Universidad Nacional Autónoma de Méexico

Facultad de Ciencias

Algoritmos genéticos (jav: aplicados a la cosmologia?)

T E S I S

que para obtener el título de:

Licenciado en Ciencias

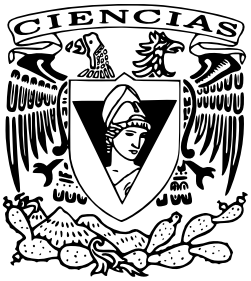
presenta:

Alejandro Alfredo Morales Sánchez

Tutor:

José Alberto Vázquez González

CIUDAD UNIVERSITARIA CD.MX 2021



Dedicatoria ...
(jav: escribir/terminar todo)

Agradecimientos

(jav: escribir/terminar todo)

Resumen

hay typos: El presente trabajo tiene como finalidad explorar la cosmología computacional, su estrecha relación con la cosmologíia observacional y su potencial al relacionarla con la rama de la inteligencia artificial: la computación evolutiva.

(jav: Definir acronimos, o poner una breve explicacion.)La formación de estructura cosmológica ha sido objeto de estudio desde antes del descubrimiento de la expansión del Universo. El proceso de evolución conlleva a proponer la existencia de elementos hasta ahora desconocidos, la materia oscura y la energía oscura, ambos descritos en el modelo cosmológico estándar: ΛCDM ((jav: definir brevemente)). Posteriormente surge la pregunta de como describir los diferentes componentes del universo dando origen a extensiones del modelo ΛCDM, como son la energía oscura con ecuación de estado variable en el tiempo.

El método aquí descrito utiliza la habilidad de algoritmos genéticos para identificar los parámetros, dado un modelo matemático, que mejor describen las observaciones.

El capítulo 1 está dedicado a presentar la motivación de este trabajo. (jav: y el cap 2?) El capítulo **??** presenta una introducción a grandes rasgos de los conceptos biológicos relevantes en este trabajo.

Posteriormente en capítulo 2 se presentan los puntos más importantes de la optimización matemática además de una serie de algoritmos de optimización metaheurística.

El capítulo 3 se explica en profundidad el funcionamiento teórico de los algoritmos ge-

néticos. Es esta misma sección se presenta un algoritmo para la optimización de funciones de N dimensiones y se compara con algoritmo de optimización puramente matemática.

(jav: mover y cambiar) Más adelante en capítulo **??** se presenta los antecedentes históricos y teóricos de la cosmología, para posteriormente describir dos aplicaciones de los algoritmos genéticos, la primera para el ajuste de datos y la segunda para utilizarlos como método de validación.

Notación

Índice general

Índice de figuras

Capítulo 1

Introducción

Hace 50 años, gracias a uno de los proyectos más ambiciosos en la historia de la ciencia, fue posible decodificar el genoma humano. Hoy en día comparado con la colección de genomas de microorganismos viviendo en nuestro cuerpo, en el océano, todos nuestros alrededores, un genoma humano, el cual puede ser codificado en un DVD, se vuelve simple. Sus 3 billones de bases de DNA y cerca de 20,000 genes son insignificantes comparados con las 100 billones de bases y genes que uno puedo encontrar en los microbios en el cuerpo humano. En la rama de la física, cada año, la organización europea para la investigación nuclear, también conocida como CERN, produce aproximadamente 115 petabytes de datos. Este es un problema actual de la ciencia de hoy en día, y existe una gran cantidad de datos esperando a ser explorados, entre ellos podemos encontrar $H(z)$, Baryon acoustic oscillations (BAO), supernovas tipo Ia (SNIa), brote de rayos gamma (GRB) por mencionar algunos, pero el problema es ¿Cómo? (jav: definir acronimos antes de usarlos LISTO).

La naturaleza es rica en conocimiento y gran parte del desarrollo de la especie humana se debe al gran trabajo de conocerla e imitarla. No hay duda alguna de que no existe otro mecanismo tan avanzado que se haya perfeccionado tanto así mismo como la naturaleza. Es por esta misma razón que una gran cantidad de problemas han sido resueltos, inspirándose en diseños y comportamientos biológicos. La razón es muy sencilla, esta ha descubierto lo que funciona y lo que perdura [Benyus, 2012].

Con los grandes avances tecnológicos, una de las ramas que más se ha beneficiado es la computación. Por esta razón desde hace dos décadas, la biomímesis o biomimética (imitación de la vida) junto con el poder computacional actual, han presentado un papel muy importante en la resolución de problemas y explotación de datos. Con este desarrollo técnicas computacionales como Redes Neuronales Artificiales y el cómputo evolutivo comenzaron a popularizarse en la industria. Sin embargo, estas técnicas computaciones han sido utilizadas por la comunidad científica para conocer el pasado y explorar el futuro de nuestros alrededores, nuestro planeta y universo.

Es difícil darse una idea de la inmensidad del universo, por eso no es sorpresa encontrar un gran volumen de información que los científicos han recopilado a lo largo de los años. Uno de los primeros campos que comenzaron con la explotación de datos haciendo uso de algoritmos computacionales fueron la astronomía. Esta relación datos-análisis se remonta hasta el siglo XVI, antes de la existencia de telescopios. Cuando Tycho Brahe y Johannes Kepler trabajaron en conjunto. Tycho hizo mediciones cinco veces más precisas que sus contemporáneos, creando un conjunto de datos limpio. Posteriormente colaboró con Kepler y, usando rigor matemático, los datos observacionales fueron convertidos en descubrimientos científicos. De esta manera Kepler y Tycho, desarrollaron métodos científicos y algoritmos para crear las leyes de Kepler. En la actualidad existe una gran cantidad de observatorios astronómicos y telescopios espaciales construidos por diferentes agencias espaciales, estos son los encargados de suministrar los datos. El análisis es realizado por diferentes instrumentos computacionales entre los más famosos se encuentran software como HEALPix [Gorski et al., 2005], códigos como CMBFAST [Seljak and Zaldarriaga, 1999], CAMB [Lewis and Bridle, 2002], CMBEASY [Doran, 2005] y CLASS [Blas et al., 2011] o paquetes como CosmoMC [Lewis and Bridle, 2003] y SimpleMC por mencionar algunos (jav: agregar referencias FALTA SIMPLE MC).

En la era actual uno puede encontrar diferentes herramientas computacionales para encontrar información relevante. El objetivo de (jav: que? LISTO)este trabajo consiste en encontrar los parámetros cósmicos más relevantes de ciertos modelos cosmológicos con la

ayuda de algoritmos genéticos.

Capítulo 2

Optimización matemática

2.1. Introducción

En muchas disciplinas la optimización juega un papel clave. En la física, los sistemas son llevados a su estado de energía más baja según las leyes físicas que lo rigen. En los negocios, las empresas buscan maximizar el precio de sus acciones. En biología, los organismos más óptimos tienen una mayor (jav: que?) de sobrevivir [Kochenderfer and Wheeler, 2019]. Ingenieros espaciales utilizan simulaciones computacionales para optimizar la aerodinámica de un misil o avión. Científicos farmacéuticos diseñan experimentos para obtener la máxima información de un nuevo medicamento [James, 2003]. Dado su importancia en la industria, no es novedad encontrar un gran interés por encontrar nuevas y mejores técnicas de optimización.

2.2. Antecedentes

El campo de la optimización es uno que ha sido estudiado por muchos años. Se tienen registros con una gran cantidad de problemas de optimización, por ejemplo alrededor de 300 A. C., Euclides descubrió que un cuadrado tiene la mayor área entre todos los rectángulos con el mismo perímetro. Alrededor de 100 A.C. Herón de Alejandría propuso que la luz viaja a lo largo del camino geométricamente más corto [Cillero Fernando, 2010]. Hoy se sabe que esto es falso (jav: pero si lo hace, y se les llama geodesicas), sin embargo se

convirtió en un problema de optimización para los años posteriores y el cuál sería resuelto por Fermat con el principio del mismo nombre [Xin-She, 2010].

El astrónomo alemán Johannes Kepler perdió a su esposa en 1611 debido al cólera, y comenzó a buscar una nueva compañera de vida, por lo que en 1613 encontró la solución óptima al famoso problema de la secretaria [Macía Vázquez, 2020] [Ferguson, 1989]. En 1687 en sus *Principia Mathematica* Newton resolvió el problema de la forma del cuerpo con menor resistencia que él mismo había propuesto en 1685, el cual se convirtió en un problema optimización muy importante en cálculo de variaciones. El principal objetivo consistía en encontrar la forma de un sólido de revolución para minimizar su resistencia a través de un fluido homogéneo con velocidad constante en la dirección del eje de revolución.

En 1696 Johann Bernoulli retó a sus contemporáneos para encontrar la curva que conecta dos puntos a diferentes alturas, tales que un cuerpo caiga sobre dicha curva en el menor tiempo posible. Actualmente se conoce como el problema de Brachistochrone.

En 1746 Pierre-Louis Moreau de Maupertuis propone su principio de mínima acción, en el cual plantea que en todos los fenómenos naturales se minimiza una cantidad llamada "acción". En términos actuales dicho principio se conoce como principio de Hamilton. En 1781 Gaspard Monge, enfocó su energía y tiempo, en resolver el problema de optimizar la transportación y distribución de mercancías. Posteriormente en 1942 Leonid Kantorovich demostró que este es un problema de optimización combinatoria.

Frederich Gauss, en 1801, utilizó el método de mínimos cuadrados para predecir la orbita del asteroide Ceres. Sin embargo en 1805 Adrien Legendre describió de manera rigurosa dicho método en el apéndice de su libro *Nouvelle Meethodes pour la Determination des Orbites des Cometes*. Un año después utilizó el mismo principio para el ajuste de una curva. Gauss afirmaba que el mismo había creado el método en 1795 y en 1808 Robert Adrain publica el mismo método pero tomando en cuenta los errores observacionales.

David Ricardo propuso su Ley de los rendimientos decrecientes para el cultivo de tierras en 1815, de la cual derivó la Ley del Costo de Oportunidad Creciente. En 1847 Augustin-Louis Cauchy propuso un método general para resolver sistemas de ecuaciones de manera iterativa, lo cual construyó las bases para dos de los método iterativos más famosos que existen el método del gradiente y el método del descenso más pronunciado.

A principios del siglo XX Johan Jensen introduce el concepto de convexidad y deduce la famosa desigualdad de Jensen la cual es pieza clave en la optimización convexa y económica. Para 1930 Karl Menger plantea en un coloquio de Viena el problema del agente viajero (o famosamente conocido como el problema TSP) [Cummings, 2000]. Dicho problema consiste en encontrar el camino más corto entre un conjunto finito de ciudades. En 1917 Harris Hancock publica el primer libro sobre optimización "Theory of Minima and Maxima" [Hancock, 1917].

Continuando con los avances en el campo de la optimización en 1939 Leonid Kantorovich desarrollo el primer algoritmo de programación lineal para identificar la óptima asignación de los recursos y métodos de fabricación, por el cual obtuvo el premio nobel en 1975. En 1947 George Dantzig crea el método símplex para la resolución de problemas de programación lineal. La siguiente década, en 1957, Richard Bellman desarrolla la programación dinámica y la ecuación de Bellman para la toma de decisiones.

En el siglo XX con el avance tecnológico de la computación se dio origen a una gran cantidad de algoritmos de optimización, por esta misma razón a partir de 1960 la literatura sobre dicho tema empezó a ver un incremento como nunca antes.

(jav: relacionado con los comentarios anteriores, por que necesitas algoritmos de optimizacion en cosmologia? Agregar un intro)

2.3. Optimización Estocástica

En el siguiente capítulo se presenta una técnica de optimización con algoritmos genéticos es por esta razón que necesitamos explicar, a grandes rasgos, estos métodos de optimización.

Se le conoce como optimización estocástica al conjunto de técnicas y algoritmos que emplean cierto grado de aleatoriedad para encontrar la solución más óptima para un problema dado. Los algoritmos metaheurísticos son los más generales de este grupo y tienen aplicaciones en una gran variedad de campos [Bianchi et al., 2008].

La optimización metaheurística es aplicada a aquellos problemas de los cuales tenemos poca información, donde no sabemos nada respecto a la solución. Uno podría preguntarse si un algoritmo de este tipo es equivalente a utilizar fuerza bruta. La respuesta es no, los primeros exploran aleatoriedad en su búsqueda para posteriormente ser guiada de alguna manera, la fuerza bruta no cuenta con ningún tipo de guía y en espacios muy grande pueden llegar a ser muy costosos computacional y temporalmente.

2.3.1. Optimización con gradientes

Antes de profundizar en los métodos metaheurísticos, es importante conocer uno de los métodos de optimización matemática más importante que existen. El descenso o ascenso del gradiente (ver Algoritmo 1). La idea es muy sencilla, identificar la pendiente y bajar o subir según si uno quiere encontrar el mínimo o máximo respectivamente. Este método no requiere evaluar $f(x)$, pero asume que uno puede calcular la pendiente de x, en otras palabras, asume la existencia de $f'(x)$ [Andrychowicz et al., 2016].

Tomando el problema de maximización, nosotros empezamos con un punto aleatorio para x. Posteriormente sumamos repetidamente una pequeña parte de la pendiente a dicho punto, $x \leftarrow x + \alpha f'(x)$, donde α es un valor positivo muy pequeño. Si la pendiente es positiva x aumentará, si es negativa disminuirá. El algoritmo se detiene cuando x esta en

Algoritmo 1 Ascenso del gradiente

$\vec{x} \leftarrow$ vector inicial
repeat
  $\vec{x} \leftarrow \vec{x} + \alpha \nabla f(\vec{x})$ ▷ En una dimensión $x \leftarrow x + \alpha f(x)$
Hasta que $\vec{x}$ sea una solución ideal o se cumplió el tiempo límite de ejecución
Return $\vec{x}$

el punto más alto, en aquel punto donde la pendiente es cero y x no podrá cambiar de valor.

Usualmente los problemas que uno busca resolver son multidimensionales, para escalar el algoritmo anterior a n dimensiones solo basta con reemplazar x, por un vector $\vec{x}$ y reemplazar la pendiente $f'(x)$ con el gradiente de $\vec{x}$, $\nabla f(\vec{x})$.

Cabe destacar que el algoritmo corre hasta que hemos encontrado la "solución ideal" o "se cumplió el tiempo límite de ejecución". ¿Cuándo sabemos que hemos encontrado la solución ideal? Generalmente cuando la pendiente es cero. Sin embargo hay puntos diferentes del máximo, donde la pendiente también tiene un valor de cero, conocidos como óptimos locales. Igualmente tenemos otro tipo de puntos conocidos como puntos de silla o montura, que son aquellos donde no tenemos ningún tipo de máximo pero si una pendiente de cero (ver Figura 2.1).

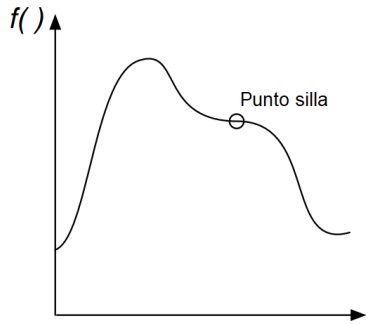


Figura 2.1: Representación de punto silla.

El problema con este tipo de algoritmos es el tiempo de convergencia. Mientras nos acercamos al máximo, existe la posibilidad de saltar al otro lado del óptimo, saltando de un lado a otro hasta converger. Este problema puede ser resuelto ajustando el valor de α por un valor pequeño, de esta manera esquivaremos accidentalmente el máximo pero se

necesitará un mayor número de evaluaciones para converger.

2.3.2. Métodos de estado simple

La optimización con gradientes asume que uno puede calcular la primera, o incluso la segunda, derivada. En la mayoría de los casos, uno no puede calcular dichas derivadas porque no contamos con una función matemática explicita para representar el problema, si no que deseamos realizar un ajuste a una serie de datos, queremos la mejor combinación de activos para un portafolio y aunque evaluemos (jav: ?). Lo único con lo que contamos es una manera de crear o modificar las entradas de la función para verificar su calidad en términos del problema [Luke, 2013].

Todo lo que tenemos es una problema de caja negra, la cual describe el problema de optimización. Esta toma una posible solución y devuelve la calidad de la solución en términos del problema. Estos problemas son complicados porque, en ocasiones, no hay manera de obtener información gráfica, la solución puede no consistir de números, vectores, estos pueden ser una estructura de decisiones, un conjunto de reglas, una palabra. La solución es aquella que sea apropiada para el problema.

Para encontrar una solución son necesarias las siguientes cuatro condiciones:

- Proporcionar uno o más posibles soluciones. Este es conocido como el proceso de **inicialización**.
- Evaluar la calidad de las posibles soluciones. Esto es conocido como el proceso de **evaluación**.
- Generar copias de las posibles soluciones.
- Modificar las soluciones, esto produce un conjunto de nuevas soluciones aleatorias, las cuales son un poco diferentes a las primeras. Esta condición sumado con la generación de copias es conocido como el proceso de **modificación**.

2.3.3. Ascenso de colina

El algoritmo de ascenso de colina o escalada simple, mayormente conocido por su nombre en inglés Hill-Climbing (ver Algoritmo 2), es similar al ascenso del gradiente pero no necesita información de derivadas o dirección, solo se necesita evaluar nuevas soluciones en la región del candidato actual y adoptar las nuevas si son mejores. Esto permite "escalar" la colina hasta encontrar el óptimo local.

Algoritmo 2 Ascenso de colina

```
S ← solución candidata                                  ▷ Inicialización repeat
  R ← modificación(copia(S))                            ▷ Modificación
  if Calidad(R) >Calidad(S) then
    S ← R
  end if
until S sea una solución ideal o se cumplió el tiempo límite de ejecución
Return S
```

Si uno compara ambos algoritmos podrá darse cuenta de sus similitudes. La única diferencia real radica en el enfoque estocástico (parcialmente aleatorio) para buscar las mejores soluciones.

El significado de modificar

El proceso de inicialización, evaluación y modificación son los encargados de crear la representación de las posibles soluciones. En conjunto estas operaciones pueden verse como una fábrica, la cual se encarga de crear soluciones siguiendo un grupo de reglas, las cuales generarán la apariencia y comportamiento en términos del problema. Lo importante en este tipo de problemas es cumplir las cuatro condiciones antes mencionadas.

Para ilustrar mejor el significado de modificar tomemos uno de los ejemplos más generales, la representación de soluciones como vectores de longitud fija con valores reales. Crear un vector de estas características es sencillo, solo tendremos que seleccionar número aleatorias dentro de nuestras límites. Tomemos min y max como el límite inferior y superior y l la longitud del vector (ver Algoritmo **??**) (jav: ?).

Algoritmo 3 Generación de vector aleatorio

$min \leftarrow$ límite inferior
$max \leftarrow$ límite superior
$\vec{v} \leftarrow$ un nuevo vector $\langle v_1, v_2, ..., v_n \rangle$
  for i from 1 to l **do**
    $v_i \leftarrow$ número aleatorio seleccionado entre min y max.
  end for
Return $\vec{v}$

Para modificar un vector uno puede sumar una pequeña cantidad de ruido aleatorio a cada número, siguiendo la definición de modificación, asumamos que el ruido a sumar es un valor pequeño. Para cada posición de nuestro vector generaremos un número aleatorio entre 0 y 1, si este número es menor a nuestra probabilidad p realizaremos la suma. Este enfoque es utilizado para modificar posibles soluciones en la mayoría de los algoritmos metaheurísticos.

Los algoritmos presentados anteriormente resuelven problemas matemáticos de una manera eficiente, sin embargo cuando tenemos poca información del problema empiezan a fallar en la localización del óptimo. Por esta razón se comenzaron a desarrollar diferentes algoritmos más avanzados, inspirados en diferentes conceptos, entre ellos la biología y física.

2.3.4. Métodos de población

Los métodos de población difieren de los métodos de estado simple en el hecho que los primeros guardan una **muestra** de las soluciones candidatas a solo una. Cada uno de las soluciones será sometida al proceso de modificación, pero lo que previene a estos de convertirse en un ascenso de colina paralelizado es el hecho que las mismas soluciones guían la exploración. Un ejemplo de esto sería el efecto de las mejores soluciones rechazando aquellas con un peor desempeño o modificándolas para generar mejores.

No es sorpresa que la mayoría de estos métodos están basados en conceptos biológicos, cuyo conjunto recibe el nombre de computación Evolutiva, los cuales son basados en

conceptos como la genética y evolución. Entre estos algoritmos podemos encontrar los algoritmos genéticos (GA) y estrategias evolutivas (ES), los cuales son generacionales. Esto significa que modifican a la muestra en cada iteración, sin embargo también podemos encontrar versiones de estado firme o estable, los cuales modifican solo unos candidatos de toda la muestra [Beheshti and Shamsuddin, 2013].

La computación evolutiva son técnicas de remuestreo: nuevas muestras son generadas a partir de los resultados de las anteriores. El algoritmo evolutivo básico consiste de crear una población inicial, seguido de tres pasos. Primero evaluar a la población, posteriormente crear una nueva generación según la evaluación anterior y por último colocar los nuevos individuos en la muestra original.

2.3.5. Estrategias evolutivas

Esta familia de algoritmos fue desarrollada en 1960 y 1970 por Ingo Rechenberg y Hans-Paul Schwefel en la Universidad Técnica de Berlín [Rechenberg and Eigen, 1973, Schwefel., 1977]. Estrategias Evolutivas (por su siglas en inglés ES) se caracterizan por utilizar un método de selección llamado Selección truncada que solo utiliza la mutación como operador de modificación.

Entre estos algoritmos podemos encontrar el (μ, λ) (ver Algoritmo 4). Empezamos con una muestra de λ soluciones generadas aleatoriamente. Primero evaluamos las soluciones, luego eliminamos todas excepto las μ mejores. Cada uno de los μ_i individuos produce λ/μ soluciones hijas, a través de métodos de modificación, de esta manera creamos λ que serán utilizados como las nuevas soluciones en la siguiente iteración.

De forma resumida λ es el número de soluciones que sobreviven y μ es el número de soluciones hijas. Es importante aclarar que λ tiene que ser un múltiplo de μ.

En los métodos de población existen una gran variedad de procesos de modificación, sin

Algoritmo 4 Estrategia evolutiva (μ, λ)

$\mu \leftarrow$ número de soluciones seleccionadas
$\lambda \leftarrow$ número de soluciones hijas generadas
$P \leftarrow [\,]$
  for i from 1 to l **do**
    $P \leftarrow P$ y nueva solución aleatoria
  end for
$mejor \leftarrow$ x
  for P_i in P **do**
    Evalucion(P_i)
    if Calidad(P_i) >Calidad(mejor) **then**
     mejor $\leftarrow P_i$
    end if
  end for
  Q $\leftarrow$ los μ mejores individuos
  P $\leftarrow [\,]$
  for Q_i in Q **do**
    for λ/μ **do** $P \leftarrow P$ y modificacion(copia(Q_j)
    end for
  end for
until *mejor* sea una solución ideal o se cumplió el tiempo límite de ejecución
Return Mejor Solución

embargo hay dos que destacan más que otros, la **mutación**, la cual consiste de pequeños cambios, los cuales modifican a los individuos (soluciones) y la **recombinación**, proceso en el que se seleccionan múltiples individuos los cuales serán combinados para obtener un hijo.

Este algoritmo tiene tres parámetros que pueden ser ajustados para explorar o explotar el espacio de soluciones. Por explorar nos referimos a hacer una búsqueda de todo el espacio, explotación consiste en enfocar la búsqueda en las zonas más prometedoras.

- Tamaño de λ. Este parámetro controla el tamaño de la población. Cuando λ tiende a ∞ tendremos exploración. Básicamente una búsqueda aleatoria.

- Tamaño de μ. Este parámetro controla que tan selectivo es el algoritmo. Valores pequeños de μ respecto a λ, provoca que la búsqueda sea más explosiva, ya que solo los individuos más aptos sobrevivirán.

- El grado de mutación (modificación). Si la mutación es muy alta, las nuevas soluciones serán aleatorias sin importar la selectividad de μ.

2.3.6. Optimización por enjambre de partículas

Optimización por enjambre de partículas o por sus siglas en inglés PSO, es un algoritmo de optimización estocástica similar a los algoritmos evolutivos pero con la diferencia de que este es modelado a partir de comportamientos de enjambre o rebaños de animales. A diferencia de otros métodos de población, PSO no vuelve a muestrear a la población para generar una nueva, este algoritmo mantiene una población estática cuyos individuos son modificados según los descubrimientos del espacio. Está técnica fue desarrollada por James Kennedy y Russell Eberhart a mediados de los años 90's [Kennedy and Eberhart, 1995].

PSO opera exclusivamente un espacio multidimensional con valores reales, esto se debe a que el método muta a las soluciones candidatas hacia las mejores soluciones descubiertas. Dado que se utilizan valores reales y que este algoritmo esta inspirado en enjambres y rebaños nos referiremos al conjunto de soluciones como enjambre de partículas. Estás partículas exploran el espacio a partir de la mutación. Una partícula esta compuesta de dos partes:

- La ubicación de la partícula en el espacio $\vec{x} = \langle x_1, x_2, ..., x_n \rangle$.
- La velocidad de la partícula $\vec{v} = \langle v_1, v_2, ..., v_n \rangle$. Esta es la rapidez y dirección a la que viaja la partícula. Si $\vec{x}^{t-1}$ y $\vec{x}^{t}$ son las ubicaciones de una partícula al tiempo $t-1$ y t respectivamente, entonces al tiempo t, $\vec{v} = \vec{x}^{t} - \vec{x}^{t-1}$.

Cada partícula empieza en una ubicación y con una velocidad aleatoria. Es importante registrar:

- La mejor ubicación de $\vec{x}^{*}$ que $\vec{x}$ ha descubierto.
- La mejor ubicación de $\vec{x}^{+}$ que cualquiera de los informantes $\vec{x}$ ha descubierto. Los informantes son un conjunto de partículas seleccionadas aleatoriamente.

- La mejor ubicación de $\vec{x}^!$ que cualquiera de las partículas ha descubierto.

En cada iteración tenemos que realizar las siguientes operaciones:

- Evaluar el desempeño de cada partícula y de ser necesario actualizar las ubicaciones.
- Determinar el proceso de mutación. Para cada partícula $\vec{x}$, actualizamos el vector de velocidad $\vec{v}$ sumando un vector apuntando a $\vec{x}^*$, a $\vec{x}^+$ y a $\vec{x}^!$ en pequeñas cantidades.
- Mutar cada partícula a lo largo de su vector se velocidad.

Esta implementación depende de cinco parámetros:

- α que tanta velocidad es retenida.
- β que tanto del mejor desempeño personal es retenido y mezclado en las futuras iteraciones. Si β es muy grande las partículas se moverán más hacia su mejor desempeño personal en lugar de mejor desempeño global.
- γ que tanto del mejor desempeño de informantes es retenido y mezclado en las futuras iteraciones.
- δ que tanto del mejor desempeño global es retenido y mezclado en las futuras iteraciones.
- ϵ la velocidad de la partícula. Si este parámetro es alto se moverá más rápido hacia mejores zonas.

2.3.7. Sistema de colonia de hormigas

Este algoritmo fue presentado por primera vez en 1992 por Marco Dorigo [Dorigo, 1992]. Sistema de colonia de hormigas (por sus siglas en inglés ACO) toma un enfoque de optimización combinatoria, en este caso la modificación se vuelve opcional y la sustituye por una selección de componentes las cuales competirán por atención 6.

Algoritmo 5 Optimización por enjambre de partículas

$size \leftarrow$ tamaño de enjambre
$\alpha \leftarrow$ velocidad
$\beta \leftarrow$ mejor personal
$\gamma \leftarrow$ mejor informante
$\delta \leftarrow$ mejor global
$\epsilon \leftarrow$ tamaño del salto
$P \leftarrow [\,]$
$\overrightarrow{Mejor} \leftarrow$
  for size **do**
    $P \leftarrow P$ y nueva partícula con ubicación $\vec{x}$ y velocidad $\vec{v}$ aleatoria
    for cada partícula $\vec{x}$ en P **do**
      Evaluar($\vec{x}$)
      if Evaluacion($\vec{x}$) >Evaluacion($\overrightarrow{Mejor}$) **then**
        $\overrightarrow{Mejor} \leftarrow \vec{x}$
      end if
    end for
    for cada partícula $\vec{x}$ en P **do**
      Evaluar($\vec{x}$)
      $\vec{x}^{*} \leftarrow$ mejor ubicación de anterior de $\vec{x}$
      $\vec{x}^{+} \leftarrow$ mejor ubicación de anterior de informantes de $\vec{x}$
      $\vec{x}^{!} \leftarrow$ mejor ubicación anterior de cualquier partícula
      for cada dimensión **do**
        $b \leftarrow$ número aleatorio entre [0, β]
        $c \leftarrow$ número aleatorio entre [0, γ]
        $d \leftarrow$ número aleatorio entre [0, δ]
      end for
      for cada partícula $\vec{x}$ en P **do**
      $\vec{x} \leftarrow \vec{x} + \epsilon\vec{v}$
      end for
    end for
  end for
until $\overrightarrow{Mejor}$ sea una solución ideal o se cumplió el tiempo límite de ejecución
Return $\overrightarrow{Mejor}$

La fuente de inspiración de este algoritmo es el camino de feromonas que las hormigas utilizan como medio de comunicación. Como analogía dicha técnica se encuentra basada en el método de comunicación indirecta de una colonia de agentes simples, llamados hormigas artificiales, regulados por los caminos de feromonas. Estos caminos sirven como una distribución de información numérica que las hormigas utilizan para construir, probabilísticamente, soluciones para el problema. Las hormigas se adaptarán durante la ejecución para reflejar su experiencia en la búsqueda [Dorigo and Stützle, 2006].

Algoritmo 6 Colonia de hormigas

$C \leftarrow \{C_2, C_2, ..., C_n\}$
$size \leftarrow$ tamaño de enjambre
$\vec{p} \leftarrow$ un nuevo vector $\langle p_1, p_2, ..., p_n \rangle$
$\overrightarrow{Mejor} \leftarrow 0$
repeat
  $P \leftarrow size$ creación de rastros a partir de feromonas
  for P_i from P **do**
    $P_i \leftarrow$ Ascenso de colina opcional P_i
    if Evaluacion(P_i)>Evaluacion($mejor$) **then**
     $mejor \leftarrow P_i$
    end if
    Actualizar componentes de $\vec{p}$ basado en las evaluaciones de cada P_i en P
  end for
until $mejor$ sea una solución ideal o se cumplió el tiempo límite de ejecución
Return mejor

En este algoritmo una feromona puede ser vista como la calidad histórica de todos los rastros a la que el componente ha pertenecido. Las feromonas nos informan sobre que tan bueno es seleccionar un componente sobre otro sin importar de su valor o costo computacional.

Capítulo 3

Algoritmos Genéticos

3.1. Introducción

Una parte importante de la tecnología de nuestra época tiene su base en la imitación de la naturaleza, a tal grado que a la estrategia de imitar los diseños y los procesos biológicos para resolver problemas humanos se le ha asignado un nombre propio: biomímesis o biomimética. El punto de partida de esta técnica es la suposición razonable de que, después de 3800 millones de años de evolución, la naturaleza ha descubierto lo que funciona, lo que es apropiado y lo que perdura [Benyus, 2012]. En el caso de la computación, las áreas más importantes de la Inteligencia Artificial (las Redes Neuronales Artificiales y el Cómputo Evolutivo) siguen, como sus nombres así lo indican, las estrategias de la biomímesis de manera transparente.

Esta técnica de imitación dio origen a una rama de la computación, conocida como computación bioinspirada [Bagavathi and Saraniya, 2019]. Esta se puede dividir en tres grupos principales (ver Figura 3.1), los algoritmos evolutivos, inteligencia de enjambre y ecología. En el primer grupo se encuentran los algoritmos genéticos, las estrategias evolutivas, evolución diferencial entre otros. El segundo replica el comportamiento de diferentes especies para reproducirse, como las luciérnagas [Yang, 2010] o bacterias [Passino, 2002], o los patrones para conseguir alimento de una manada de lobos [Mirjalili et al., 2014] o un banco de peces. El tercero se inspira en el comportamiento ecológico, como lo puede

ser la biogeografía [Simon, 2008], el crecimiento de hierba o la simbiosis. Existe un último, el cual combina ambos fenómenos, entre ellos tenemos algoritmos como: campo de arroz, sistema de río natural y salto de rana [Riquelme Medina and Luna, 2014], a dicho grupo se le conoce como grupo mixto.

Los Algoritmos Genéticos (AG) suelen definirse como métodos heurísticos de búsqueda inspirados en lo que sabemos acerca del proceso de la evolución natural, los cuales han resultado ser también métodos muy potentes de optimización. Al igual que otras técnicas heurísticas, los AG se han popularizado debido a que permiten abordar problemas en los que es muy difícil aplicar procedimientos matemáticos tradicionales, como la optimización basada en el cálculo de derivadas. De tal manera que han sido aplicados con éxito a problemas de matemáticas, planificación, diseño, estadística, aprendizaje o control automático, entre otros.

3.2. Algoritmos Genéticos como un modelo de la evolución de las especies

Los AG están inspirados en la teoría darwiniana de la evolución de las especies, por ello conviene comenzar estableciendo una síntesis de la misma que nos permita aprovecharla para establecer la terminología y los operadores que son aprovechados en la técnica computacional.

3.2.1. Selección Natural y Genética: una síntesis

Al hablar de evolución no hay otro nombre que resalte más en nuestra cabeza que el de Charles Darwin. Darwin fue un biólogo inglés conocido por sus estudios pioneros respecto a la evolución. En su libro El Origen De Las Especies, publicado en 1859, estipula que el proceso de evolución es una consecuencia de la selección natural [Darwin, 1877].

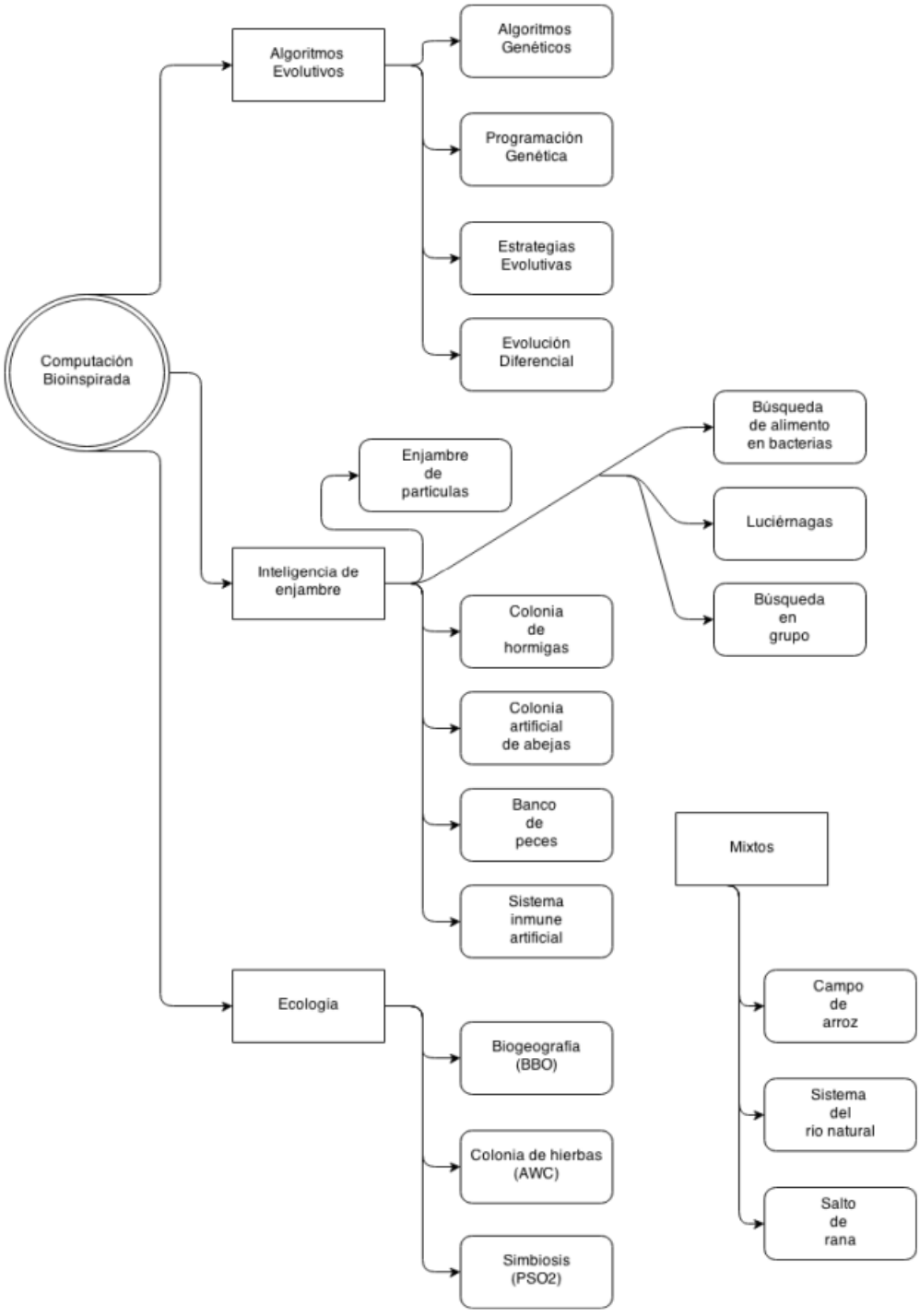


Figura 3.1: Tipos de bioalgoritmos [Riquelme Medina and Luna, 2014].

La selección natural es un proceso de competencia por los recursos, en el cual los organismos con los rasgos más aptos tendrán una mayor probabilidad de sobrevivir, teniendo una mayor descendencia, por lo que se aumenta la frecuencia de sus características en las siguientes generaciones. Los organismos son sistemas abiertos que operan sobre la base de una serie de instrucciones codificadas en sus genomas. Durante la reproducción los genomas parentales son replicados para producir nuevas copias en la siguiente generación, pero la replicación no es perfecta y las mutaciones heredadas producen, continuamente, novedades genéticas. Además la reproducción sexual implica una recombinación de cromosomas para dar origen a nuevas variedades, por lo que los individuos, adquieren, gracias a este mecanismo, variaciones de genomas, lo que da origen a una serie de nuevos rasgos [Kutschera, 2017]. Al conjunto de individuos se le conoce como población, estas habitan en ecosistemas y es en este lugar donde diferentes poblaciones compiten por los recursos (espacio, comida etc.). Cada organismo, tiene una serie de características únicas, las cuales le proporcionan diferentes habilidades que pueden ser explotadas para la obtención de esos recursos, es decir, maximizar su probabilidad de sobrevivir, lo cual se traduce en una maximización del éxito reproductivo (Fitness Darwiniano), en un nicho ecológico especifico.

La evolución es un fenómeno de poblaciones, individuos nacen, se reproducen y mueren, mientras que la población evoluciona. Estudios morfológicos, genéticos y bioquímicos han demostrado que poblaciones las cuales se expanden albergan una gran cantidad de variaciones y esta variabilidad es la base para la evolución biológica [Kutschera, 2017]. La evolución actúa como un potenciador lo que permite que una población sea diversa para que perdure adaptandóse a futuros cambios y riesgos ambientales.

La variabilidad sirve como material en "crudo" para la creación de nuevas especies. Si los miembros de una población aislada de moscas se reproducen entre ellos más que con otros miembros de su especie, tendremos que su perfil genético (ADN) divergiría, lo que significa que: nuevas mutaciones se originarían y la selección natural se encargaría de propagarlas o eliminarlas dependiendo de su aportación a su supervivencia. Sin embargo

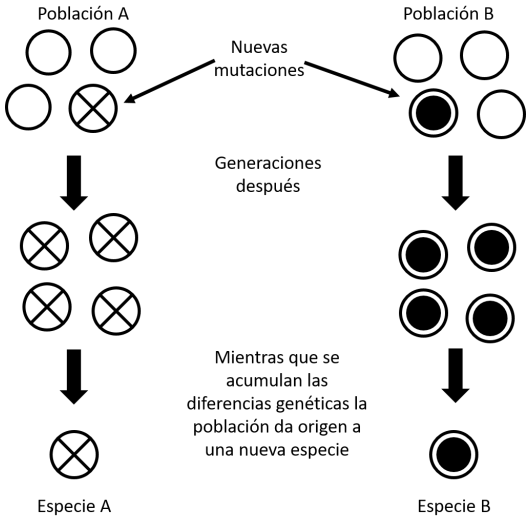


Figura 3.2: Representación de incompatibilidad de poblaciones aisladas a lo largo del tiempo. (jav: explicar que debo ver aqui, .. y tambien poner una pequena explicacion en el texto.)

dado que la población de moscas solo se esta dando dentro de la misma, las mutaciones no podrían ser propagadas al resto de la especie. La población aislada se volvería cada vez más genéticamente distinta y se puede dar el caso en el que sus nuevos genes sean incompatibles con los genes de las moscas ajenas a su población. Si el aislamiento se da por un período largo de tiempo las moscas podrían perder su capacidad de reproducirse con otras moscas o sus descendientes híbridos serían estériles (Fig 3.2), pero si el grupo de moscas termina su aislamiento en el tiempo adecuado, la reproducción se llevaría a cabo sin ningún problema dando origen a una nueva especie [Dobzhansky, 1951].

Los encargados de transmitir estos rasgos son los **cromosomas**, en estos se almacena toda la información genómica o genética. Un genoma está compuesto por un conjunto de cromosomas, los cuales están formados por una serie de genes, que se encuentran compuestos por una serie de ácidos nucleicos. Podemos ver esta relación como un conjunto de muñecas rusas. En los humanos y otros organismos eucariontes (organismos compuestos por una o más células con núcleo y organelos) que se reproducen sexualmente, tendremos que se encuentran emparejados, heredando características de cada padre. Las células humanas contienen 23 pares de cromosomas, para un total de 46. Este diploide (células que

contienen dos copias de cada cromosoma) transmiten sus cromosomas en dos procesos de división celular: mitosis y meiosis [Rosenberg and Rosenberg, 2012]. En la primera, división que ocurre en todas las células de nuestro cuerpo (células somáticas), cada "célula hija" recibe un cromosoma del padre. En la segunda, la cual ocurre en células germinales (células reproductoras), el diploide 23 es dividido, produciendo un haploide (una célula u organismo con un único conjunto de cromosomas).

La variación heredada entre dos organismos se da gracias a dos eventos los cuales ocurren durante la fase de meiosis: la separación aleatoria de los cromosomas y la combinación de cromosomas.

Los genes se encuentran compuestos por ácido desoxirribonucleico (ADN), estos almacenan la información en moléculas lineales de ADN, las cuales comprenden cuatro bases núcleicas, G (guanina), C (citosina), A (adenina) y T (timina), en conjunto estas bases forman los nucleótidos. Cada gen tiene una secuencia de nucleótidos, los cuales le brindan una función única. El ADN es una hélice compuesta por dos hilos, cada uno de estos se encuentra compuesto por bases pares de A-T o G-C.

Cada uno de ellos se compone de ácido ribonucleico (RNA). Los cromosomas se dividen en diferentes partes, estas reciben el nombre de genes, cada una de ellas contiene un rasgo físico, por lo que, distintas combinaciones de genes, generarán variedad de individuos. Tales combinaciones se dan a partir de la reproducción celular, mitosis o meiosis.

¿Cómo se transmiten estos rasgos? La respuesta más actual la ofrece la síntesis moderna, la cual añade la explicación genética de la herencia a la selección natural "e identifica a las mutaciones genéticas como fuente de variabilidad de las especies" [García Ortega,]. La síntesis moderna es una rama de la biología que describe a la evolución en términos de la variación genética. Dicha variación se puede dar por diferentes factores, la recombinación y la mutación.

La recombinación genética se refiere al reordenamiento de las secuencias de ADN por el rompimiento y reincorporación de cromosomas o segmentos de cromosomas. Como consecuencia de dichos arreglos, tendremos nuevas combinaciones las cuales presentarán nuevas características en las siguientes generaciones. Este proceso también es conocido como reproducción sexual dado que se encuentra "programado" en la mayor parte de procesos los cuales involucran meiosis en la mayoría de la reproducción de organismos. Este fenómeno también sirve como un sistema de reparación para daños, potencialmente, letales a cromosomas [Carroll, 2013].

La mutación es un cambio permanente en la secuencia genética, sin importar si tienen efectos notables en el organismo. Una mutación sucede cuando, se presentan errores al copiar el material genético en el proceso de división de una célula, por exposición a radiación ionizante, exposición a mutágenos (agentes físicos o químicos los cuales cambian permanentemente el material genético) o infección por virus. Con frecuencia las mutaciones presentan efectos perjudiciales en la habilidad de que un gen funcione correctamente, las consecuencias pueden ser enfermedad o muerte del organismo mutado. Aquellas mutaciones con efectos pequeños o que ocasionalmente presentan beneficios pueden volverse puntos claves en el desarrollo de la secuencia genética [Ripley, 2001].

3.2.2. El Algoritmo Genético Simple

Los algoritmos genéticos pueden describirse como algoritmos de búsqueda estocástica dirigida, basados en los mecanismos de la selección natural y la genética. Estos fueron presentados por John Holland en 1975, en su libro Adaptation in Natural and Artificial Systems [Holland, 1992] y desarrollados por él, sus colegas y estudiantes de la Universidad de Michigan, principalmente, David E. Goldberg y Kenneth A. De Jong. En un principio, la investigación de Holland tenía los objetivos de abstraer y explicar rigurosamente los procesos de adaptación de los sistemas naturales, además de diseñar sistemas artificiales que emularan aquellos mecanismos de adaptación más importantes en los sistemas naturales [Goldberg, 1989]. Debido a que requieren de un cómputo intensivo los AG no se explotaron de inmediato, pero actualmente la disponibilidad de computadoras más poderosas ha pro-

piciado que se usen cada vez más en una gama muy amplia de problemas de optimización.

Los AG tienen como objetivo principal encontrar la solución óptima de un problema. Estos son normalmente utilizados en problemas los cuales involucran una representación matemática compleja, problemas sin representación matemática, problemas con mucho ruido o problemas cuyo ambiente cambie con el tiempo. Uno puede encontrar aplicaciones de los algoritmos genéticos en campos como en la medicina para identificar aquellas regiones de interés en mamogramas y para diferenciar tumores de pecho malignos y benignos en imágenes de ultrasonido [ghaheri et al., 2015]. En la geotecnia para determinar la máxima discontinuidad de frecuencia [Simpson and Priest, 1993]. Igualmente los algoritmos genéticos han sido utilizados en la predicción de precios de mercancías [Drachal and Pawłowski, 2021] e incluso en la optimización de un reactor industrial [de S. Victorino and Filho, 2006].

Regresando a la analogía entre la selección natural y los algoritmos genéticos, tendremos que cada **individuo** representará una posible solución. El genotipo o información genética, contenido en los cromosomas de cada individuo será representado por una cadena de caracteres, por ejemplo, cada bit representará un gen como se muestra en Fig 3.3 y Fig 3.4 . Podríamos decir que los cromosomas contienen la información en crudo sobre las posibles soluciones al problema. El fenotipo o la representación física del genotipo será la expresión en términos de este.

Los Algoritmos Genéticos hacen uso de una versión simplificada del proceso de evolución biológica, como la siguiente: La selección natural es el proceso mediante el cual los individuos mejor adaptados a su entorno tienden a producir más descendientes, en promedio, que sus competidores menos adaptados, y está dirigida por dos ingredientes clave: 1) herencia: los padres "transmiten"de alguna manera su aptitud a su descendencia; y 2) variación: la existencia de un espectro de aptitud en una población dada.

Para comprender mejor este proceso, debemos comenzar estableciendo la distinción entre **genotipo** y **fenotipo**. El primero hace referencia a la composición genética de

un individuo, codificada y almacenada en forma de secuencias lineales de **genes** dentro de sus **cromosomas**; el segundo es la manifestación externa del genotipo, es decir, las características del individuo que determinan su aptitud de adaptación. A grandes rasgos, el fenotipo es simplemente una versión decodificada de su genotipo, y esta decodificación ocurre durante el proceso que llamamos crecimiento o desarrollo del individuo (durante el cual, por supuesto, existen influencias ambientales no incluidas en el genotipo). Una aptitud fenotípica alta se traduce, en promedio, en éxito reproductivo alto y por ende en una influencia directa en la reserva genética de la próxima generación, lo que conduce a una proporción cada vez mayor de fenotipos de alta aptitud en la población. Para un ambiente de condiciones fijas, la población convergerá de manera natural al genotipo de aptitud más alta presente en la reserva genética inicial. La rapidez de este proceso de convergencia es una función de **la presión de selección**, es decir, la relación entre aptitud y éxito reproductivo.

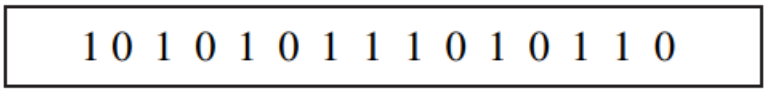


Figura 3.3: Representación de cromosoma. Como se puede observar cada cromosoma se encuentra representado por 1 y 0 (codificación binaria).

Al conjunto de individuos se le conoce como **población**, esta tomará valores dentro del **espacio de búsqueda**. En este espacio cada punto representará una posible solución al problema. La solución más óptima será encontrada al combinar diferentes cromosomas para dar origen a individuos más aptos. En los algoritmos genéticos hay dos aspectos muy importantes de la población: **población inicial** y **tamaño de población**. La población inicial son aquellos primeros individuos que serán evaluados. Necesitamos tener la mayor diversidad posible, por esta misma razón serán elegidos aleatoriamente. El tamaño dependerá siempre de la naturaleza del problema pero deberá de ser lo más grande posible para abarcar el espacio de búsqueda con mayor eficacia.

Cada individuo en la población se le asignará un número para conocer que tan bueno o malo es para resolver el problema dado. Esta medida cuantitativa es asignada por una

función mejor conocida como **función fitness** o **función objetivo**. La búsqueda de las mejores soluciones será guiada por dicha función. La función fitness no solo representa la fortaleza de las soluciones sino que también nos da información sobre que tan cerca se encuentra un cromosoma de ser el óptimo.

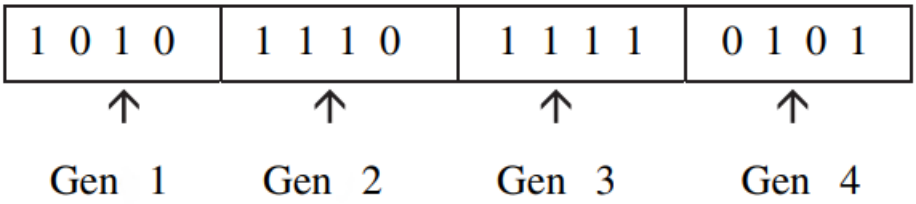


Figura 3.4: Representación de gen en cromosoma. Se puede observar los diferentes cromosomas, compuestos por cuatro cromosomas en codificación binaria.

Al proceso de la representación de genes se le denomina **codificación**. En los AG uno puede representar los genes como bits, números, árboles, arreglos, listas o diferentes objetos. Para ilustrar un poco más este concepto tomemos unos ejemplos. En la codificación binaria cada cromosoma esta codificado por una cadena binaria, lo que significa que solo podrá tener valores de 0 o 1, en la codificación octal podrá tener valores enteros de 0 a 7. En el caso de la codificación de árbol cada gen es un árbol que puede representar funciones, comandos o instrucciones.

La característica más importante de los algoritmos genéticos es la **crianza**. Este proceso consta de cuatro pasos: **Selección**, **Recombinación**, **Mutación** y **Reemplazo**.

La selección de estos parámetros, junto con su valor, dependerá de la naturaleza de cada problema y la mayoría de las veces estos son seleccionados a mano. Sin embargo existe una herramienta de análisis denominado “análisis de sensibilidad”. Dicho análisis busca identificar la influencia de cada parámetro en las soluciones. Esta técnica nos puede ayudar a contestar la siguiente pregunta: ¿Qué factores provocan la mayor y menor incertidumbre en el resultado? Esto mide la importancia de los factores en el modelo analizado [Pinel et al., 2011].

3.2.3. Selección

Es el proceso de "seleccionar" uno, dos o más padres de la población. El propósito de esta es contar con los mejores individuos para que, de esta manera, la descendencia obtenga un valor más alto de fitness con el paso de las generaciones. Los cromosomas son seleccionados de la población inicial para reproducirse.

Este proceso selecciona cromosomas aleatoriamente según el fitness de cada individuo. Entre mejor sea el fitness mayor será la probabilidad de ser seleccionado, esto permite que el proceso de búsqueda sea guiado, a este proceso se le conoce como presión selectiva. Igualmente entre mayor sea la presión selectiva, tendremos que los individuos seleccionados serán más aptos.

Es esta misma presión la que lleva a encontrar las soluciones más óptimas. Una presión muy baja provocará que el algoritmo tome más tiempo del necesario para encontrar la solución óptima, presión muy alta aumenta la probabilidad de obtener un óptimo local. Una presión adecuada lleva al AG a mejorar el fitness de la población con el paso de las generaciones.

Existen diferentes métodos de selección como lo son:

- Selección de ruleta: El principio de este método se basa en una búsqueda lineal en una ruleta en la cual cada espacio tendrá un peso proporcional al fitness del individuo (Figura 3.5).
- Selección aleatoria: Se eligen los padres de la población con una probabilidad igual para cada individuo. Esto significa que si tenemos una población de 10 individuos cada uno tendrá un 10 % de probabilidad de ser seleccionado (Figura 3.6).
- Selección de ranking: Este método le asigna rangos a la población y los ordena de mejor fitness al peor, según la naturaleza del problema. Es una alternativa a selección de ruleta en el caso de que la diferencia de fitness sea muy grande. Si el mejor

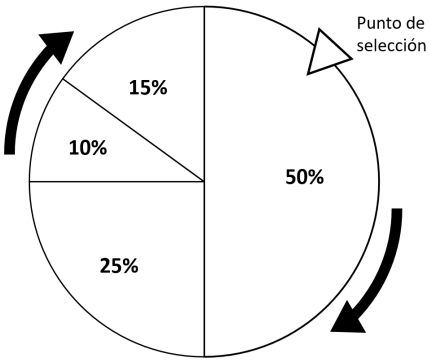


Figura 3.5: Representación de selección de ruleta. El individuo con mayor fitness tendrá una mayor probabilidad de ser seleccionado.

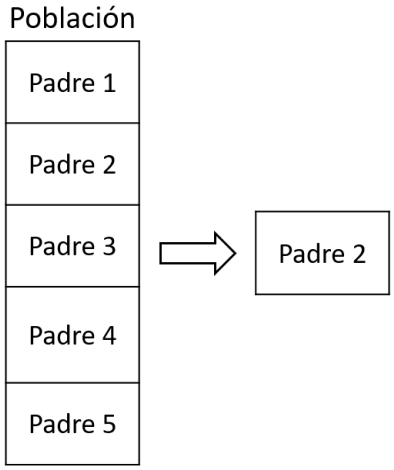


Figura 3.6: Representación de selección de aleatoria. Cada individuo tendrá la misma probabilidad de ser seleccionado.

cromosoma tiene un fitness del 90 % su circunferencia en la ruleta será muy grande provocando así poca diversidad en la población, problema que puede ser evitado con esta técnica.

- Selección de torneo: Como su nombre lo indica se realiza una competencia entre un número N de individuos de la población seleccionados al azar, se comparará su fitness y el más apto será seleccionado para reproducirse. Se repite este proceso hasta que tengamos el número de padres necesarios para la nueva descendencia. Este método asegura un mejor fitness global, lo que lo hace más eficiente.

- Selección de entropía de Boltzman: Este método no se basa en el fitness de los

individuos, si no en la entropía y métodos de muestreo de una simulación de Monte Carlo [Lee, 2003].

3.2.4. Recombinación

El siguiente proceso es la recombinación, la cual consiste en tomar dos padres y reproducirlos para tener un hijo. Como vimos anteriormente la selección genera buenos individuos pero con poca variabilidad total en la población. El objetivo de la recombinación es generar una mayor diversidad.

Existen diferentes métodos de recombinación como lo son:

- Recombinación de un punto: Es utilizada por la mayoría de algoritmos genéticos, en este tipo de recombinación se eligen dos cromosomas y se cortan en el mismo punto (línea punteada Figura 3.7). Una vez divididos se intercambia el material genético para dar origen a la siguiente generación.

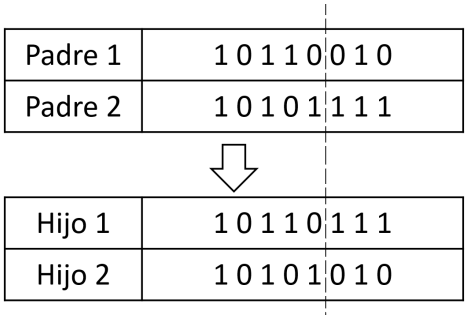


Figura 3.7: Recombinación de un punto.

- Recombinación de dos puntos: A partir de la recombinación de un punto surgieron más clases de recombinaciones. Análogamente al punto anterior se eligen dos puntos para cortar al cromosoma y se intercambian los cromosomas.

- Recombinación de N puntos: Es el caso generalizado de los dos métodos de recombinación anteriores y existen dos maneras de implementarlo, con números pares o impares. En el primer caso los cromosomas se ordenan en un círculo y se intercambian

al azar. En el segundo, se recombinan cromosomas que no se hayan intercambiando antes.

- Recombinación uniforme: En este tipo de recombinación se seleccionan cromosomas de cualquiera de los dos padres de acuerdo a una máscara binaria aleatoria de la misma longitud que los cromosomas. Cuando el valor de dicha máscara sea 1 se copiará la información del primer padre, en el caso de que sea 0 se copiará del segundo. Una máscara es generada aleatoriamente de los dos padres. Por lo que la descendencia será una combinación de ambos genes (Figura 3.8).

Padre 1	1 0 1 1 0 0 1 1
Padre 2	0 0 0 1 1 0 1 0
Máscara	1 1 0 1 0 1 1 0
Hijo 1	1 0 0 1 1 0 1 0
Hijo 2	0 0 1 1 0 0 1 1

Figura 3.8: Recombinación uniforme.

- Recombinación de tres padres: En esta técnica de recombinación se eligen tres padres al azar. Cada gen del primer padre es comparado con el gen del siguiente. Si el valor es el mismo este gen será el elegido para la descendencia.

- Recombinación aleatoria: Esta relacionado con la recombinación uniforme. Se selecciona un punto para realizar un corte, pero antes de intercambiar la información, esta se combinará aleatoriamente entre dos padres.

- Recombinación ordenada: Este método utiliza el principio de la recombinación en dos puntos. Se seleccionan dos padres y se cortan en dos puntos aleatorios, creando tres partes, la izquierda, central y derecha. La información de esta partición será distribuida de la siguiente manera, el primer hijo heredará la información de las partes izquierda y derecha del padre uno. La sección del centro será determinada por los genes del padre uno pero con el orden de los valores del segundo padre (Figura 3.9) [Deep and Adane, 2011].

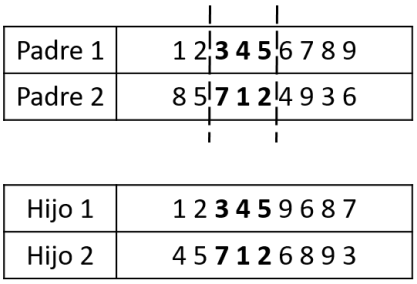


Figura 3.9: Recombinación ordenada.

El parámetro principal en la recombinación es la probabilidad de recombinación P_c. Este parámetro describe el ritmo al cual una recombinación será realizada. En el caso donde $P_c = 100$ toda la descendencia es producto de recombinación, en el otro extremo $P_c = 0$ la descendencia serán copias exactas de los cromosomas padres.

3.2.5. Mutación

Después de la recombinación los individuos son sometidos al proceso de mutación. Este proceso es el encargado de salvar al algoritmo de quedar atascado en un óptimo local. Se podría ver como aquel proceso disruptivo encargado de recuperar información perdida en el proceso de búsqueda además de perturbar al algoritmo. El operador de mutación es el encargado de mantener diversidad en la población para así buscar más soluciones en el espacio y evitar óptimos locales. Existen diferentes tipos de mutaciones:

- Flipping: Este método de mutación consiste en tomar un cromosoma de mutación, comparar al padre con este y en el caso de que los valores sean los mismos se sustituye ese valor por 0 y en el caso contrario por 1 (Figura 3.10).

Padre	**1** 0 1 1 **0** 1 0 **1**
Cromosoma de mutación	**1** 0 0 0 **1** 0 0 **1**
Hijo	**0** 0 1 1 **1** 1 0 **0**

Figura 3.10: Mutación tipo flipping.

- Intercambio: Se eligen dos posiciones al azar y se intercambian sus valores.

- Reversing: Se selecciona una posición al azar y los valores de la izquierda y derecha son intercambiados.

- Inverso: Consiste en una variación del método anterior, se toma una secuencia aleatoria de genes y se invierten el orden de los genes.

- Scramble: En este tipo de mutación igualmente se toma una secuencia al azar y aleatoriamente se intercambian las posiciones.

3.2.6. Reemplazo

El último paso para concluir el proceso de crianza consiste del reemplazo, el cual es un método donde se seleccionan dos padres de la población para obtener un par de hijos pero los cuatro individuos no pueden regresar a esta, por lo que dos serán reemplazados. Existen diferentes métodos para reemplazar individuos pero todos tienen que cumplir una regla: la población se tiene que mantener constante a lo largo del periodo de ejecución del código. Existen tres métodos para ejecutar este mecanismo:

- Reemplazo fuerte: En este método se reemplazan dos individuos de la población al azar. En el reemplazo fuerte tanto los padres como los hijos pueden participar. Esto es de utilidad para realizar una búsqueda más completa en el espacio con poblaciones pequeñas. Uno es libre de seleccionar un número n de participantes, pero lo común es seleccionar dos para mantener la mayor diversidad en los individuos.

- Reemplazo débil: Del conjunto padres-hijos se seleccionan los dos individuos más aptos. Este proceso mejora el fitness general de la población (ver Figura 3.11). Como se mencionó anteriormente se suelen seleccionar dos participantes para mantener la mayor diversidad en la población.

- Ambos padres: Es el más sencillo de todos los métodos, los hijos reemplazan a los padres. La desventaja de este método radica en que favorece a los padres con mayor fitness.

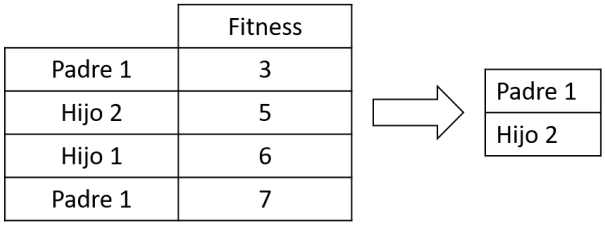


Figura 3.11: Reemplazo débil para problema de optimización.

3.2.7. Criterios de convergencia

Otro concepto muy importante en los algoritmos genéticos es el **criterio de convergencia**. Al cumplir este criterio el algoritmo detendrá el proceso de búsqueda. Este dependerá del objetivo y poder computacional. Los criterios más comunes son:

- Número de generaciones: El algoritmo genético se detiene cuando se alcanza el número máximo de generaciones.
- Tiempo de ejecución: El algoritmo genético se detiene al transcurrir una cantidad especifica de tiempo.
- Fitness estacionario: El algoritmo genético se detiene cuando no hay ningún cambio en el mejor fitness de la población a un número específico de generaciones.
- Stall de generaciones: El algoritmo se detiene si no se presenta ninguna mejora en la función objetivo para una secuencia consecutiva de generaciones.
- Stall de tiempo: El algoritmo se detiene si no se presenta ninguna mejora en la función objetivo en un intervalo de tiempo especificado.

3.3. ¿Por que los algoritmos genéticos funcionan?

La búsqueda heurística de los AG está basada en el teorema del esquema de Hollad (ver Apéndice A), el cual nos menciona que existen esquemas (patrones) que se encuentran dentro de los cromosomas. Cada esquema representa un subconjunto de cromosomas que son similares entre sí. Esto significa que las mejores opciones tienen mejores bloques

constructores (combinaciones de bits), por lo que esquemas de bajo orden, bien definidos y con fitness promedio serán recombinados para producir un esquema de mayor orden y un fitness mayor que el promedio. Aquí es donde toma relevancia la hipótesis de bloques de construcción. Este es uno de los criterios más importantes de como funciona un algoritmo genético y nos dice lo siguiente, "Un algoritmo tiene un mejor desempeño cuando esquemas cortos de bajo orden con buen fitness se recombinan para formar esquemas de mayor orden con aún un mejor desempeño" [Forrest and Mitchell, 1993].

Además de esta hipótesis existen explicaciones alternativas al funcionamiento de los algoritmos genéticos como lo son:

Hipótesis de macro-mutación: Esta hipótesis menciona que el proceso de recombinación puede verse como una macro-mutación (mutación de más de 1 o 2 bits). El operador del mismo nombre (ver Fig 3.12) tomará una secuencia de posiciones y la reemplazará con otra aleatoria. Este operador es muy parecido a la recombinación de uno o dos puntos. Además tiene la ventaja de, en algunos casos, no necesitar de una población.

X	1	1	1	0	1	0	0	1	1
Y				1	0	1			
Z	1	1	1	1	0	1	0	1	1

Figura 3.12: Ejemplo de operador macro-mutación.

Hipótesis de mutación adaptativa: En esta hipótesis el operador de recombinación sirve como un mecanismo que se adapta a las diferentes etapas de convergencia de la población. Esto significa que cuando la población comienza a converger en una región más pequeña del espacio de búsqueda, la recombinación producirá nuevos individuos dentro de la misma. Por lo que en esta hipótesis podemos considerar a la recombinación como un método de mutación adaptativa que reduce la fuerza de la mutación a lo largo del tiempo. En este caso hace uso de una población pero no a través de bloques de construcción.

3.4. El problema de One Max

Uno de los problemas más sencillos que se pueden utilizar para la implementación de algoritmos genéticos es el problema One Max, el cual consiste en por encontrar el máximo valor de una cadena de caracteres de valores binarios [Saad et al., 2007]. Para generar un número de cadenas uno puede hacerlo manualmente, simulando el lanzamiento de una moneda la cantidad de veces que sea necesario. Los valores de cada una de los caras de la moneda serán 1 y 0. Para calcular el valor máximo de una cadena uno puede utilizar la siguiente función [Schaffer and Eshelman, 1991].

$$F(X) = \sum_{i=1}^{N} x_i. \tag{3.1}$$

Donde N es el número de en nuestra población y X_i es el arreglo binario.

Puede ser bastante obvio para nosotros que el mejor individuo es aquel cuyas entradas tienen un valor de 1. Sin embargo el algoritmo genético no sabe nada de esto y necesita buscar ciegamente por la mejor solución utilizando sus operadores genéticos.

Uno de las ventajas de este problema radica en el hecho que nos permite explorar las técnicas de recombinación y mutación de una manera sencilla, además entender como estos afectan la búsqueda del algoritmo.

Para este ejemplo se buscó el mejor individuo de longitud 30, para esto se utilizaron 50 generaciones y 50 individuos. En este caso se utilizó una de recombinación de un punto con probabilidad de 90 %, mutación Flipping con 2 % de probabilidad y una selección de torneo con tres individuos. El código utilizado se puede encontrar en Apéndice B. A continuación se presentan los mejores individuos en las generaciones 1, 10, 20, 30, 40 y 50.

Generación	Mejor Individuo	Fitness
1	0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 0 1 1 1 1	23
10	1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1	27
20	1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	29
30	1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	29
40	1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	29
50	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	30

Tabla 3.1: Mejores individuos en las generaciones 1, 10, 20, 30, 40 y 50.

3.5. Algoritmo genético en Python

En las secciones anteriores se mencionaron los principios teóricos de un algoritmo genético, en esta sección se implementará un AG para encontrar los puntos máximos de funciones de una sola variable.

Los algoritmos genéticos comparten la estructura que se presenta en Figura 3.13.

- Crear población inicial: Se crea la primera población.
- Selección: Se seleccionarán los mejores padres.
- Recombinación: Se combinarán los mejores padres para obtener mejores soluciones.
- Mutación: Se mutan algunos individuos de la población para agregar diversidad.
- Se calcula el fitness de toda la población
- Criterio de convergencia: Si se cumple el criterio de convergencia se detiene el algoritmo y se elige al individuo más apto, de no ser así se regresa a la selección.

Actualmente existen diferentes librerías con las cuales uno puede incorporar algoritmos genéticos, como, Distributed Evolutionary Algorithms (DEAP) [Fortin and De Rainville,], GENEAI [Matos Chaves,], Karoo GP [Staats,], Tiny Genetic Programming [Sipper,], Symbiotic Bid-Based GP [de Castro Bonson,]. Dichas librerías facilitan la implementación de algoritmos genéticos sin embargo aquí nos enfocaremos principalmente en el funcionamiento de estos, construiremos nuestro propio algoritmo y posteriormente haremos uso de

Algoritmo 7 Algoritmo genético simple (jav: no se menciona en el texto.)

```
Padres ← población generada aleatoriamente
P ← población inicial
  while not (condición do
    for p in P do
      Hijos ← ∅
      Evaluar(p)
      Seleccionar(p)
      Recombinar padres para crear hijos c1 y c2
      Hijos ← Hijos ∪{c1, c2}
      Mutar(p)
    end for
  end while
x ← mejor individuo
Return x
```

DEAP. Para el caso de funciones de una variable se utilizó algoritmo genético en Apéndice C.

3.5.1. Resultados

Se comparan los resultados del AG con la función optimize de la librería Scipy, la cual puede utilizar diferentes algoritmos [opt,]. Para esta comparación solo se tomaron en cuenta los algoritmos Broyden Fletcher Goldfarb Shanno(L-BFGS-B) [Morales and Nocedal, 2011, Byrd et al., 2003], Sequential quadratic programming (SLSQP) [Kraft, 1988] y método de Powell [Powell, 1964, Press et al., 2007]. Los primeros dos algoritmos tienen la característica de que utilizan derivadas y asumen que las funciones son diferenciables. Esto, por una parte, presenta un beneficio en costo computacional siempre y cuando las derivadas puedan ser calculadas ya que este cálculo solo se realiza una sola vez por iteración a diferencia de un AG que tiene que realizar el mismo cálculo múltiples veces [Gray and Fowler, 2011].

En este caso para comparar el algoritmo genético con la librería de Optimize se tomarán las siguientes funciones (Fig 3.14) [Gjylapi and Kasëmi, 2014]. De arriba hacia abajo, nos referiremos a dichas funciones como función $f_1(x)$, $f_2(x)$, $f_3(x)$, $f_4(x)$ y $f_5(x)$.

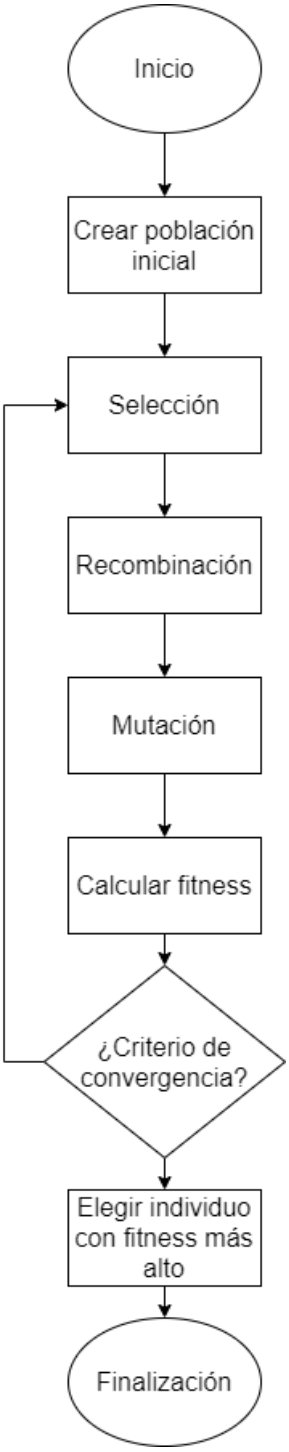


Figura 3.13: Diagrama de flujo de algoritmo genético.

Los parámetros utilizados para las funciones consistieron de 100 individuos. El método de mutación consiste en mutar todos los padres dentro de todo el espacio de búsqueda alrededor del mejor padre, posteriormente, con el paso de las generaciones, la mutación se dará en espacios cada vez más pequeños alrededor del mejor individuo. Dada la naturaleza del problema no es necesario hacer uso de recombinación. Igualmente la selección consiste en un método de torneo entre todos los individuos. El criterio de convergencia consistió en un número de generaciones de 100.

$$f_1(x) = (x^2 + x)\cos(2x) + x^2, \quad -20 < x < 20. \tag{3.2}$$

$$f_2(x) = \sin^2(3x + 45) + 0.9\sin^3(9x) - \sin(15x + 50)\cos(2x - 30). \tag{3.3}$$

$$f_3(x) = \frac{-x^6}{60} - \frac{x^5}{50} + \frac{x^4}{2} + \frac{2x^3}{3} - 3.2x^2 - 6.4x. \tag{3.4}$$

$$f_4(x) = \begin{cases} x - 2, & \text{if } x \geq 1 \\ -x^2, & \text{otro caso} \end{cases} \tag{3.5}$$

$$f_5(x) = -\frac{x}{(x-1)^2}. \tag{3.6}$$

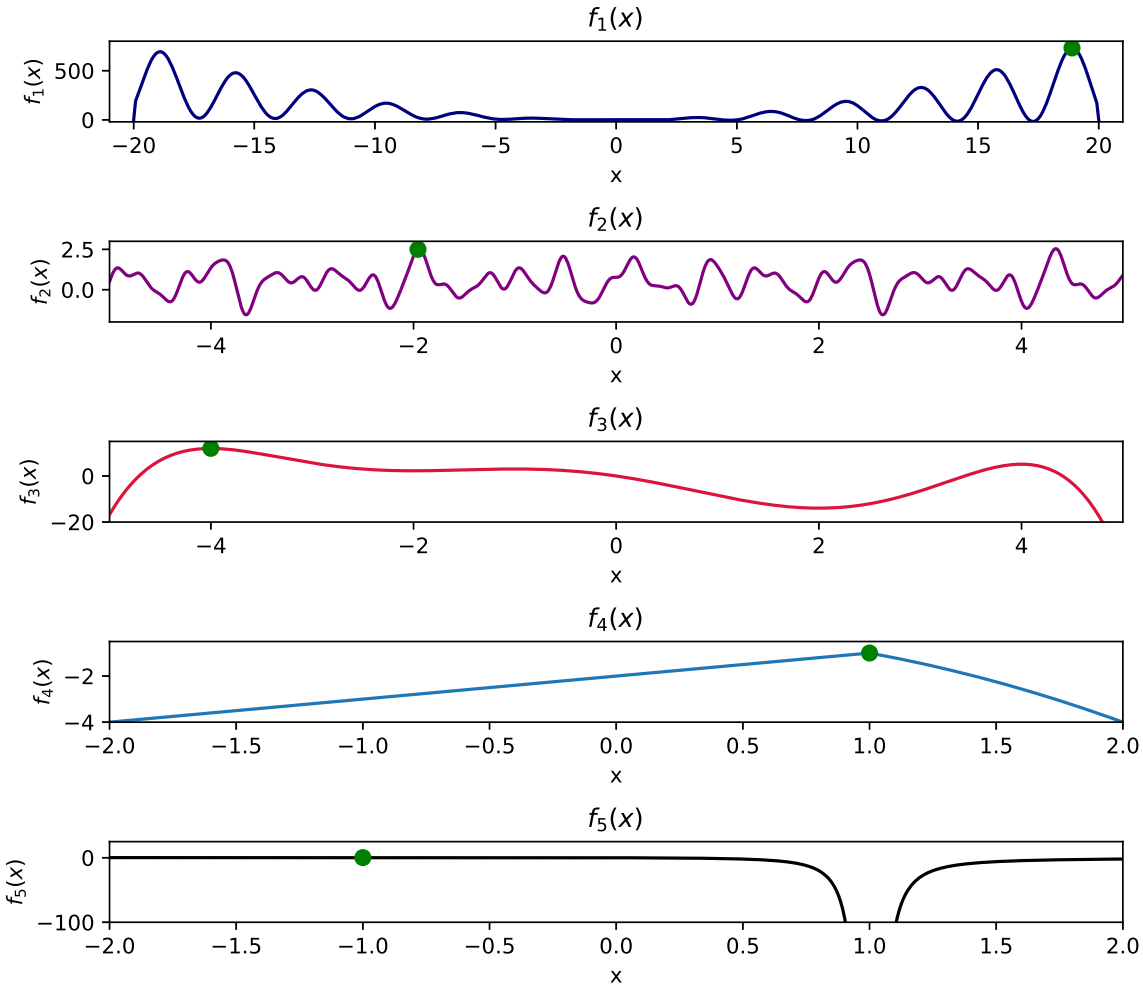


Figura 3.14: De arriba hacia abajo, gráficas de ecuaciones (3.2), (3.3), (3.4), (3.5) y (3.6) con sus respectivos máximos globales en verde: 18.90, -1.95, -4.00, -1.00 y 0.25. (jav: es la posicion o valor maximo?) (jav: revisar f4 y f5.)

Realizando la comparación entre al AG y los métodos de optimización para las ecuaciones 3.2 y 3.3, se puede observar que el algoritmo genético fue el único capaz de encontrar el óptimo global (ver Tabla 3.2). Para la función 3.4 se puede observar que el algoritmo de Powell también encontró el óptimo. Por último para el resto de las funciones (3.5 y 3.6) todos los algoritmos encontraron el óptimo. (jav: mecionar los rangos o puntos iniciales que necesitaste en BFGS, SLSQP, POWELL, ya que encontrar la solucion depende de estos valores.) Dado que en este ejercicio estamos analizando funciones de una variable, que se pueden representar visualmente en dos dimensiones, es posible visualizar la función

Función		AG	BFGS	SLSQP	POWELL
$f_1(x)$	Mejor padre	18.896	0	0	9.538
	Mejor fitness	731.390	0	0	170.330
$f_2(x)$	Mejor padre	-1.940	-0.018	-0.018	1.244
	Mejor fitness	2.503	0.768	0.768	1.055
$f_3(x)$	Mejor padre	-4.000	-2.000	-2.000	-4.000
	Mejor fitness	11.946	2.240	2.240	11.947
$f_4(x)$	Mejor padre	1.000	1.000	1.000	1.000
	Mejor fitness	-1.010	-1.000	-1.001	-1.000
$f_5(x)$	Mejor padre	-1.000	-1000.	-1.000	-1.00
	Mejor fitness	0.250	0.250	0.250	0.250

Tabla 3.2: Tiempo promedio, mejor padre y mejor fitness para funciones de prueba. Los óptimos globales se encuentran en 18.9, 1.94, -4.0, 1.0, -1.0 para $f_1(x)$, $f_2(x)$, $f_3(x)$, $f_4(x)$ y $f_5(x)$ respectivamente.

para posteriormente utilizar técnicas de bracketing [Kochenderfer and Wheeler, 2019] para encerrar el óptimo global y asegurar una efectiva localización de este.

Uno se puede preguntar ¿Porque utilizar un AG sobre otro método de optimización? Esto dependería de tu problema, si nuestra función no es continua no podríamos implementar de manera tan sencilla los métodos mencionados en el capítulo anterior.

3.5.2. Enfoque de algoritmo genético

Como se mencionó en la sección anterior el AG es una de las muchas opciones para el encontrar el óptimo de funciones de N variables. Sin embargo dicho desempeño dependerá de los parámetros como el número de individuos en la población y el número de iteraciones. Sabemos que un número pequeño de individuos e iteraciones se traducirán en menor tiempo de ejecución pero en una menor exploración del espacio de búsqueda.

Con un número bajo de individuos el AG explora el mismo espacio de búsqueda pero con menor precisión. Se realizaron distintas pruebas para encontrar la mejor combinación entre cantidad de individuos y generaciones, para la función 3.4 basta con 50 individuos en la población y 50 generaciones, con un tiempo promedio de 28.7ms $\pm$ 2.2ms iteraciones para encontrar el óptimo global. Para mostrar un poco más a fondo como las generaciones y el número de individuos puede afectar el tiempo de ejecución de un algoritmo genético, se

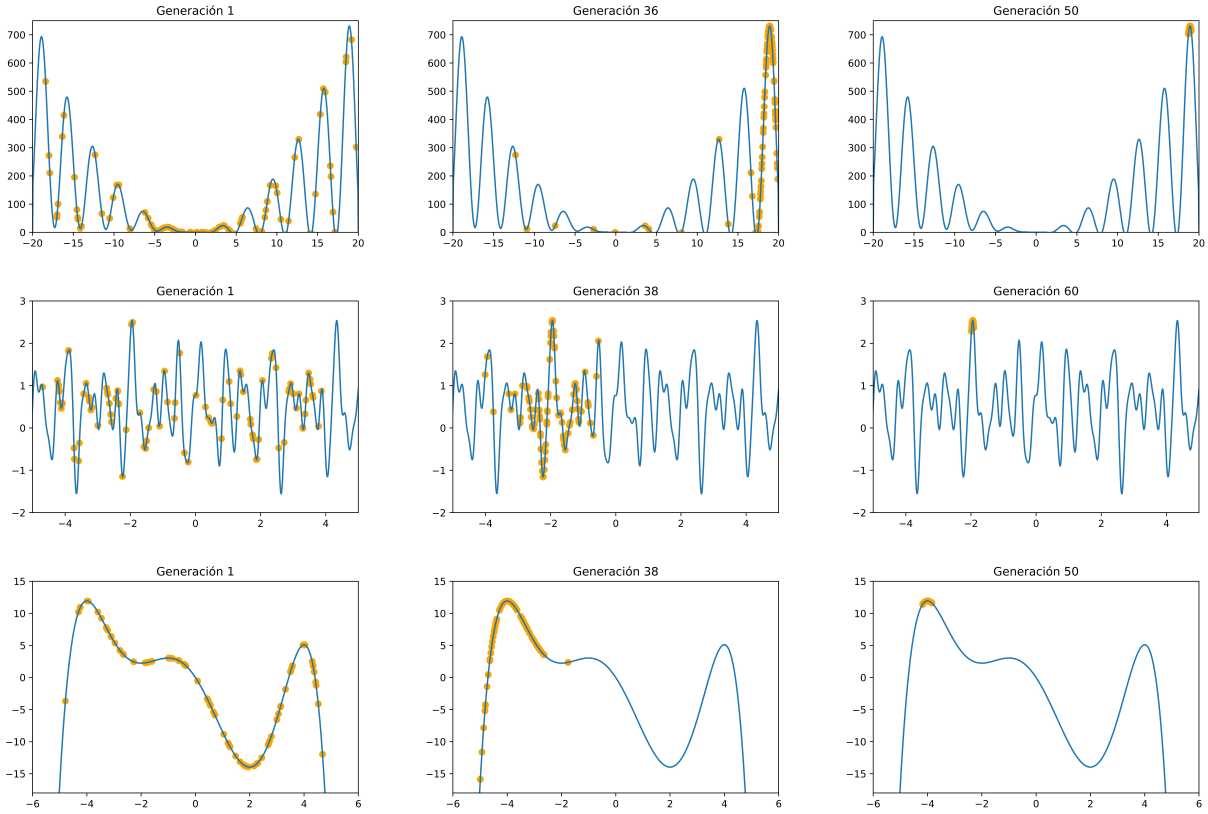


Figura 3.15: Exploración del espacio de búsqueda para diferentes generaciones. (jav: no se menciona ni explica en el texto.)

realizo una prueba con 100 individuos y 100 generaciones, esta combinación de parámetros tiene un tiempo de ejecución promedio de 64.6ms ± 1.56ms. Para encontrar los tiempos aquí mencionados se realiza la ejecución del código cien veces para calcular el promedio y la desviación estándar.

Tenemos que recordar que estas funciones de prueba son bidimensionales (jav: ??), por lo que se pueden representar en un espacio tridimensional (jav: ??), por dicha razón es importante considerar diferentes números de individuos y generaciones, métodos de mutación y recombinación.

Dada la naturaleza estocástica de los AG nos encontraremos que uno de los mayores problemas para encontrar la mejor solución son los parámetros que vuelven al algoritmo más eficaz. En este caso se presenta un algoritmo genético lo suficientemente sencillo para aplicarlo a funciones de una variable. Para validar el desempeño de nuestro AG comparamos los resultados y tiempos de ejecución contra DEAP (una de las librerías más populares de algoritmos genéticos para Python) en este al igual que en el caso anterior se tomo el mismo número de individuos, generaciones y límites que en el caso anterior. Los resultados del algoritmo genético en DEAP son los siguientes, 18.9004, -1.9449, -4.0001, -1.0000 y -1.0000, para las función 3.2, 3.3, 3.4, 3.5 y 3.6 respectivamente.

Para atacar este problema de "funcionalidad" para funciones con más variables uno podría proponer utilizar un número grande de individuos e iteraciones. Aunque esta solución parezca ser la más fácil, no es la mejor dado que se aumentaría el tiempo de ejecución. La solución en este caso consistiría de utilizar métodos de selección, mutación y recombinación adecuados al problema. Hoy en día no existe un método para encontrar la mejor combinación de parámetros para un AG, la única manera de hacerlo es analizar la naturaleza del problema para después con prueba y error encontrar aquellos con el mejor rendimiento.

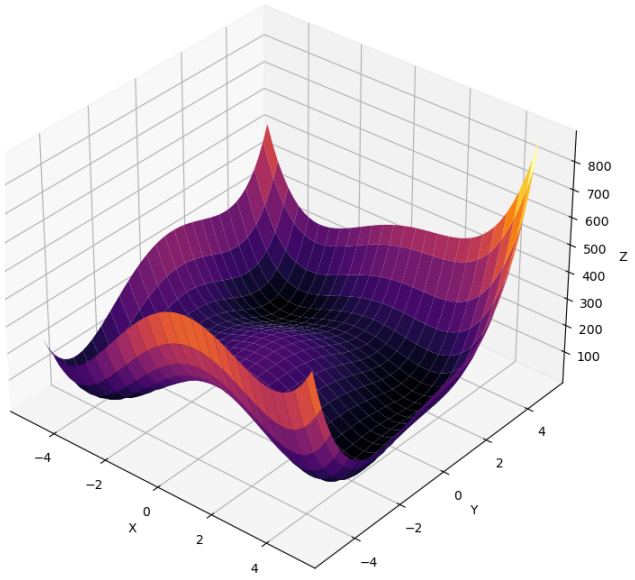


Figura 3.16: Función de Himmelblau

3.6. Aplicación a N dimensiones

En la sección anterior presentamos un algoritmo genético para la optimización de funciones de una variable, pero no se pueden modelar fenómenos naturales en una sola dimensión. Es por ello que se presenta un AG para la optimización de funciones de N dimensiones.

Uno puede crear el algoritmo desde cero pero su programación puede complicarse en función del problema, por lo que para la programación de este algoritmo utilizaremos la librería DEAP.

Nuestro objetivo consiste en encontrar el óptimo de la función de Himmelblau:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2. \tag{3.7}$$

Aunque esta función parece más simple en comparación a otras funciones tridimensionales, llama la atención porque es multimodal, es decir, tiene más de un mínimo global (puntos rojos en Fig 3.17). Para ser exactos, la función tiene cuatro mínimos globales que se evalúan en 0, que se pueden encontrar en los siguientes lugares [Wirsansky, 2020]:

1. $x = 3.000$, $y = 2.000$.

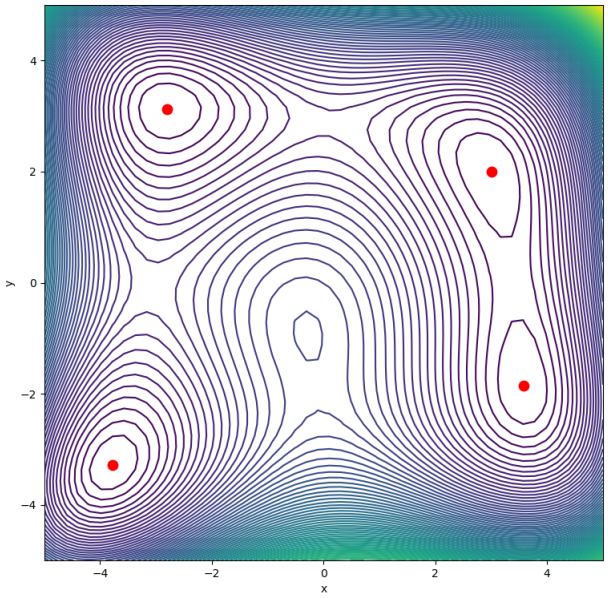


Figura 3.17: Gráfica de superficie de función de Himmelblau.

2. $x = -2.805$, $y = 3.131$.

3. $x = -3.779$, $y = -3.283$.

4. $x = 3.584$, $y = -1.848$.

Si queremos encontrar todos los óptimos globales en una sola ejecución, podemos utilizar la técnica de "niching" y "sharing", una técnica de AG que imita la forma en que un entorno natural se divide en múltiples subentornos, o nichos. Estos nichos están poblados por diferentes especies o subpoblaciones, que aprovechan los recursos únicos disponibles en cada nicho, mientras que los especímenes que coexisten en el mismo nicho tienen que competir por los mismos recursos. Implementar un mecanismo de repartición dentro del algoritmo genético animará a los individuos a explorar nuevos nichos y puede utilizarse para encontrar varias soluciones óptimas, cada una de las cuales se considera un nicho. Una forma común de lograr la compartición es dividir el valor del fitness de cada individuo con alguna función, la cual haga uso de las distancias combinadas de todos los demás individuos, penalizando de forma efectiva a una población saturada al compartir la riqueza local entre sus individuos.

Cuando queremos implementar esta técnica necesitamos hacer uso de más individuos y más generaciones esto se debe a que queremos explorar toda la búsqueda y explotar las mejores zonas con el fin de poblarlas lo más posible para buscar otro óptimo. Hemos ejecutado nuestro algoritmo con 200 individuos y 200 generaciones y se encontraron los siguientes resultados:

- x= 3.010, y= 1.998
- x= -2.802, y = 3.133
- x = -3.774, y = -3.292
- x = 3.585, y =-1.847

Como podemos ver estos resultados son muy similares a los esperados. Aquí encontramos los óptimos globales de una función de 2 variables. Esto se puede generalizar a N dimensiones y M óptimos globales. Sin embargo, cuantas más dimensiones, más recursos y tiempo van a necesitar los algoritmos.

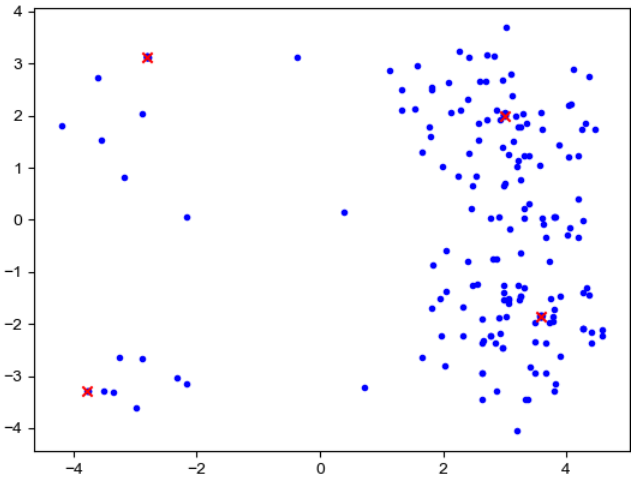


Figura 3.18: Niching y sharing para función de Himmelblau.

Capítulo 4

Resultados

4.0.1. Solución con algoritmo genético

Nuestro primer objetivo es encontrar aquellos valores de $\Omega_{CDM,0}$ y $\Omega_{\Lambda,0}$ que mejor se ajustan a los datos observados (jav: en que ecuacion?). Para esto se tomaron en cuenta las siguientes consideraciones:

- $H_0 = 70$.
- $\Omega_{\Lambda,0} = 1 - \Omega_{CDM,0}$.

Para encontrar aquellos valores de Ω que mejor se ajustan a los datos observacionales, tendremos como función objetivo una prueba de chi-cuadrada, ya que dicha prueba nos puede dar la información suficiente de que tan bueno es un modelo para el conjunto de datos dado. Básicamente esta prueba nos dice si una distribución de frecuencias observada difiere de una distribución teórica .

$$\chi^2 = \sum \frac{[H_{th}(z) - H_{dat}(z)]^2}{\sigma^2}. \tag{4.1}$$

Donde representa $H_{th}(z)$ los datos teóricos (obtenidos a partir de ecu (**??**)), $H_{obs}(z)$ los datos observacionales (jav: cronometros cosmicos?) (jav: esta funcion se puede generalizar a cualquier conjunto de datos?) y σ el error asociado a los datos.

La estructura es muy similar al ejemplo anterior con la única diferencia que se utiliza la ecu **??** para definir nuestra función objetivo 4.1.

Con este algoritmo se asume que $H_0 = 70$ por lo que nuestro se problema se convierte en un problema de una sola variable. Encontramos un valor de $\Omega_{CDM,0} = 0.3079$, $\Omega_{\Lambda,0} = 0.6921$. y $\chi^2 = 7.5494$

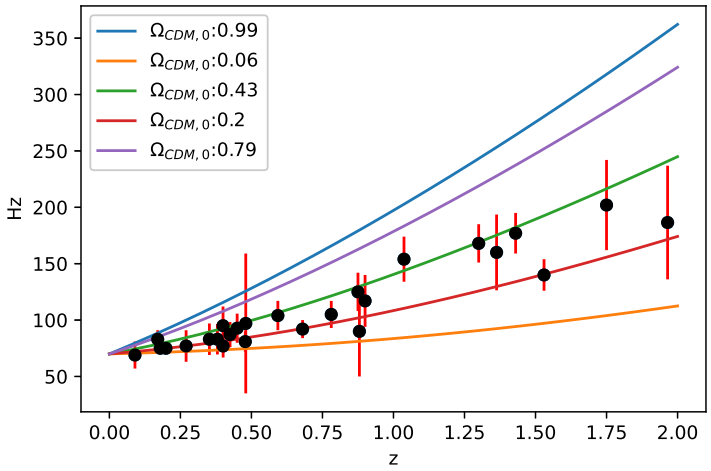


Figura 4.1: Ajuste de modelo **??** para diferentes valores de $\Omega_{\Lambda,0}$, $\Omega_{CDM,0}$ y un valor fijo de $H_0 = 70$ para conjunto de datos $H(z)$. (jav: no se meciona en el texto y explicar que se graficaron algunas selecciones de valores 'aleatorias'.)

Posteriormente se dejo de tomar un valor para H_0 por lo que nuestro problema de optimización se vuelve de dos variables, pero como se mencionó anteriormente los algoritmos genéticos no están limitados a una dimensión. Para este problema en particular se utilizó un código más completo el cual cuenta con diferentes métodos de selección y mutación [Amat Rodrigo, 2019]. En particular en este ejemplo se utilizo una selección de torneo y una mutación uniforme.

Con este algoritmo encontramos un valor de $H_0 = 68.3653$, $\Omega_{CDM,0} = 0.3197$, $\Omega_{\Lambda,0} = 0.6803$ y $\chi^2 = 7.2062$.

Los valores reportados en la literatura nos dicen que el valor de H_0 se encuentra en un rango de $[64.9-72.0]$ $\text{kms}^{-1}\text{Mpc}^{-1}$[Bernal et al., 2016, Busti et al., 2014b, Busti et al., 2014a, Luković et al., 2016] y $\Omega_{CDM,0}$ entre $[0.28-0.35]$[Luković et al., 2016, Val, 2002, Conley et al., 2006].

Tomando en cuenta nuestros resultados y aquellos reportados en la literatura podemos darnos cuenta que los valores encontrados por el AG se encuentran dentro de los rangos aceptados.

Se utilizó el mismo código del ejemplo anterior en combinación con los datos $H(z)$ y ecuación **??** (jav: poner nombres de ecns/modelos), se obtuvieron los siguientes resultados: $H_0 = 68.7361$, $\Omega_{m,0} = 0.3103$, $\Omega_{1,0} = 0.1024$, $\Omega_{2,0} = 0.0337$ y $\chi^2 = 8.6686$. (jav: esto no es correcto)

Se realizó el mismo ejercicio para ecuación **??** (jav: poner nombres de ecns/modelos). En este caso se utilizo *SimpleMC* un código el cual cuenta con diferentes métodos de estimación de parámetros y datos.

Se ejecutó el algoritmo 50 veces, cada una de ellas con 300 individuos en la población y 150 generaciones además de una combinación de datos BAO y UnionSN [uni,]. Se obtuvieron los siguientes resultados $H_0 = 64.4078$, $\Omega_{m,0} = 0.2777$, $w_0 = -0.8826$ y $w_a = 0.1253$ con un fitness de $\chi = 15.7503$, estos valores se encuentran dentro de los valores aceptados [Ferramacho et al., 2010][Pan et al., 2020] si se toma en cuenta que se cada uno de los parámetros como valores desconocidos.

(jav: Escribir que chi-cuadrada es para cada conjunto de datons, H, SN, BAO. Y que la chisq es la sumas de sus indiviuales. Poner porq?)

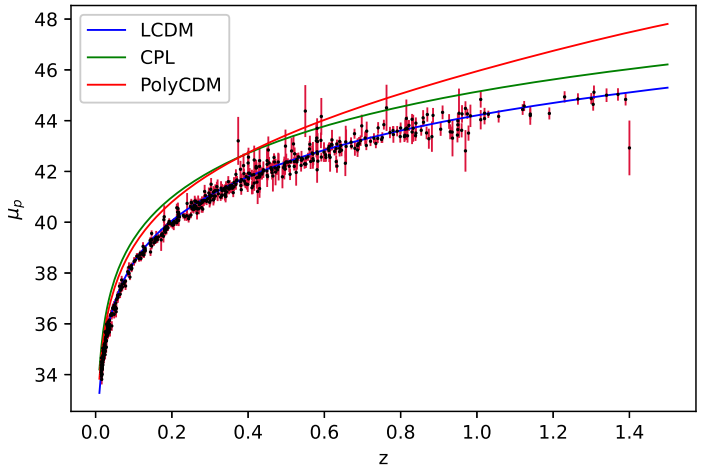


Figura 4.2: Datos de UnionSN. Este conjunto de datos representa la el módulo de distancia y el corrimiento al rojo para 580 supernovas. (jav: deben conicidir las lineas con los datos.)

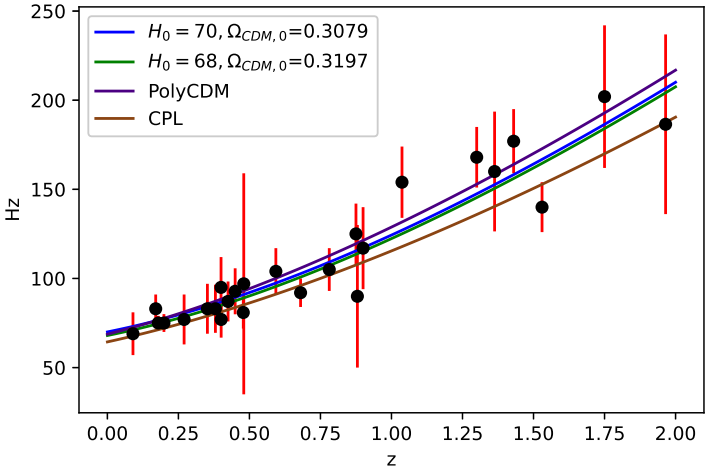


Figura 4.3: Se presentan los tres diferentes ajustes para cada modelo. Línea azul representa ΛCDM con valor fijo de $H_0 = 70$, línea verde representa mismo modelo pero sin ningún valor fijo, por último línea morada representa el ajuste al modelo PolyCDM.

(jav: Explicar resultados tabla 1. Estan deacuerdo los datos? son correctos? hay diferencias? hay algo interesante de notar?)

(jav: Explicar tabla 2, que resultados puedo obtener de ahi, coinciden con las figuras? que puedo concluir?)

(jav: Escribir que aqui nos limitamos a estos modelos, como primera prueba, pero se puede

	Parameters	LCDM	wCDM	PolyCDM	CPL
HD		0.3192	0.3103	0.3349	0.2782
HD+BBAO	Ω_m	0.2995	0.3020	0.3398	0.3603
HD+BBAO+SN		0.2991	0.2893	0.3255	0.3004
HD		68.1785	73.1839	69.7875	71.8957
HD+BBAO	H_0	67.4737	63.3047	67.2558	59.6734
HD+BBAO+SN		67.4931	65.1389	69.4497	64.3478
HD		-	-1.4148	-	-
HD+BBAO	w	-	-0.8145	-	-
HD+BBAO+SN		-	-0.9103	-	-
HD		-	-	-0.4116	-
HD+BBAO	$\Omega_{1,0}$	-	-	1.0512	-
HD+BBAO+SN		-	-	0.6752	-
HD		-	-	0.0426	-
HD+BBAO	$\Omega_{2,0}$	-	-	-0.3559	-
HD+BBAO+SN		-	-	-0.2813	-
HD		-	-	-	-1.3884
HD+BBAO	w_0	-	-	-	-0.2670
HD+BBAO+SN		-	-	-	-0.7837
HD		-	-	-	1.1430
HD+BBAO	w_a	-	-	-	-1.6533
HD+BBAO+SN		-	-		-0.4081

Tabla 4.1: Valores de parámetros encontrados por SimpleMC según el modelo y conjunto de datos. (jav: Hacer figuras de datos y modelos, usando los mejores ajustes de los 4 modelos para la combinacion HD+BBAO+SN.)

Dataset	LCDM	wCDM	PolyCDM	CPL
HD	7.2500	7.2181	7.2081	7.1338
HD+BBAO	14.0176	13.3099	12.1203	12.8534
HD+BBAO+SN	30.6286	30.4691	29.6429	29.1315

Tabla 4.2: Valores fitness para diferentes modelos y conjuntos de datos. (jav: Leer este art. https://arxiv.org/abs/astro-ph/0608184) (jav: Calcular AIC, BIC y decir cual es el mejor modelo que describe cada combinacion de datos.)

extender el analisis a cualquier modelo mas complejo o mas conjuntos de datos.)

4.0.2. Algoritmos genéticos como método de validación

Como se mencionó anteriormente uno de los puntos más fuertes de los AG es su enfoque de caja negra, lo que significa que no se necesita información acerca del problema para encontrar una solución. Es por esta misma razón que los AG pueden funcionar como

herramienta de validación. Por ejemplo, imaginemos que utilizamos un método de regresión simbólica como en [Arjona and Nesseris, 2020]. En [Arjona and Nesseris, 2020] se utiliza un AG para reconstruir una función analítica que mejor describe un conjunto de datos. Ecuación 4.2 tiene una serie de coeficientes con valores de $a = 0.652$, $b = 0.228$ y $c = 0.017$. (jav: Esto se puede ver como una variacion del metodo polyCDM, mencionar.)

$$H(z) = H_0(1 + z(a + bz - cz^3)^2). \tag{4.2}$$

En este caso utilizaremos el AG para confirmar si efectivamente estos coeficientes, son aquellos, que minimizan una prueba χ^2. Para esto comparemos exclusivamente los resultados de χ^2 presentados en [Arjona and Nesseris, 2020] con los mismos datos (jav: cuales datos, solo un conjunto? dos? tres?).

Ejecutando el AG se obtuvo un promedio de 68.42 para H_0 y $a = 0.615$, $b = 0.3090$ y $c = 0.0288$ para los coeficientes de la función, con un fitness promedio de 12.499. Los resultados son similares a los encontrados en [Arjona and Nesseris, 2020] sin embargo estos tienen un mejor fitness que aquellos presentados en dicho artículo (jav: esto es por la diferencia de datos que ellos usan.). Se cree que la razón de esto es que los autores de [Arjona and Nesseris, 2020] no solamente encontraron los coeficientes si no también las variables, términos independientes y grados de la función, y no conocemos los pesos para encontrar los parámetros a priorizar. Por lo que podemos aplicar el AG en dos pasos, el primero para encontrar la función y el segundo para encontrar aquellos coeficientes que mejor se ajustan al conjunto de datos.

	Parámetros	LCDM	PolyCDM	CPL
HD		0.3192	0.3349	0.2782
HD+BBAO	Ω_m	0.2995	0.3398	0.3603
HD+BBAO+SN		0.2991	0.3255	0.3004
HD		68.1785	69.7875	71.8957
HD+BBAO	H_0	67.4737	67.2558	59.6734
HD+BBAO+SN		67.4931	69.4497	64.3478
HD		-	-0.4116	-
HD+BBAO	$\Omega_{1,0}$	-	1.0512	-
HD+BBAO+SN		-	0.6752	-
HD		-	0.0426	-
HD+BBAO	$\Omega_{2,0}$	-	-0.3559	-
HD+BBAO+SN		-	-0.2813	-
HD		-	-	-1.3884
HD+BBAO	w_0	-	-	-0.2670
HD+BBAO+SN		-	-	-0.7837
HD		-	-	1.1430
HD+BBAO	w_a	-	-	-1.6533
HD+BBAO+SN		-		-0.4081

Tabla 4.3: Valores de parámetros encontrados por SimpleMC según el modelo y conjunto de datos. (jav: son iguales a las anteriores? se repiten?)

Datos	LCDM	PolyCDM	CPL
HD	7.2500	7.2081	7.1338
HD+BBAO	14.0176	12.1203	12.8534
HD+BBAO+SN	30.6286	29.6429	29.1315

Tabla 4.4: Valores fitness para diferentes modelos y conjuntos de datos. (jav: son iguales a las anteriores? se repiten?)

Capítulo 5

Conclusiones

Al inicio de este trabajo se presentan una variedad de técnicas de optimización matemática, estos algoritmos son utilizados en diferentes campos y destacan resolviendo problemas de los cuales tenemos poca información, no tenemos información sobre las derivadas o tenemos una interpretación compleja del problema. Posteriormente se propone el uso de algoritmos genéticos para abordar dichos problemas.

Los algoritmos genéticos son una gran herramienta de búsqueda para funciones de N dimensiones, especialmente dimensiones grandes. Una desventaja de dicha técnica son sus tiempos de ejecución altos, pero tienen un mejor desempeño que otros algoritmos de optimización al momento de realizar búsquedas en espacios con muchos óptimos locales.

Se comparo un algoritmo genético con los métodos de optimización BFGS, SLSQP y Powell. Se mostró que, efectivamente, el tiempo de ejecución del AG puede llegar a ser hasta 30 veces más alto que los métodos mencionados anteriormente. Por un lado esto es un desventaja pero por otro lado el AG fue capaz de localizar los óptimos globales de diferentes funciones de prueba con 100 % de eficacia.

Se presentaron dos ejemplos del uso de algoritmos genéticos en el campo de la cosmología, la primera para el ajuste de modelos cosmológicos, ΛCDM, PolyCDM, Chevallier, Polarski y Linder (CPL), para encontrar los parámetros de densidad y materia oscura. La

segunda aplicación corresponde a la regresión simbólica y confirmación de resultados con algoritmos genéticos. Esta aplicación puede explorarse más a fondo y aplicarse a diferentes algoritmos de aprendizaje de máquina, para la afinación de hiperparámetros.

Apéndice A

Teorema del esquema

Como se mencionó anteriormente un esquema es una plantilla la cual describe un subconjunto de cadenas con similitudes respecto a la posición. Esta formada por una terna $\{0, 1, *\}$, donde $*$ describe todas las posibles similitudes entre cadenas de cierta longitud [Sivanandam and Deepa, 2007].

Un esquema representa una variedad afín del espacio de búsqueda por ejemplo: el esquema $01{*}{*}11{*}0$ es un subespacio del espacio de aquellos individuos con una longitud de 8 bits.

Un algoritmo genético modelado con el teorema del esquema cuenta con los siguientes operadores:

- El operador de selección. Una selección proporcionada donde el fitness sea una variable para generar a la siguiente generación, esto es: entre mayor sea el fitness de un individuo mayor será su probabilidad de ser seleccionado.
- Los operadores genéticos, recombinación de un solo punto y mutación flipping aplicados aleatoriamente con una probabilidad de P_r y P_m.

Una definición más formal de dicho teorema, también conocido como el teorema fundamental de los algoritmos genéticos, viene dado por:

Para un esquema H sea:

- $m(H,t)$ la frecuencia relativa del esquema H en la población de la generación t^{ma}.
- $f(H)$ el fitness promedio de los elementos de H.
- $O(H)$ el número de bits fijos en el esquema H, llamado orden del esquema.
- $\delta(H)$ la distancia entre el primer y último bit fijo del esquema H, llamado la longitud definida del esquema.
- $\bar{f}$ es el fitness promedio de toda la población.
- $l(H)$ longitud del esquema.
- P_r la probabilidad de recombinación.
- P_m la probabilidad de mutación.

Entonces:

$$E[m(H,t+1)] \geq m(H,t)\frac{f(H)}{f}\left[1 - P_r\frac{\delta(H)}{l(H)-1} - O(H)P_m\right] \tag{A.1}$$

Donde E es la esperanza del esquema. Observando el término entre corchetes, también conocido como probabilidad de disrupción, podemos darnos cuenta que, por una parte, para valores pequeños de $\delta(H)$, (H) y $O(H)$ tendremos una disrupción pequeña. Análogamente para valores pequeños de P_r y P_m. En el caso contrario, para valores muy grandes, tendremos que la disrupción será mucho mayor.

Esta expresión cualitativamente nos dice que aquellos esquemas con mejor desempeño, longitud definida corta y de bajo orden aumentarán su frecuencia en las siguientes generaciones.

Para entender un poco más a fondo este teorema tomemos un ejemplo. Consideremos el esquema H:

1 * 1 0 * 1

Figura A.1: Ejemplo de un esquema de longitud 6.

En este caso la longitud $l = 6$, la longitud definida $\delta(H) = 5$ y el orden $O(H) = 4$. Si este esquema H tiene un mejor fitness que los demás, en la población observaremos más individuos con sus posibles combinaciones de H.

Apéndice B

Algoritmo genético One Max

```
#CODIGO EXTRAIDO DE https://github.com/PacktPublishing/Hands-On-Genetic-
    Algorithms-with-Python/blob/master/Chapter03/01-OneMax-long.py

from deap import base, creator, tools
import matplotlib.pyplot as plt
import random

# constantes
ONE_MAX_LENGTH = 30
POPULATION_SIZE = 50 # individuos
P_CROSSOVER = 0.9 # probabilidad de recombinacion
P_MUTATION = 0.1 # probabilidad de mutacion
MAX_GENERATIONS = 50 # numero de generaciones

#Funcion para generar individuos
def zeroOrOne():
    return random.randint(0, 1)


creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
```

```
toolbox.register("zeroOrOne", random.randint, 0,1)
toolbox.register("individualCreator", tools.initRepeat, creator.
    Individual, toolbox.zeroOrOne, ONE_MAX_LENGTH)
toolbox.register("populationCreator", tools.initRepeat, list, toolbox.
    individualCreator)

#Funcion fitness
def oneMaxFitness(individual):
    return sum(individual),

#evaluar fitness
toolbox.register("evaluate", oneMaxFitness)

toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("mate", tools.cxOnePoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=1.0/ONE_MAX_LENGTH)

toolbox.register("individualCreator", tools.initRepeat, creator.
    Individual, toolbox.zeroOrOne, ONE_MAX_LENGTH)
toolbox.register("populationCreator", tools.initRepeat, list, toolbox.
    individualCreator)

def main():
# Crear poblacion inicial
    population = toolbox.populationCreator(n=POPULATION_SIZE)
    generationCounter = 0

    # Calcular fitness de poblacion
    fitnessValues = list(map(toolbox.evaluate, population))

    for individual, fitnessValue in zip(population, fitnessValues):
        individual.fitness.values = fitnessValue

    # extraer fitess
    fitnessValues = [individual.fitness.values[0] for individual in
```

```
population]

 maxFitnessValues = []
 meanFitnessValues = []

 #Loop principal
 while generationCounter < MAX_GENERATIONS:


     # Seleccion
     offspring = toolbox.select(population, len(population))
     # Clonamos generacion
     offspring = list(map(toolbox.clone, offspring))

     # Aplicar recombinacion y mutacion
     for child1, child2 in zip(offspring[::2], offspring[1::2]):
         if random.random() < P_CROSSOVER:
             toolbox.mate(child1, child2)
             del child1.fitness.values
             del child2.fitness.values

     for mutant in offspring:
         if random.random() < P_MUTATION:
             toolbox.mutate(mutant)
             del mutant.fitness.values

     # Calcular fitness
     freshIndividuals = [ind for ind in offspring if not ind.fitness.
valid]
     freshFitnessValues = list(map(toolbox.evaluate, freshIndividuals
))
     for individual, fitnessValue in zip(freshIndividuals,
freshFitnessValues):
         individual.fitness.values = fitnessValue

     # Reemplazo
```

```
        population[:] = offspring

        fitnessValues = [ind.fitness.values[0] for ind in population]

        maxFitness = max(fitnessValues)
        meanFitness = sum(fitnessValues) / len(population)
        maxFitnessValues.append(maxFitness)
        meanFitnessValues.append(meanFitness)
        print("- Generation {}: Max Fitness = {}, Avg Fitness = {}".
    format(generationCounter, maxFitness, meanFitness))

        # Buscar mejor individuo
        best_index = fitnessValues.index(max(fitnessValues))
        print("Best Individual = ", *population[best_index], "\n")

    plt.plot(maxFitnessValues, color='red', label='Max')
    plt.plot(meanFitnessValues, color='green', label='Promedio')
    plt.xlabel('Generacion')
    plt.ylabel('Fitness')
    plt.legend(loc="best")
    plt.title('Fitness maximo y promedio')
    plt.savefig("onemaxfitness.png", format="png", dpi=800)
    plt.show()
```

Apéndice C

Algoritmo genético en Python

```
#Calculamos valores de H para cada valor z para cada parametros
def h_square(z,list_cdm):
    h_dict = {}

    for i in list_cdm:
        h_dict[i] = list(np.sqrt((H0**2)*(i*(1+z)**3+ (1-i) )))

    return h_dict

def chi_square(z, H_dat, cdm, sigma):
    #Calculamos H para cada Omega
    H_th  = h_square(z,cdm)
    lista2 = []

    for i in H_th:
        resultado = 0
        lista = H_th[i]
        for j in range(len(lista)):
            resultado += (lista[j]-H_dat[j])**2 / sigma[j]

        lista2.append(resultado)

    return lista2
```

```
def _obtener_fitness(z, H_dat, padres, sigma):
    _fitness = chi_square(z, H_dat, padres, sigma)
    Pfitness = list(zip(padres,_fitness))
    Pfitness.sort(key = lambda x: x[1], reverse=False)
    mejor_padre, mejor_fitness = Pfitness[0]
    return round(mejor_padre,6), round(mejor_fitness, 6), Pfitness

def mutacion(z, H_dat, padres, sigma, factor_mutacion):

    n = int(len(H_dat))
    padre, fitness, poblacion = _obtener_fitness(z, H_dat, padres, sigma
   )
    hijos = np.random.normal(padre, factor_mutacion, size=17)

    return hijos

def AG_simple(z, H_dat, padres, sigma, max_iter):

    Poblacion = {}
    Historial=[]
    f_mutacion = 0.2

    mejor_padre, mejor_fitness, poblacion = _obtener_fitness(z, H_dat,
   padres, sigma)
    padre = mutacion(z, H_dat, padres, sigma, f_mutacion)

    for i in range(1, max_iter):

        padre = mutacion(z, H_dat, padre, sigma, f_mutacion)
        padre_actual, fitness_actual, poblacion = _obtener_fitness(z,
   H_dat, padre, sigma)

        Poblacion[i] = poblacion

        if fitness_actual < mejor_fitness:
```

```
            mejor_padre = padre_actual
            mejor_fitness = fitness_actual

        Historial.append((i, fitness_actual))

        f_mutacion = f_mutacion/2

    return mejor_padre, mejor_fitness, Historial, Poblacion
```

Bibliografía

[opt,] scipy.optimize.minimize — SciPy v1.6.3 Reference Guide.

[uni,] Union compilation magnitude vs. redshift table.

[Val, 2002] (2002). ω_M — different ways to determine the matter density of the universe. *Space Science Reviews - SPACE SCI REV*, 100:299–309.

[Amat Rodrigo, 2019] Amat Rodrigo, J. (2019). Optimización con algoritmo genético GA Python.

[Andrychowicz et al., 2016] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent.

[Arjona and Nesseris, 2020] Arjona, R. and Nesseris, S. (2020). What can machine learning tell us about the background expansion of the universe? *Physical Review D*, 101(12).

[Bagavathi and Saraniya, 2019] Bagavathi, C. and Saraniya, O. (2019). Chapter 13 - evolutionary mapping techniques for systolic computing system. In Sangaiah, A. K., editor, *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, pages 207–223. Academic Press.

[Beheshti and Shamsuddin, 2013] Beheshti, Z. and Shamsuddin, S. M. (2013). A review of population-based meta-heuristic algorithm. *International Journal of Advances in Soft Computing and Its Applications*, 5:1–35.

[Benyus, 2012] Benyus, J. M. (2012). Biomímesis.

[Bernal et al., 2016] Bernal, J. L., Verde, L., and Riess, A. G. (2016). The trouble withH0. *Journal of Cosmology and Astroparticle Physics*, 2016(10):019–019.

[Bianchi et al., 2008] Bianchi, L., Dorigo, M., Gambardella, L. M., and Gutjahr, W. J. (2008). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8:239–287.

[Blas et al., 2011] Blas, D., Lesgourgues, J., and Tram, T. (2011). The cosmic linear anisotropy solving system (CLASS). part II: Approximation schemes. *Journal of Cosmology and Astroparticle Physics*, 2011(07):034–034.

[Busti et al., 2014a] Busti, V., Clarkson, C., and Seikel, M. (2014a). Evidence for a lower value for h_0 from cosmic chronometers data? *Monthly Notices of the Royal Astronomical Society: Letters*, 441.

[Busti et al., 2014b] Busti, V., Clarkson, C., and Seikel, M. (2014b). The value of h_0 from gaussian processes.

[Byrd et al., 2003] Byrd, R., Lu, P., Nocedal, J., and Zhu, C. (2003). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16.

[Carroll, 2013] Carroll, D. (2013). Genetic recombination. In Maloy, S. and Hughes, K., editors, *Brenner's Encyclopedia of Genetics (Second Edition)*, pages 277–280. Academic Press, San Diego, second edition edition.

[Cillero Fernando, 2010] Cillero Fernando, A. (2010). Herón de alejandría. su aplicación a la materia de tecnologías. https://www.feandalucia.ccoo.es/indcontei.aspx?d=5016&s=5&ind=230.

[Conley et al., 2006] Conley, A., Goldhaber, G., Wang, L., Aldering, G., Amanullah, R., Commins, E. D., Fadeyev, V., Folatelli, G., Garavini, G., Gibbons, R., and et al. (2006). Measurement of ω_m, $\omega_\Lambda$from a blind analysis of type ia supernovae with cmagic: Using color information to verify the acceleration of the universe. *The Astrophysical Journal*, 644(1):1–20.

[Cummings, 2000] Cummings, N. (2000). A brief history of the travelling salesman problem. https://www.theorsociety.com/about-or/or-methods/heuristics/a-brief-history-of-the-travelling-salesman-problem/.

[Darwin, 1877] Darwin, Charles, .-. (1877). Origen de las especies por medio de la seleccion natural ó conservacion de las razas en su lucha por la existencia por charles darwin traduccion directa de la sexta edicion inglesa por enrique godinez.

[de Castro Bonson,] de Castro Bonson, J. P. Symbiotic bid-based gp.

[de S. Victorino and Filho, 2006] de S. Victorino, I. R. and Filho, R. M. (2006). Application of genetic algorithms to the optimization of an industrial reactor. *IFAC Proceedings Volumes*, 39(2):857–862. 6th IFAC Symposium on Advanced Control of Chemical Processes.

[Deep and Adane, 2011] Deep, K. and Adane, H. (2011). New variations of order crossover for travelling salesman problem. *International Journal of Combinatorial Optimization Problems and Informatics*, 2.

[Dobzhansky, 1951] Dobzhansky, T. (1951). *Genetics and the Origin of Species*. Columbia University Press., New York, USA, 3d edition.

[Doran, 2005] Doran, M. (2005). CMBEASY: an object oriented code for the cosmic microwave background. *Journal of Cosmology and Astroparticle Physics*, 2005(10):011–011.

[Dorigo, 1992] Dorigo, M. (1992). Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano*.

[Dorigo and Stützle, 2006] Dorigo, M. and Stützle, T. (2006). *The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances*, volume 37, pages 250–285.

[Drachal and Pawłowski, 2021] Drachal, K. and Pawłowski, M. (2021). A review of the applications of genetic algorithms to forecasting prices of commodities. *Economies*, 9(1).

[Ferguson, 1989] Ferguson, T. S. (1989). Who solved the secretary problem? *Statistical Science*, 4(3):282–289.

[Ferramacho et al., 2010] Ferramacho, L., Blanchard, A., Zolnierowski, Y., and Riazuelo, A. (2010). Constraints on dark energy evolution. *Astronomy and Astrophysics*, 514:A20.

[Forrest and Mitchell, 1993] Forrest, S. and Mitchell, M. (1993). Relative building-block fitness and the building-block hypothesis. In WHITLEY, L. D., editor, *Foundations of Genetic Algorithms*, volume 2 of *Foundations of Genetic Algorithms*, pages 109–126. Elsevier.

[Fortin and De Rainville,] Fortin, F. A. and De Rainville, F. M. Distributed evolutionary algorithms.

[García Ortega,] García Ortega, J. M. El agente secreto de la evolución.

[ghaheri et al., 2015] ghaheri, a., Shoar, S., Naderan, M., and Hoseini, S. s. (2015). The applications of genetic algorithms in medicine. *Oman Medical Journal*, 30:406–416.

[Gjylapi and Kasëmi, 2014] Gjylapi, D. and Kasëmi, V. (2014). The genetic algorithm for finding the maxima of single-variable functions. *Research Inventy: International Journal Of Engineering And Science*, 4:46–54.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition.

[Gorski et al., 2005] Gorski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., and Bartelmann, M. (2005). HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere. 622(2):759–771.

[Gray and Fowler, 2011] Gray, G. and Fowler, K. (2011). The effectiveness of derivative-free hybrid methods for black-box optimisation. *Int. J. of Mathematical Modelling and Numerical Optimisation*, 2:112 – 133.

[Hancock, 1917] Hancock, H. (1917). *Theory of maxima and minima.* Ginn & Company, Boston, USA.

[Holland, 1992] Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA.

[James, 2003] James, S. C. (2003). *Stochastic Search and Optimization: Motivation and Supporting Results*, pages 1–33. John Wiley & Sons, Ltd.

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4.

[Kochenderfer and Wheeler, 2019] Kochenderfer, M. J. and Wheeler, T. A. (2019). *Algorithms for Optimization.* The MIT Press.

[Kraft, 1988] Kraft, D. (1988). *A software package for sequential quadratic programming.* Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR.

[Kutschera, 2017] Kutschera, U. (2017). Evolution. In *Reference Module in Life Sciences.* Elsevier.

[Lee, 2003] Lee, C.-Y. (2003). Entropy-boltzmann selection in the genetic algorithms. *Trans. Sys. Man Cyber. Part B*, 33(1):138–149.

[Lewis and Bridle, 2002] Lewis, A. and Bridle, S. (2002). Cosmological parameters from CMB and other data: A Monte Carlo approach. 66:103511.

[Lewis and Bridle, 2003] Lewis, A. and Bridle, S. (2003). Cosmological parameters from CMB and other data: A monte carlo approach. *Physical Review D*, 66(10).

[Luke, 2013] Luke, S. (2013). *Essentials of Metaheuristics.* Lulu, second edition. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[Luković et al., 2016] Luković, V., D'Agostino, R., and Vittorio, N. (2016). Is there a concordance value for h_0? *Astronomy & Astrophysics*, 595.

[Macía Vázquez, 2020] Macía Vázquez, L. (2020). El problema de la secretaria.

[Matos Chaves,] Matos Chaves, D. Geneai.

[Mirjalili et al., 2014] Mirjalili, S., Mirjalili, S. M., and Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software*, 69:46–61.

[Morales and Nocedal, 2011] Morales, J. and Nocedal, J. (2011). Remark on "algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound constrained optimization". *ACM Trans. Math. Softw.*, 38:7.

[Pan et al., 2020] Pan, S., Yang, W., and Paliathanasis, A. (2020). Imprints of an extended chevallier–polarski–linder parametrization on the large scale of our universe. *The European Physical Journal C*, 80(3).

[Passino, 2002] Passino, K. (2002). Passino, k.m.: Biomimicry of bacterial foraging for distributed optimization and control. ieee control systems magazine 22(3), 52-67. *Control Systems, IEEE*, 22:52 – 67.

[Pinel et al., 2011] Pinel, F., Danoy, G., and Bouvry, P. (2011). Evolutionary algorithm parameter tuning with sensitivity analysis.

[Powell, 1964] Powell, M. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Comput. J.*, 7:155–162.

[Press et al., 2007] Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (2007). *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3 edition.

[Rechenberg and Eigen, 1973] Rechenberg, I. and Eigen, M. (1973). Evolutionsstrategie: Optimierung technischer systeme nachprinzipien der biologischen evolution.

[Ripley, 2001] Ripley, L. (2001). Mutation. In Brenner, S. and Miller, J. H., editors, *Encyclopedia of Genetics*, pages 1268–1275. Academic Press, New York.

[Riquelme Medina and Luna, 2014] Riquelme Medina, I. and Luna, J. (2014). *Revision de los Algoritmos Bioinspirados*. PhD thesis.

[Rosenberg and Rosenberg, 2012] Rosenberg, L. E. and Rosenberg, D. D. (2012). Chapter 2 - introducing the core concepts. In Rosenberg, L. E. and Rosenberg, D. D., editors, *Human Genes and Genomes*, pages 9–18. Academic Press, San Diego.

[Saad et al., 2007] Saad, A., Avineri, E., Dahal, K., Sarfraz, M., and Roy, R. (2007). *Soft computing in industrial applications. Recent and emerging methods and techniques*, volume 39.

[Schaffer and Eshelman, 1991] Schaffer, J. and Eshelman, L. (1991). On crossover as an evolutionary viable strategy.

[Schwefel., 1977] Schwefel., H.-P. (1977). Numerische optimierung von computer-modellen mittels der evolutionsstrategie,.

[Seljak and Zaldarriaga, 1999] Seljak, U. and Zaldarriaga, M. (1999). CMBFAST: A microwave anisotropy code. Astrophysics Source Code Library, record ascl:9909.004.

[Simon, 2008] Simon, D. (2008). Biogeography-based optimization. *IEEE Transactions on Evolutionary Computation*, 12(6):702–713.

[Simpson and Priest, 1993] Simpson, A. R. and Priest, S. D. (1993). The application of genetic algorithms to optimisation problems in geotechnics. *Computers and Geotechnics*, 15(1):1–19.

[Sipper,] Sipper, M. Tiny genetic programming.

[Sivanandam and Deepa, 2007] Sivanandam, S. N. and Deepa, S. N. (2007). *Introduction to Genetic Algorithms*. Springer Publishing Company, Incorporated, 1st edition.

[Staats,] Staats, K. Karoo gp.

[Wirsansky, 2020] Wirsansky, E. (2020). *Hands-On Genetic Algorithms with Python: Applying genetic algorithms to solve real-world deep learning and artificial intelligence problems*. Packt Publishing.

[Xin-She, 2010] Xin-She, Y. (2010). *A Brief History of Optimization*, chapter 1, pages 1–13. John Wiley & Sons, Ltd.

[Yang, 2010] Yang, X.-S. (2010). *Nature-Inspired Metaheuristic Algorithms*.