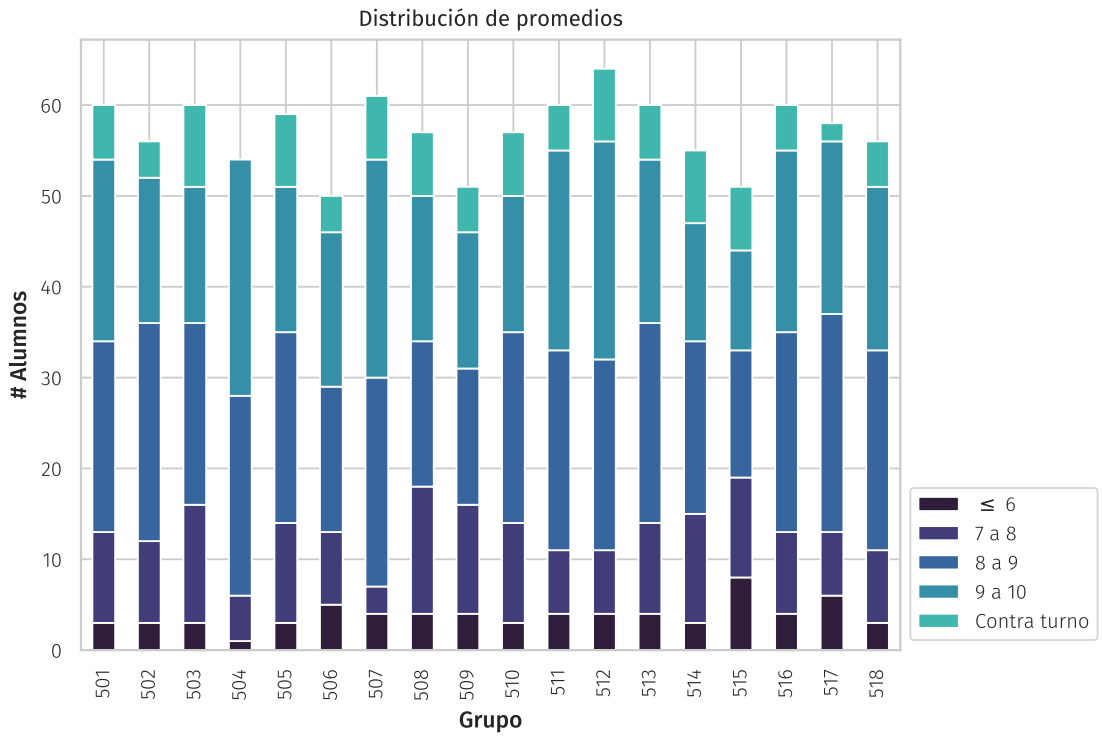


(a) Óptimo global vs Óptimo local en cada iteración.

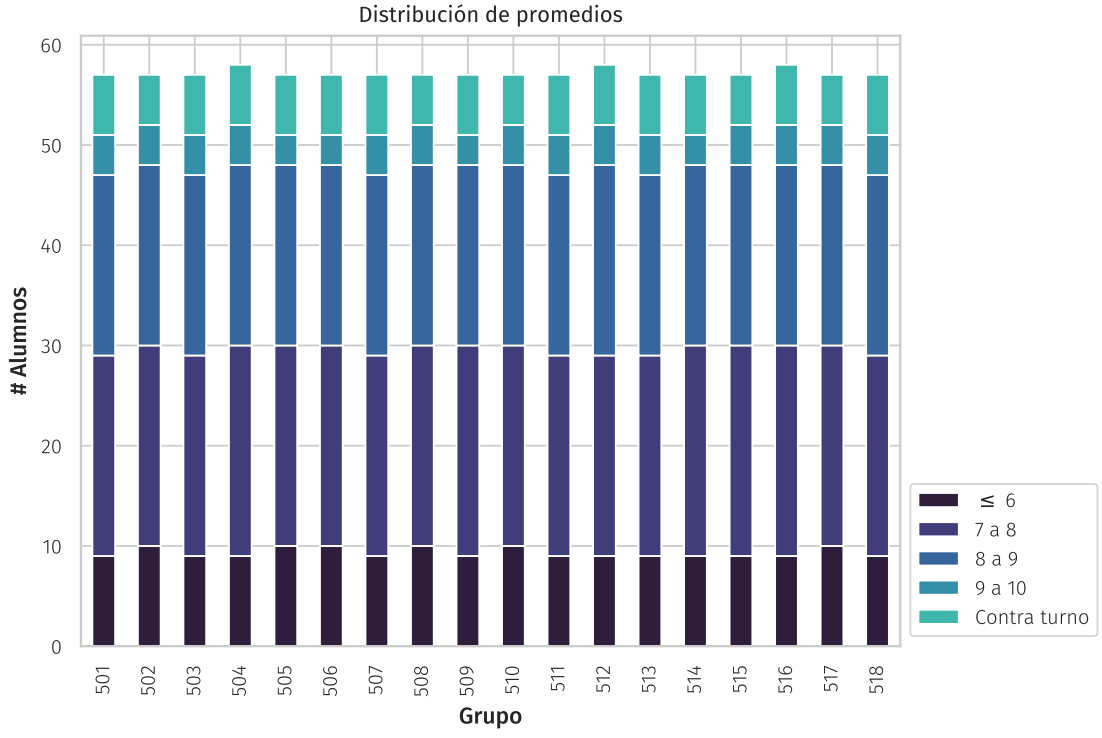


(b) Número de alumnos en un grupo de preferencia h de la mejor solución en cada iteración.

Figura 6.5 – Comportamiento de la búsqueda en cada iteración.

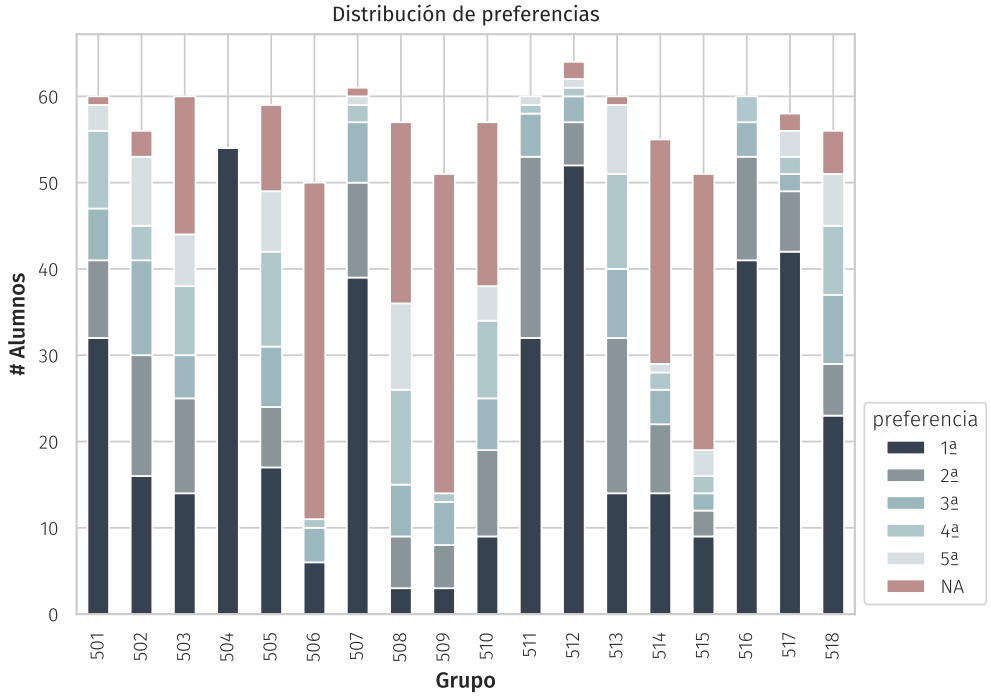


(a) Distribución de promedios en los grupos obtenida por el algoritmo actual.

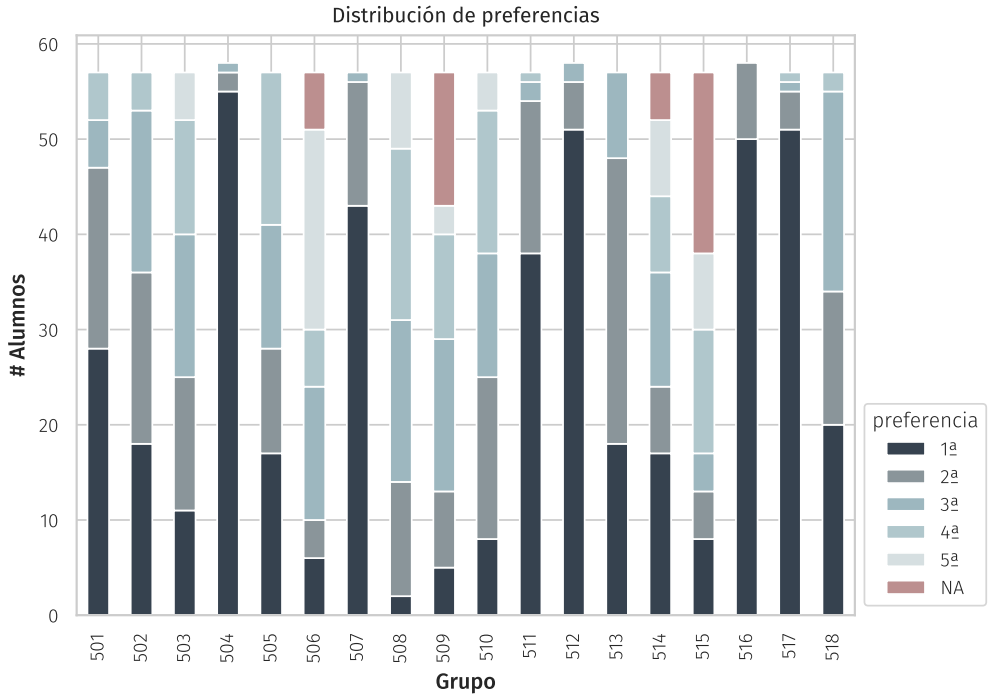


(b) Distribución de promedios en los grupos obtenida por el AFA.

Figura 6.6 – Comparativa de la distribución de promedios en los grupos dada por el algoritmo actual y el AFA.

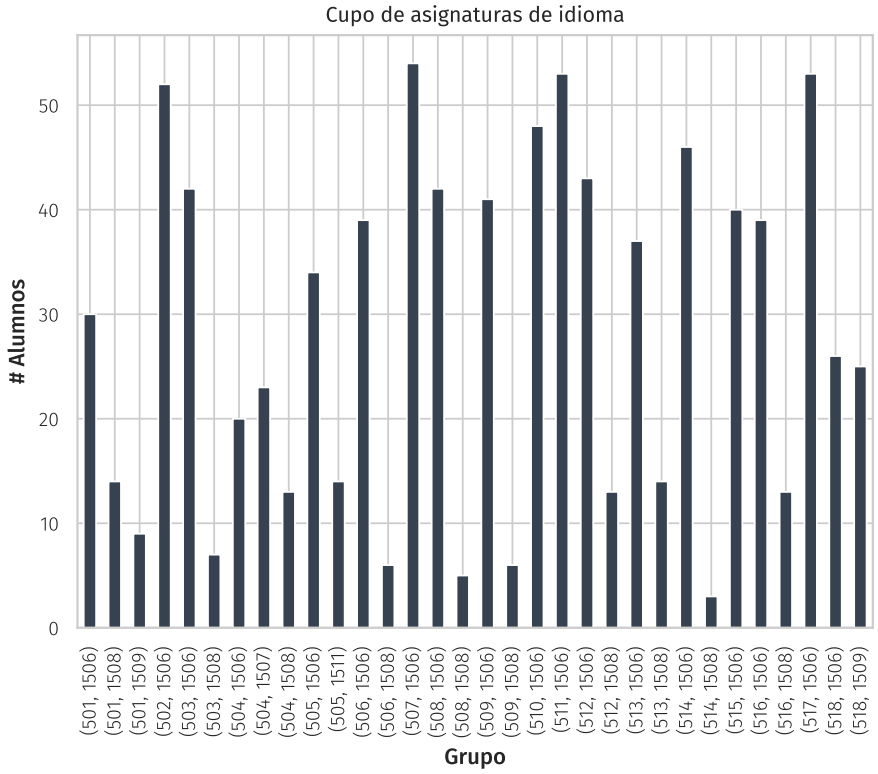


(a) Distribución de preferencias en la asignación obtenida por el algoritmo actual.

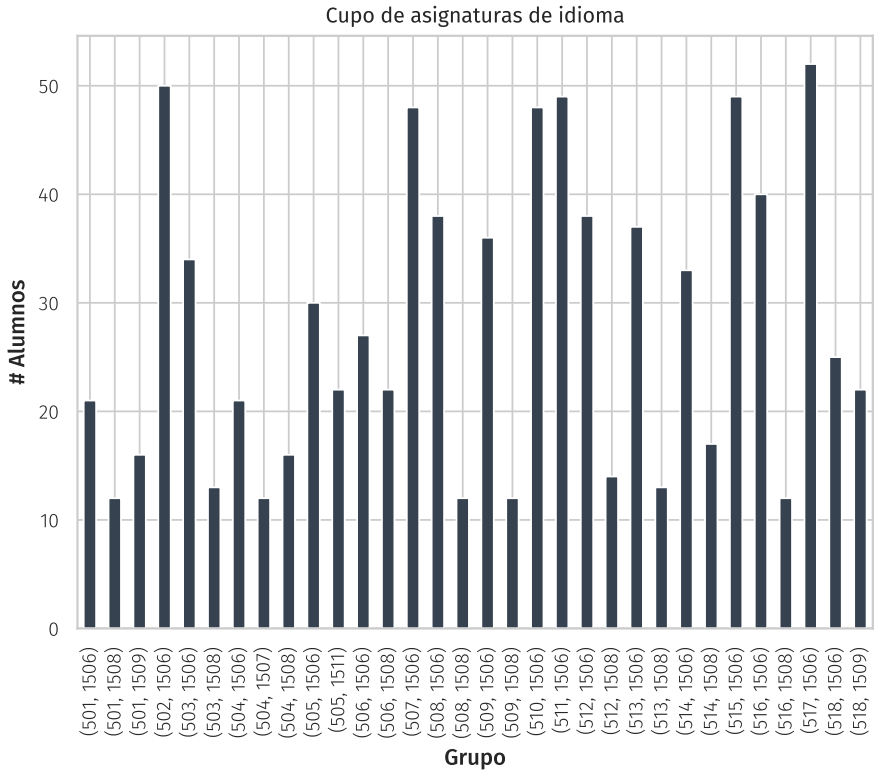


(b) Distribución de preferencias en la asignación obtenida por el AFA.

Figura 6.7 – Comparativa de la distribución de preferencias por el grupo asignado en el algoritmo actual y el AFA.



(a) Número de alumnos en cada idioma de cada grupo obtenida por el algoritmo actual.



(b) Número de alumnos en cada idioma de cada grupo obtenida por el AFA.

Figura 6.8 – Comparativa del cupo de las asignaturas de idioma en el algoritmo actual y el AFA.

Conclusiones

7

En las Ciencias de la Computación es usual trabajar con problemas teóricos; buscamos un problema general y lo estudiamos. El tratar con un problema de la vida real fue interesante pues me llevó a tratar con restricciones y datos que no se suelen tener en un proyecto de clase. Mi primera exposición al problema de asignación de grupos fue cuando iniciaba mis clases de licenciatura, el primer acercamiento fue completamente inocente, sin saber que se trataba de un problema de optimización *NP*–duro. A lo largo de la carrera hay clases con las que de pronto todo hace sentido; materias como Análisis de Algoritmos, Modelado y Programación, Ingeniería de Software y Complejidad Computacional me hicieron ver las fallas del sistema actual de asignación, y en el Seminario de Heurísticas de Optimización Combinatoria vimos problemas de este tipo y cómo atacarlos.

Fue intencional ver las metaheurísticas como un marco algorítmico y analizar los componentes que se suelen tener. Las metaheurísticas han sido usadas exitosamente para resolver problemas de optimización combinatoria aunque en teoría no se tiene una justificación, pues muchos aspectos siguen sin entenderse; por ejemplo, su convergencia. Yagiura y col. [37], Blum y col.[2], Talbi [36] y Taillard y col. [35] dan una comparación conceptual de las metaheurísticas en busca de una vista unificada, de donde se basó la estructura planteada en este trabajo. Es deseo de la autora que el lector analice los algoritmos metaheurísticos bajo esta estructura, dejando a un lado la filosofía de los mismos.

La implementación de un algoritmo metaheurístico no suele tener complicaciones, pero elegir el modelo adecuado e implementar operadores eficientes fue el verdadero reto. Existen alternativas al modelo usado, planteando por ejemplo, el cupo de los grupos como una restricción de penalización o usando otra estructura de vecindario. El análisis de la demanda de los grupos también podría aprovecharse por parte del plantel, identificando los grupos con baja demanda para mejorar la atractividad del mismo. Aún queda trabajo por hacer, este proyecto se orientó a las necesidades de un plantel pero para su uso en otros planteles podrían necesitarse modificaciones para evaluar la calidad de una asignación. Además, se podrían incluir otros factores de diversidad que no fueron contemplados en este trabajo o incluso realizar un análisis del paisaje adaptativo del problema (lo cual sale del alcance de la experiencia de la autora).

Los resultados obtenidos fueron exitosos, pues la satisfacción general de los alumnos fue mejorada sin romper el balance de los grupos. La asignación de grupos en la Escuela Nacional Preparatoria inició siendo un proceso del que nadie sabía, dando origen a heurísticas entre los alumnos, como poner el grupo que de verdad se quería en última

opción. El automatizar el proceso de asignación hizo que tales historias terminaran, pues un algoritmo no debe tener sesgo; los alumnos registran ahora los grupos que quieren en orden real de preferencia, lo que hizo posible tener estadísticas más confiables de la demanda de los grupos. Esta nueva versión busca una asignación más justa a la que el algoritmo glotón provee, además de ser mucho más flexible en las restricciones que se pudieran agregar.

El modelo propuesto fue preferido en las reinscripciones del ciclo escolar 2021 en la ENP 6, pues como parte de las medidas ante la emergencia Covid-19 se buscaban grupos con la misma cantidad de alumnos y que todas las asignaturas de idioma y optativas tuvieran cierto cupo mínimo, con el objetivo de evitar aconglomeraciones en caso de regresar a clases presenciales. El algoritmo actual no tenía la flexibilidad para incorporar estas restricciones, mientras que el implementación del modelo propuesto fue fácilmente adaptado. Los resultados obtenidos fueron satisfactorios y se espera que el uso en el futuro de este trabajo no solo facilite la asignación de grupos, sino que sea de provecho para los alumnos, motivándolos al darles un grupo de su elección.

Problemas de optimización combinatoria

A

El propósito de este apéndice es mostrar las definiciones de algunos de los problemas de optimización combinatoria mencionados en este trabajo.

Emparejamiento (*Matching*): Dada una gráfica G, se busca un conjunto de aristas disjuntas; el objetivo suele ser encontrar el conjunto más grande posible. Un "Emparejamiento perfecto" incluye todos los vértices.

Asignación (*Assignment problem*): Una generalización del problema de emparejamiento en una gráfica bipartita con pesos asociados a las aristas; el peso total del conjunto se maximiza.

Coloración (*Coloring*): Dada una gráfica G, se busca una partición de vértices en k conjuntos independientes asignando un color a cada vértice sin asignar el mismo a vértices adyacentes. El objetivo suele ser minimizar k.

Ruta más corta (*Shortest path*): Dados dos vértices u y v de una gráfica con pesos, se busca la ruta de menor peso entre los vértices u y v.

Árbol generador (*Spanning tree*): Dada una gráfica G, se busca un árbol $F \subseteq G$ con los mismos vértices. Si hay pesos asociados a cada arista, el *árbol generador de peso mínimo* es un árbol generador con el menor peso asociado.

Ruta hamiltoneana (*Hamiltonian path*): Dada una gráfica G, se busca una ruta $P \subseteq G$ que contenga todos los vértices de G.

Problema del Agente Viajero (PAV) (*Traveling Salesman Problem*): Dada una gráfica G con pesos asociados a las aristas, se busca un ciclo $C \subseteq G$ con los mismos vértices y con el menor peso.

Problema de la mochila (*Knapsack problem*): Dada una secuencia de pesos $w_1, \ldots, w_n$, una cota W, y una secuencia de valores $v_1, \ldots, v_n$, se busca un conjunto $K \subseteq \{1, \ldots, n\}$ tal que $\sum_{k \in K} w_k \leq W$; el objetivo es maximizar $\sum_{k \in K} v_k$.

Subproblemas en la asignación de grupos

B

En la Escuela Nacional Preparatoria se consideran tres situaciones escolares en las que pueden estar los alumnos:

1. Alumnos *regulares* que no adeudan ninguna asignatura e inscriben exclusivamente grupos del siguiente año escolar.
2. Alumnos *irregulares* que adeudan a lo más tres asignaturas e inscriben grupos del siguiente año escolar y grupos del año de las materias que se deben. Por ejemplo: un alumno de quinto año que adeude Matemáticas V solicita grupos para sexto año y grupos de quinto año para recursar.
3. Alumnos *repetidores* que adeudan más de tres asignaturas e inscriben grupos de los años que deben. Por ejemplo: un alumno de quinto año que adeude tres asignaturas de quinto y una de cuarto, inscribe grupos de cuarto y quinto año para recursar.

Los grupos solicitados para recursar son a contra turno y se suele asignar un cupo extra, debido a que son alumnos que cursan pocas materias. En el caso de que un alumno repita el año, se asigna grupo en su turno y otro a contra turno en caso de deber asignaturas de años pasados. En el caso de alumnos de sexto año, se inscriben grupos por área de estudio; un caso extraordinario son los alumnos que adeudan asignaturas de tronco común y eligen grupos de todas las áreas. Las asignaturas a inscribir son seriadas, por lo que un alumno que adeude Matemáticas IV no puede cursar Matemáticas V aunque curse todas las demás asignaturas de quinto año.

Para dividir los alumnos considerados en cada vector de asignación se tiene la siguiente estrategia:

1. Cuarto año[1]:
    a) Alumnos que recursan asignaturas en turno contrario.
    b) Alumnos que repiten el año.
2. Quinto año:
    a) Alumnos que recursan asignaturas en turno contrario.

[1] Solo se asignan alumnos que repiten alguna asignatura, debido a que el primer ingreso sigue otro proceso.

b) Alumnos regulares por cada rango de promedio.

3. Sexto año, para cada área:

   a) Alumnos que recursan asignaturas en turno contrario.

   b) Alumnos regulares por cada rango de promedio.

Los rangos de promedio son calculados para: $[0, 5), [5, 6), [6, 7), [7, 8), [8, 9)$ y $[9, 10]$; permitiendo unir dos rangos si el número de alumnos en alguno es menor a 50.

B.1 Limitaciones

1. Una mejora a esta división es restringir el cupo de acuerdo a las asignaturas que se recursan: si solo dos alumnos recursan e inscriben el mismo grupo pero se trata de asignaturas distintas, se les podría dar su primera opción a ambos. Esto también puede aplicarse a los alumnos que repiten año.

2. Los siguientes casos de reinscripción son omitidos:

   a) En sexto año se filtran alumnos que registran opciones en distintas áreas.

   b) Se filtran alumnos que recursan exclusivamente un idioma u optativa, debido a las penalizaciones sobre el cupo.

   c) En caso de que el idioma a cursar por grupo registrado sea inconsistente, se elimina el idioma de las asignaturas consideradas.

   Una vez realizada la asignación se puede volver a estos casos y asignar un grupo siguiendo una estrategia glotona. En general, los alumnos filtrados representan un porcentaje muy bajo en la asignación.

3. En el caso de que un alumno quede asignado a un grupo fuera de sus opciones, se asigna un idioma dentro de sus preferencias de ser posible, en otro caso, se asigna Inglés. Análogamente con las optativas, pero en caso de no haber optativas compatibles en el grupo asignado, se marca como acción manual y no se le asigna ninguna. En sexto año el peso de la penalizacion de idioma es muy bajo, pues en algunos casos, el idioma que cursa un alumno no se encuentra en grupos del área que inscribe; por ejemplo, los grupos que ofrecen Italiano II pertenecen a área I y área IV, por lo que un alumno que inscribe un grupo en área III cursará su idioma en otro grupo.

Implementación en Rust

C

El diseño orientado a objetos en Rust se logra a través de estructuras `structs` similares a C y el comportamiento se define con `traits` similar a una interfaz en Java. A continuación se listan las estructuras y comportamiento dado para la implementación definida en 5.2. Algunas funciones se omiten o se ponen parcialmente pues no son funcionalidad indispensable.

C.1 Flujo de Agua

```rust
pub trait Solution {
  fn get_neighbors(&self, rng: &mut StdRng, n: usize) -> Vec<Self>;

  fn cost(&self) -> f32;

  fn merge(&mut self, another: &Self);

  fn perturbate(&self, rng: &mut StdRng) -> Self;
}

pub trait Waterflow<T: Solution + Ord> {
  pub mass: f32,
  pub velocity: f32,
  pub representation: T
}

impl<T: Solution + Ord> Waterflow<T> {
  pub fn split(&self,
    gravity: f32,
    subflow_limit: i32,
    base_momentum: f32,
    min_velocity: f32,
    rng: &mut StdRng
  ) -> Vec<Waterflow<T>> {

    let subflows = (self.mass * self.velocity / base_momentum).floor() as i32;
    let subflows = min(max(1, subflows), subflow_limit) as usize;
    let mut neighbors = self.representation.get_neighbors(rng, subflows);

    let mut flows = Vec::new();
    let sum_to_n = (subflows * (subflows + 1) / 2) as f32;
    let mut rank = 1.0;
```

```
    while let Some(neighbor) = neighbors.pop() {
      let proportion = rank / sum_to_n;
      let delta = self.representation.cost() - neighbor.cost();
      let momentum = self.velocity.powi(2) + 2.0 * gravity * delta;

      flows.push(Waterflow {
        mass: proportion * self.mass,
        velocity: match momentum {
          zero if zero <= min_velocity => 0.0,
          value => value.sqrt()
        },
        representation: neighbor
      })

      rank += 1.0;
    }

    flows
  }

  pub fn merge(&mut self, flow: &Waterflow<T>) {
    self.mass += flow.mass;
    self.velocity = (self.mass * self.velocity + flow.mass * flow.velocity)
      / (self.mass + flow.mass);
    self.representation.merge(&flow.representation);
  }

  pub fn evaporate(&mut self, factor: f32) {
    let evaporated_mass = self.mass * factor;

    self.mass -= evaporated_mass;
  }
}
```

C.2 Algoritmo de Flujo de Agua

A diferencia de la clase propuesta, se divide en dos estructuras: una con la configuración de los parámetros y otra con la información de la ejecución. Esto debido a detalles de implementación al inicializar una instancia del algoritmo.

```
#[derive(Serialize, Deserialize, Copy, Clone)]
pub struct Parameters {
  pub gravity: f32,
  pub base_momentum: f32,
  pub initial_mass: f32,
  pub initial_velocity: f32,
  pub iteration_limit: i32,
  pub subflow_limit: i32,
```

```
  pub rain_range: i32,
  pub min_velocity: f32,
  pub min_mass: f32,
  pub seed: u64
}

pub struct WaterflowAlgorithm<T: Solution + Ord> {
  parameters: Parameters,
  pub iteration: i32,
  pub population: Vec<Waterflow<T>>,
  pub rng: StdRng
}

impl<T: Solution + Ord> WaterflowAlgorithm<T> {
  pub fn new(parameters: Parameters) -> Self {
    let rng = StdRng::seed_from_u64(parameters.seed);

    WaterflowAlgorithm {
      iteration: 0,
      population: Vec::new(),
      parameters,
      rng
    }
  }

  pub fn init(&mut self, initial_solution: T) {
    self.population = vec![
      Waterflow {
        mass: self.parameters.initial_mass,
        velocity: self.parameters.initial_velocity,
        representation: initial_solution
      }
    ];
    self.iteration = 1;
  }

  pub fn next(&mut self) {
    self.split();
    self.merge();
    self.evaporate();

    if self.iteration % self.parameters.rain_range == 0 {
      self.rain();
    }

    self.iteration += 1;
  }

  pub fn has_next(&self) -> bool {
```

```
    self.iteration < self.parameters.iteration_limit
      && self.population.len() > 0
  }

  pub fn split(&mut self) {
    let mut new_population = Vec::new();

    while let Some(flow) = self.population.pop() {
      if flow.velocity < self.parameters.min_velocity {
        new_population.push(flow);
      } else {
        let mut neighbors = flow.split(...);

        new_population.append(&mut neighbors);
      }
    }

    let all_zero = new_population.iter()
      .all(| flow | flow.velocity <= self.parameters.min_velocity);

    if all_zero {
      self.enforced_rain();
    } else {
      self.population = new_population;
    }
  }

  pub fn merge(&mut self) {
    self.population.sort_by(| x, y | x.cmp(&y));
    self.population.dedup_by(| a, b | {
      let are_equal = a.representation == b.representation;

      if are_equal {
        b.merge(a);
      }

      are_equal
    });
  }

  pub fn evaporate(&mut self) {
    let mut new_population = Vec::with_capacity(self.population.len());
    let factor = 1.0 / self.parameters.rain_range as f32;

    while let Some(mut flow) = self.population.pop() {
      flow.evaporate(factor);

      if flow.mass > self.parameters.min_mass {
        new_population.push(flow);
```

```
      }
    }

    self.population = new_population;
  }

  fn get_total_mass(&self) -> f32 { ... }

  pub fn rain(&mut self) {
    let accumulated = self.parameters.initial_mass - self.get_total_mass();
    let mut new_flows = Vec::new();

    self.population.sort_by(| x, y | x.cmp(y));

    let total_mass = self.get_total_mass();

    for i in 0..self.population.len() {
      let flow = self.population.get(i).unwrap();

      new_flows.push(Waterflow {
        velocity: self.parameters.initial_velocity,
        mass: accumulated * (flow.mass / total_mass),
        representation: flow.representation.perturbate(&mut self.rng)
      });
    }

    self.population.append(&mut new_flows);
    self.merge();
  }

  fn enforced_rain(&mut self) {
    let mut new_population = Vec::new();
    let proportion = self.parameters.initial_mass / self.get_total_mass();

    for flow in self.population.iter() {
      new_population.push(Waterflow {
        velocity: self.parameters.initial_velocity,
        mass: flow.mass * proportion,
        representation: flow.representation.perturbate(&mut self.rng)
      });
    }

    self.merge();
    self.population = new_population;
  }
}
```

Soluciones obtenidas en cada año

D

A continuación se muestran los resultados obtenidos a partir de los parámetros establecidos en la tabla 6.3. Se muestra la instancia; el número de alumnos y grupos; el número de alumnos asignados a un grupo de su h–ésima opción; la cota mínima de alumnos fuera de sus opciones; el número de iteraciones en que la búsqueda se detuvo; y el tiempo de ejecución en segundos. La instancia está codificada de forma que los primeros dos dígitos corresponden al año, los siguientes dos caracteres corresponden al grado y turno, y en caso de sexto, el último dígito corresponde al área; por ejemplo: `19-6m2` corresponde a la instancia de sexto año, área II, turno matutino del 2019. En algunos casos el número de grupos es menor a cinco, por lo que no hay alumnos *fuera de sus opciones*; en estas instancias se marca con un guión la ausencia de la h–ésima opción.

Instancia	N	G	*1a*	*2a*	*3a*	*4a*	*5a*	*NA*	*Cota*	*Iteraciones*	*Tiempo (s)*
15-4m	180	17	103	53	24	0	0	0	0	1,584	49
15-4v	161	16	80	52	25	4	0	0	0	1,315	47
15-5m	1,089	18	481	250	196	87	35	40	29	2,317	879
15-5v	839	14	374	273	142	46	4	0	0	1,682	526
15-6m1	268	5	152	78	14	21	3	-	0	1,668	211
15-6m2	311	6	166	99	35	11	0	0	0	2,490	388
15-6m3	202	6	122	59	14	3	4	0	0	690	80
15-6m4	102	2	77	25	-	-	-	-	0	521	22
15-6v1	144	3	94	33	17	-	-	-	0	610	37
15-6v2	165	3	117	34	14	-	-	-	0	642	49
15-6v3	139	5	72	34	20	12	1	-	0	605	38
15-6v4	84	2	65	19	-	-	-	-	0	672	20
16-4m	162	17	95	42	21	2	1	1	0	1,536	47
16-4v	197	16	109	62	25	1	0	0	0	1,277	44
16-5m	1,057	18	422	239	182	93	60	61	49	1,668	677

16-5v	879	14	483	246	132	18	0	0	0	2,758	886
16-6m1	308	5	207	47	30	23	1	-	0	702	107
16-6m2	307	6	175	81	42	9	0	0	0	1,765	270
16-6m3	198	6	103	55	23	12	5	0	0	1,108	129
16-6m4	103	2	85	18	-	-	-	-	0	520	19
16-6v1	166	3	120	35	11	-	-	-	0	660	50
16-6v2	128	3	78	29	21	-	-	-	0	637	39
16-6v3	145	5	92	40	13	0	0	-	0	756	52
16-6v4	92	2	79	13	-	-	-	-	0	521	17
17-4m	174	17	95	59	19	1	0	0	0	1,629	52
17-4v	187	16	109	60	18	0	0	0	0	1,079	37
17-5m	1,074	18	442	213	178	126	78	37	23	2,318	1,014
17-5v	928	14	467	272	151	31	4	3	2	2,034	762
17-6m1	295	6	163	54	32	38	8	0	0	1,093	155
17-6m2	313	6	144	94	55	15	5	0	0	1,518	242
17-6m3	176	5	101	53	14	1	7	-	0	794	73
17-6m4	116	2	103	13	-	-	-	-	0	529	25
17-6v1	174	3	137	35	2	-	-	-	0	765	58
17-6v2	212	3	109	97	6	-	-	-	0	672	60
17-6v3	157	5	97	50	6	4	0	-	0	768	56
17-6v4	76	2	53	23	-	-	-	-	0	895	23
18-4m	131	17	74	35	22	0	0	0	0	1,496	34
18-4v	143	16	79	37	20	5	2	0	0	942	31
18-5m	1,096	19	436	204	193	120	68	75	52	1,903	810
18-5v	819	14	429	213	126	22	16	13	12	1,858	595
18-6m1	273	6	153	79	24	17	0	0	0	797	110

18-6m2	354	6	241	92	12	4	5	0	0	1,405	266
18-6m3	242	5	170	38	15	18	1	-	0	837	111
18-6m4	119	2	98	21	-	-	-	-	0	526	27
18-6v1	197	3	138	56	3	-	-	-	0	1,030	94
18-6v2	226	4	143	78	4	1	-	-	0	615	64
18-6v3	171	4	110	49	10	2	-	-	0	937	83
18-6v4	95	2	80	15	-	-	-	-	0	640	21
19-4m	122	17	58	40	21	3	0	0	0	1,319	29
19-4v	138	16	58	39	32	9	0	0	0	724	17
19-5m	1,029	18	441	210	164	112	57	45	31	1,793	700
19-5v	861	14	365	257	207	28	4	0	0	1,453	434
19-6m1	260	6	163	62	19	14	2	0	0	1,176	151
19-6m2	370	6	226	109	26	5	4	0	0	1,266	231
19-6m3	263	6	188	46	15	14	0	0	0	742	103
19-6m4	106	2	85	21	-	-	-	-	0	516	22
19-6v1	200	6	157	32	11	0	0	0	0	578	49
19-6v2	187	4	122	42	23	0	-	-	0	689	41
19-6v3	158	4	82	50	6	20	-	-	0	688	51
19-6v4	81	2	56	25	-	-	-	-	0	509	12

Bibliografía

1. Agustín Blas, L. E., Salcedo-Sanz, S., Ortiz-García, E. G., Portilla-Figueras, A. & Pérez-Bellido, Á. M. (2009). A hybrid grouping genetic algorithm for assigning students to preferred laboratory groups. *Expert Systems with Applications*, *36*(3), 7234-7241. https://doi.org/10.1016/j.eswa.2008.09.020

2. Blum, C. & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, *35*(3), 268-308. https://doi.org/10.1145/937503.937505

3. Chiarandini, M., Fagerberg, R. & Gualandi, S. (2019). Handling preferences in student-project allocation. *Annals of Operations Research*, *275*(1), 39-78. https://doi.org/10.1007/s10479-017-2710-1

4. Coello, C. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, *191*(11–12), 1245-1287. https://doi.org/10.1016/S0045-7825(01)00323-1

5. De Maio, A. & Roveda, C. (1971). An all zero-one algorithm for a class of transportation problems. *Operations Research*, *19*(6), 1406-1418. https://doi.org/10.1287/opre.19.6.1406

6. Du, K. & Swamy, M. (2016). *Search and Optimization by Metaheuristics*. Birkhäuser. https://doi.org/10.1007/978-3-319-41192-7

7. Duarte, A., Laguna, M. & Marti, R. (2018). General Concepts in Metaheuristic Search. *Metaheuristics for Business Analytics* (pp. 29-55). Springer, Cham. https://doi.org/10.1007/978-3-319-68119-1_2

8. Gandomi, A., Yang, X.-S., Talatahari, S. & Alavi, A. H. (2013). Metaheuristic Algorithms in Modeling and Optimization. En A. Gandomi, X.-S. Yang, S. Talatahari & A. H. Alavi (Eds.), *Metaheuristic Applications in Structures and Infrastructures*. Elsevier. https://doi.org/10.1016/b978-0-12-398364-0.00001-2

9. Gendreau, M. & Potvin, J. Y. (Eds.). (2010). *Handbook of Metaheuristics* (2.ª ed.). Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1665-5

10. Gendreau, M. & Potvin, J.-Y. (2005). Metaheuristics in Combinatorial Optimization. *Annals of Operations Research*, *140*(1), 189-213. https://doi.org/10.1007/s10479-005-3971-7

11. Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533-549. https://doi.org/10.1016/0305-0548(86)90048-1

12. Hertz, A. & Widmer, M. (2003). Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operational Research*, *151*(2), 247-252. https://doi.org/10.1016/S0377-2217(02)00823-8

13. Hubscher, R. (2010). Assigning Students to Groups Using General and Context-Specific Criteria. *IEEE Transactions on Learning Technologies*, *3*(3), 178-189. https://doi.org/10.1109/TLT.2010.17

14. Institute, C. M. (2019). *P vs NP Problem*. Consultado 15 de septiembre de 2019, desde http://www.claymath.org/millennium-problems/p-vs-np-problem

15. Knuth, D. E. (1974). Computer Programming as an Art. *Commun. ACM*, *17*(12), 667-673. https://doi.org/10.1145/361604.361612

16. Kundakcioglu, O. E. & Alizamir, S. (2009). Generalized Assignment Problem. En A. Floudas & P. M. Pardalos (Eds.), *Encyclopedia of Optimization* (pp. 1153-1162). Springer US. https://doi.org/10.1007/978-0-387-74759-0_200

17. Manfrin, M. (2020). *Metaheuristics Network*. Consultado 1 de junio de 2020, desde http://www.metaheuristics.org/index.php%3Fmain=1.html

18. McConnel, S. (2004). *Metaphors for a Richer Understanding of Software Development* (Second). Microsoft Press.

19. Merriam-Webster.com dictionary. (2020). *"rule of thumb"*. Consultado 1 de junio de 2020, desde https://www.merriam-webster.com/dictionary/rule%20of%20thumb

20. Mohammad-H., T.-N. & Prügel-Bennett, A. (2016). An Analysis of the Fitness Landscape of Travelling Salesman Problem. *Evolutionary Computation*, *24*(2), 347-384. https://doi.org/10.1162/evco_a_00154

21. Natural Sciences and Engineering Research Council of Canada & Department of Combinatorics and Optimization at the University of Waterloo. (2015). *The Problem*. Consultado 15 de agosto de 2020, desde https://www.math.uwaterloo.ca/tsp/problem/index.html

22. Nourmohammadi, A., Fathi, M., Zandieh, M. & Ghobakhloo, M. (2019). A water-flow like algorithm for solving U-shaped assembly line balancing problems. *IEEE Access*, *PP*. https://doi.org/10.1109/ACCESS.2019.2939724

23. Osman, I. H. & Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, *63*(5), 511-623. https://doi.org/10.1007/bf02125421

24. Papadimitriou, C. H. & Steiglitz, K. (1982). Combinatorial Optimization: Algorithms and Complexity. Dover Publications, Inc. https://doi.org/10.1109/TASSP.1984.1164450

25. Powell, S. & Baker, K. (2002). Methods for assigning students to groups: a study of alternative objective functions. *Journal of the Operational Research Society*, *53*(4), 397-404. https://doi.org/10.1057/palgrave/jors/26001307

26. Romero Morales, D. & Romejin, H. E. (2004). The Generalized Assignment Problem and Extensions. En D.-Z. Du & P. M. Pardalos (Eds.), *Handbook of Combinatorial Optimization: Supplement Volume B* (pp. 259-311). Springer Science & Business Media. https://doi.org/10.1007/b102533

27. Ross, G. T. & Soland, R. M. (1977). Modeling facility location problems as generalized assignment problems. *Management Science*, *24*(3), 345-357. https://doi.org/10.1287/mnsc.24.3.345

28. Rothlauf, F. (2011). Design of Modern Heuristics: Principles and Application. Springer. https://doi.org/10.1007/978-3-540-72962-4

29. Rubio, F. & Rodríguez, I. (2019). Water-Based Metaheuristics: How water dynamics can help us to solve NP-Hard problems. *Complexity*, *2019*, 1-13. https://doi.org/10.1155/2019/4034258

30. Schrijver, A. (2003). *Combinatorial Optimization: Polyhedra and Efficiency*. Springer Science & Business Media.

31. Sörensen, K. (2015). Metaheuristics—-the metaphor exposed. *International Transactions in Operational Research*, *22*(1), 3-18. https://doi.org/10.1111/itor.12001

32. Sörensen, K., Sevaux, M. & Glover, F. (2018). A History of Metaheuristics. En R. Martí, P. Panos & M. Resende (Eds.), *Handbook of Heuristics* (pp. 1-18). Springer. https://doi.org/10.1007/978-3-319-07153-4_4-1

33. Srour, A., Othman, Z. A. & Hamdan, A. R. (2014). A water flow-like algorithm for the Travelling Salesman Problem. *Advances in Compuer Engineering, 2014*, 1-14. https://doi.org/10.1155/2014/436312

34. Stützle, T. (1999). *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms, and New Applications* (Tesis doctoral). University of Sankt Augustin.

35. Taillard, É. D., Gambardella, L. M., Gendreau, M. & Potvin, J.-Y. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, *135*(1), 1-16. https://doi.org/10.1016/S0377-2217(00)00268-X

36. Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation* (Vol. 74). John Wiley & Sons. https://doi.org/10.1002/9780470496916

37. Yagiura, M. & Ibaraki, T. (2001). On metaheuristic algorithms for combinatorial optimization problems. *Systems and Computers in Japan*, *32*(3), 33-55. https://doi.org/10.1002/1520-684x(200103)32:3<33::aid-scj4>3.0.co;2-p

38. Yang, F.-C. & Wang, Y.-P. (2007). Water Flow-like Algorithm for Object Grouping Problems. *Journal of the Chinese Institute of Industrial Engineers*, *24*(6), 475-488. https://doi.org/10.1080/10170660709509062

39. Yang, X.-S. & Koziel, S. (2011). Computational Optimization: An Overview. En S. Koziel & X.-S. Yang (Eds.), *Computational Optimization, Methods and Algorithms*. Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-642-20859-1

40. Zainudin, S., Kerwad, M. M. & Othman, Z. A. (2015). A Water flow-like algorithm for Capacitated Vehicle Routing Problem. *Journal of Theoretical and Applied Information Technology*, *77*(1), 125-135.