UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Proyecto final

Algoritmo Evolutivo para optimizar una cartera de inversiones

Jesus Aldair Bernal Orozco

Juan Ignacio Domene Ashida

Lucía Elizabeth García Nieto

Cómputo Evolutivo

Dr. Fernando Ignacio Becerra López

16 de abril de 2024

Índice

Algoritmo Evolutivo para optimizar una cartera de inversiones

1. Introducción

En el mundo de las finanzas, la maximización de los rendimientos y la gestión del riesgo son los mayores retos que enfrentan los inversionistas, sean estos personas o entidades. En ese sentido, una de las tareas financieras más desafiante es la optimización de carteras, donde el propósito principal es minimizar el riesgo y maximizar los rendimientos esperados, de acuerdo con los objetivos financieros y la tolerancia al riesgo del inversor. La diversificación permite reducir el riesgo y se logra distribuyendo el capital en un amplio conjunto de activos evitando concentrar los recursos en un solo instrumento, de esta forma, al invertir en una variedad de bienes con diferente nivel de riesgo y rendimiento, los inversores pueden reducir el peligro de perder dinero si un activo en particular baja de precio.

En el presente proyecto se ha realizado la optimización de una cartera de inversiones empleando todo lo aprendido acerca de Algoritmos Evolutivos. Para construir la cartera de inversiones se consideraron activos que cotizan en le mercado financiero mexicano, activos internacionales que se pueden adquirir a través del mercado mexicano y fondos de inversión del Gobierno de México. Los datos utilizados para poner a prueba el modelo sugerido se obtuvieron a través de la API de Yahoo Finance.

Los objetivos planteados para este proyecto fueron los siguientes:

- Maximizar el rendimiento anual de la cartera de inversiones utilizando un enfoque de optimización basado en algoritmos evolutivos.
- Mantener el riesgo del portafolio debajo de un umbral determinado por indicadores financieros.
- Diversificar la cartera de inversiones para mitigar el riesgo, estableciendo un limite a la proporción de capital que se invierte en cada activo.
- Evaluar y comparar el rendimiento de la cartera optimizada respecto a benchmarks del mercado.

Alcances:

- Recopilación y procesamiento de datos históricos del mercado financiero mexicano.
- Desarrollo e implementación de un algoritmo evolutivo para la optimización de la cartera de inversiones.

- Establecimiento de restricciones y parámetros de optimización, como los límites de inversión por activo y el nivel de riesgo máximo permitido.
- Análisis comparativo con benchmarks del mercado.
- Documentación y presentación de los resultados obtenidos, junto con conclusiones y observaciones sobre la optimización de carteras.

2. Marco teórico

2.1. El mercado financiero y los activos

Un activo es cualquier recurso tangible o intangible que posee valor económico, y que se espera, genere beneficios o ingresos futuros para su propietario. La compra y venta de estos valores financieros, como acciones, bonos y otros instrumentos financieros se realiza en una bolsa de valores. Las bolsas son mercados financieros organizados donde se reúnen compradores y vendedores para negociar estos activos financieros. Los inversores realizan las transacciones a través de casas de bolsa, siguiendo un proceso de negociación regulado.

En el caso de México, es en la Bolsa Mexicana de Valores (BMV) donde se pueden comerciar una variedad de activos financieros. De acuerdo con su sitio oficial, algunos de los principales tipos de activos que se negocian en la BMV incluyen:

- Acciones: representan la propiedad parcial de una empresa. Las empresas mexicanas emiten acciones que se negocian en la BMV, permitiendo a los inversores comprar y vender participaciones en esas empresas.
- Certificados de depósito (CEDES): son títulos emitidos por instituciones financieras que representan un depósito a plazo fijo.
- Fibras (Fideicomisos de Inversión en Bienes Raíces): son instrumentos de inversión que cotizan en bolsa y están respaldados por bienes raíces. Los inversores pueden comprar y vender unidades de Fibras, lo que les permite acceder al mercado inmobiliario.
- Fibras E (Fideicomisos de Inversión en Energía e Infraestructura): son similares a las Fibras, pero están respaldadas por activos relacionados con energía e infraestructura, como proyectos de energía renovable, infraestructura de transporte, etc.
- Certificados de capital de desarrollo (CKDs): son instrumentos de inversión utilizados para financiar proyectos de desarrollo en México. Los CKDs permiten a los inversores participar en proyectos de infraestructura, bienes raíces, energía, entre otros.
- Bonos gubernamentales y corporativos: Tanto el gobierno mexicano como las empresas pueden emitir bonos para financiar sus operaciones. Estos bonos ofrecen a los inversores una forma de obtener ingresos fijos a través del pago de intereses.

La Bolsa Mexicana de Valores ofrece una amplia gama de productos financieros y proporciona oportunidades de inversión en diferentes sectores de la economía mexicana. Adicionalmente, la BMV permite a los inversores comprar y vender acciones de empresas que no cotizan en México, pero sí en otras bolsas de valores del mundo, estas operaciones se realizan a través del Sistema Internacional de Cotizaciones (SIC) [Grupo BMV, sfb].

Otro tipo de instrumentos financieros a los que se tiene acceso están disponibles a través de Cetesdirecto, una plataforma gratuita del Gobierno de México para fomentar y democratizar el ahorro y la inversión en el país, permitiendo a personas físicas tener acceso a servicios financieros y poder invertir en valores gubernamentales con montos accesibles, sin comisiones, sin la intermediación de la banca, casas de bolsa u otras instituciones [Cetesdirecto, sf]. Los valores gubernamentales disponibles son CETES, BONOS, BONDES, BONDES F y UDIBONOS, por otro lado, los fondos de inversión a los que se puede acceder son BONDDIA y ENERFIN, entre otras clases de instrumentos financieros. En este trabajo se incorporaron al abanico de activos los siguientes valores financieros:

1. CETES: Certificados de la Tesorería de la Federación. Son instrumentos de deuda, es decir, representan una obligación por parte del emisor (en este caso el gobierno) de pagar al inversor una cantidad determinada de dinero en un plazo específico, junto con intereses. Los CETES tienen plazos disponibles de 28, 91, 182, 364 y 728 días. Los CETES son considerados como uno de los instrumentos de inversión más seguros en México y por ello se utilizan comúnmente como referencia para otros productos financieros. De este modo, esta herramienta financiera representa nuestro activo libre de riesgo y es introducido para lograr mayor diversificación en el portafolio.

2. Bonddia: fondo diario de Nacional Financiera (institución que administra las inversiones), invertido mayormente en instrumentos de deuda y complementariamente en bancarios, cuenta con liquidez diaria, es decir, si tus recursos se encuentran invertidos en Bonddia podrás disponer de ellos diariamente (días hábiles bancarios). Si no tienes claro qué día del mes necesitas tu dinero, en este fondo ganas un interés diario mientras permanece invertido.

3. ENERFIN: Fondo de Inversión de Renta Variable, invertido principalmente en instrumentos de deuda y como inversión complementaria en acciones de emisoras nacionales y extranjeras relacionadas con el sector de energía. Al ser de renta variable este instrumento puede tener ganancias o pérdidas. Este Fondo cuenta con un riesgo de inversión alto asociado principalmente al del mercado, en especial el riesgo en las tasas de interés, toda vez que las características de los valores que integran su cartera se encuentran sujetos a fluctuaciones a la alza y la baja en los mercados que cotizan, por lo que al presentar cambios pudiesen reflejar variaciones positivas o negativas en el precio del Fondo y en consecuencia generar minusvalías en la inversión realizada originalmente.

2.2. Precios de cierre y rendimiento diario

El precio de cierre de un activo es el precio al cual se cotizó por última vez durante el período de negociación del mercado financiero específico. Este precio se determina al final del día de negociación, justo antes del cierre del mercado. Las operaciones de la BMV se efectúan de lunes a viernes, iniciando a las 6:50 hrs y culminando a las 15:00 hrs, exceptuando días feriados [Grupo BMV, sfa]. Esto da como resultado un aproximado de 252 días de negociación al año.

El precio de cierre es importante porque proporciona una medida clave del rendimiento 'diario' de un activo, el cual puede ser utilizado en una variedad de análisis financieros y como herramienta para la toma de decisiones de inversión. Los rendimientos, a su vez, sirven para estimar el retorno esperado y el riesgo del portafolio de inversión.

El rendimiento diario de los activos se calcula de la siguiente manera:

$$r_d = \frac{P_t - P_{t-1}}{P_{t-1}} \ , \tag{1}$$

donde P_t es le precio de cierre del día actual y P_{t-1} es el precio de cierre del día anterior [Bodie et al., 2018].

2.3. Rendimiento anual y riesgo de los activos

Como es bien sabido, una amplia variedad de fenómenos de la vida cotidiana y la naturaleza pueden ser modelados estadísticamente mediante una distribución normal. En el caso de los mercados financieros, la distribución de rendimientos de un activo, ya sea que su periodicidad sea anual, mensual, semanal o por día de negociación, también tiende a una distribución normal, claro que con matices, ya que esta aproximación no es precisa cuando se consideran datos en un rango muy amplio de tiempo; en tales escenarios la asimetría en la distribución rompe la normalidad [Brealey et al., 2011]. Este comportamiento de los retornos permite calcular el rendimiento esperado como la media de los rendimientos históricos, por otro lado, la varianza y la desviación estándar de esos datos son una estimación del riesgo asociado al activo.

En el modelo implementado en este trabajo, el rendimiento anual esperado de cada activo se calcula como:

$$\bar{r} = \frac{1}{n}\sum_{d=1}^{n} r_d \ , \tag{2}$$

$$\xi_i = \bar{r} * 252 \ , \tag{3}$$

donde r_d es el rendimiento diario del activo en el día d, n es el número de rendimientos diarios que se están considerando, $\bar{r}$ es el rendimiento promedio, el factor 252 permite anualizar el rendimiento y ξ_i es el rendimiento anual esperado del i-ésimo activo. De esta manera, se esta asumiendo que el rendimiento promedio de los retornos históricos sera el rendimiento diario de los próximos 252 días de negociación.

Por otro lado, el riesgo de cada activo (la incertidumbre de que se aprecie o se deprecie) esta dado por:

$$\text{Varianza} = \sigma^2 = \frac{1}{n-1}\sum_{d=1}^{n}(r_d - \overline{r})^2 \ , \tag{4}$$

$$\text{Desviación estándar} = \sqrt{\sigma^2} = \sigma \ . \tag{5}$$

Ambas métricas proporcionan información sobre la dispersión de los rendimientos de un activo. Una mayor varianza o desviación estándar indica una mayor volatilidad y, por lo tanto, un mayor riesgo asociado con el activo. Pero, como suele suceder con la desviación estándar, es mas practico trabajar con ella pues tiene las mimas unidades que los datos analizados, en este caso, proporciones. Por otra parte, debido a que se contemplan los rendimientos anuales de los activos también es necesario anualizar el riesgo del portafolio, esto se efectúa multiplicando por la raíz del periodo de tiempo que se esta contemplando [de Trending, sf], en este caso, $\sqrt{252}$, de modo que la volatilidad anual del portafolio es

$$\text{Volatilidad anual} = \sqrt{252}\ \ \sigma \ . \tag{6}$$

2.4. Modelo de Media-Varianza

En 1952 fue publicado un innovador trabajo del economista Harry Max Markowitz sobre la optimización de portafolios. El modelo que desarrolló fue el primero en aplicar la varianza como medida del riesgo de una cartera de inversión, y actualmente se conoce como mean–variance (MV) model. Markowitz propuso manejar los rendimientos de cada activo como variables aleatorias y, en consecuencia, emplear el valor esperado y la varianza para cuantificar el rendimiento y el riesgo de la cartera, respectivamente [Markowitz, 1952]. Como se vera a continuación, el modelo MV es un problema de optimización con restricciones, dónde es posible minimizar el riesgo para un nivel de rendimiento esperado fijo o maximizar el rendimiento esperado para un nivel de riesgo dado. En el caso de maximizar el rendimiento para un nivel dado de riesgo, el problema de optimización esta dado como:

$$\max \sum_{i=1}^{N} w_i \xi_i \ , \tag{7}$$

Sujeto a las restricciones:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} w_i w_j \sigma_{ij} \leq \beta \ , \tag{8}$$

$$\sum_{i=1}^{N} w_i = 1 \ , \tag{9}$$

$$0 \leq w_i \leq w_{max}, i = 1, \ldots, N \ , \tag{10}$$

donde N es el número de activos en el portafolio, w_i es la proporción del capital que se asigna a cada activo, ξ_i es el rendimiento anual esperado de cada activo, σ_{ij} es la covarianza entre los rendimientos diarios de los activos, β el riesgo máximo permitido en el portafolio y w_{max} la máxima proporción de capital que se puede asignar a cada activo. Esto significa que el problema radica en encontrar los pesos w_i que maximicen el rendimiento del portafolio, garantizando que se satisfagan las restricciones dadas.

De acuerdo con la ecuación (7), el rendimiento esperado del portafolio es simplemente un promedio ponderado de los rendimientos esperados de los activos individuales. Por otro lado, para calcular el riesgo del portafolio es necesario considerar cómo afectan los cambios del mercado de manera simultanea al conjunto de bienes en el portafolio. La ecuación (8) considera la correlación lineal que existe entre los rendimientos de los activos, es decir, cómo se relaciona la variación del activo i con la variación del activo j. Esta relación se conoce como covarianza (σ_{ij}) y se calcula de la siguiente manera:

$$\sigma_{ij} = \rho_{ij}\sigma_i\sigma_j \ , \tag{11}$$

donde σ_i y σ_j son las desviaciones estándar de los activos y ρ_{ij} es el coeficiente de correlación de Pearson, el cual se obtiene de la siguiente manera:

$$\rho_{ij} = \frac{\sum_{d=1}^{n}\left(r(i)_d - \overline{r(i)}\right)\left(r(j)_d - \overline{r(j)}\right)}{\sqrt{\sum_{d=1}^{n}\left(r(i)_d - \overline{r(i)}\right)^2}\sqrt{\sum_{d=1}^{n}\left(r(j)_d - \overline{r(j)}\right)^2}} \ , \tag{12}$$

donde n es el número de rendimientos diarios históricos, $r(i)_d$ y $r(j)_d$ son los rendimientos diarios del activo i y del activo j, respectivamente. Por otro lado, $\overline{r(i)}$ y $\overline{r(j)}$ son los rendimientos promedio de cada activo. El resultado final ρ_{ij} estará en el rango de -1 a 1, donde -1 indica una correlación lineal negativa perfecta, 0 indica ausencia de correlación y 1 indica una correlación lineal positiva perfecta. En este caso, estamos interesados en conocer la covarianza y no únicamente el coeficiente de correlación, pues necesitamos una medida del riesgo, dado por la varianza (σ^2) y la desviación estándar (σ). Cuando existe una covarianza positiva entre dos activos, significa que tienden a aumentar o disminuir juntos, mientras que una covarianza negativa indica que se mueven en direcciones opuestas. Una covarianza cero, indica que no hay relación lineal entre los dos activos, es decir, el rendimiento de un activo no tiene ningún impacto previsible en el rendimiento del otro activo.

Cuando el objetivo es minimizar el riesgo para un rendimiento dado, el problema de optimización se formula de la siguiente manera:

$$\text{mín} \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j \sigma_{ij} \ , \tag{13}$$

sujeto a las restricciones dadas por las ecuaciones (9) y (10), y a la condición del rendimiento mínimo aceptado:

$$\sum_{i=1}^{N} w_i \xi_i \geq \eta \ , \tag{14}$$

donde η representa el rendimiento mínimo que se espera obtener. En este caso, se buscan los pesos w_i que minimicen el riesgo del portafolio.

La elección del riesgo máximo permitido o del rendimiento mínimo aceptado es una decisión arbitraria, depende del perfil del inversor, por lo que un inversor novato podría tener problemas para elegir dichos valores. Para evitar este conflicto se implementó un modelo alternativo al modelo MV original. Este nuevo modelo fue propuesto por Chang, Yang y Chang (2009), quienes introdujeron un parámetro de ponderación δ $(0 \leq \delta \leq 1)$, que funciona como un coeficiente de aversión al riesgo, de modo que el problema de optimización replantea como:

$$\text{máx} \ \ (1-\delta) \sum_{i=1}^{N} w_i \xi_i - \delta \sqrt{252 \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j \sigma_{ij}} \ , \tag{15}$$

sujeto a las mimas restricciones dadas por las ecuaciones (9) y (10). Nosotros agregamos la raíz cuadrada a la varianza para que las unidades de ambos términos sean iguales y dicha diferencia tenga sentido, esta raíz no se contempla en la publicación citada.

En la ecuación (15), el parámetro δ se puede ajustar de acuerdo con los objetivos del inversor: un inversor arriesgado elegiría $\delta = 0$, lo que conduciría a una maximización del rendimiento sin considerar el riesgo. En cambio, un inversor conservador podría elegir $\delta = 1$, haciendo que el modelo minimice el riesgo sin tener en cuenta el rendimiento del portafolio. Los valores intermedios de δ $(0 < \delta < 1\ )$ generan soluciones entre estos dos extremos $\delta = 0$ y $\delta = 1$. Con este enfoque, el inversor puede ajustar el balance entre riesgo y rendimiento, eligiendo que aspecto quiere priorizar.

2.5. Benchmarks

Habitualmente no es fácil definir lo que constituye un éxito en cuanto a la optimización de portafolios. Sin embargo, para estimar la calidad de una estrategia de inversión, el mercado financiero ofrece algunas opciones, que se detallan a continuación.

En general, un benchmark o punto de referencia, es un indicador utilizado para comparar y evaluar el rendimiento o la calidad de algo. En el argot de las finanzas, un benchmark es un índice de referencia que se utiliza para comparar el rendimiento de una inversión o cartera con un estándar externo. Este estándar puede ser un índice de mercado, un sector específico o un producto similar. En otras palabras, los benchmarks proporcionan un marco de referencia para medir el éxito de una estrategia de inversión en relación con un mercado o un índice de referencia específico. Por lo tanto, para evaluar las soluciones dadas por el algoritmo evolutivo, se comparó el rendimiento y el riesgo de los portafolios obtenidos respecto de valores conocidos de productos financieros ofrecidos por Vanguard.

Vanguard es una de las mayores empresas de gestión de inversiones del mundo, fue pionera en la democratización de la inversión, permitiendo a inversores de todos los niveles de experiencia acceder a carteras diversificadas y de bajo costo [8]. Sus productos principales son ETFs (Exchange Traded Funds): instrumentos de inversión que combinan características de fondos mutuos [1] y acciones individuales, es decir, al igual que los fondos mutuos, los ETFs invierten en una canasta de activos, que pueden ser acciones, bonos, materias primas u otros instrumentos financieros, lo que permite a los inversionistas acceder a una cartera diversificada con una sola inversión, por otro lado, a diferencia de los fondos mutuos, los ETFs se negocian en bolsas de valores como cualquier acción individual, de modo que su precio fluctúa a lo largo del día según la oferta y la demanda.

Los ETFs de Vanguard están diseñados para replicar el rendimiento de otros índices, por ejemplo, uno de lo más conocidos y quizá el más utilizado es el índice Standard & Poor's 500 (S&P 500) que agrupa las 500 empresas más grandes y representativas de diversos sectores de la economía estadounidense (las empresas más grandes tienen un peso mayor en el índice), por lo que es considerado un indicador representativo del mercado de valores de EU. Los índices proporcionan información de toda la colección de activos como si se tratara de uno solo, es decir, el rendimiento y el riesgo que proporciona el S&P 500 corresponden a las 500 acciones en conjunto. Por lo tanto, cuando se dice que un fondo de inversión o un ETF replica el comportamiento de un índice, implica que el fondo intenta reflejar los movimientos del índice en términos de rendimiento y volatilidad. Para lograr esto, el fondo suele invertir en una cartera de activos que se asemeja lo más posible a la composición del índice. Por ejemplo, para replicar el S&P 500, se invertirá en una selección de acciones de las 500 empresas, en las mismas proporciones o pesos que tienen esas acciones en el índice.

Vanguard comparte de manera pública la información financiera relevante de sus productos,

[1] Un fondo mutuo, también conocido como fondo de inversión colectiva o simplemente fondo, es una estructura de inversión que reúne el dinero de muchos inversores para invertir en una variedad de activos financieros, como acciones, bonos, valores del mercado monetario y otros instrumentos financieros.

en ella compara los rendimientos y el riesgo de sus carteras respecto del índice que intenta replicar, esto proporciona parámetros reales del mercado respecto de los cuales se pueden comparar los resultados obtenidos con el algoritmo evolutivo.

Los ETFs que se utilizaron como punto de referencia son los siguientes:

- Vanguard FTSE BIVA Mexico Equity ETF (VMEX): esta compuesto por 43 acciones de empresas que cotizan en la BMV y busca replicar el rendimiento del FTSE BIVA Index, el cual está diseñado para representar de manera precisa y completa el desempeño del mercado de valores mexicano al incluir una amplia muestra de empresas que cotizan en el mercado mexicano. De acuerdo la ficha técnica, en 2023 este ETF tuvo un rendimiento anual del 20.80 %, mientras que el índice de referencia tuvo un 21.30 %. Para el 31 de marzo de 2024 los rendimientos del ETF presentaban una desviación estándar (riesgo) del 18.10 % y el índice de referencia del 18.11 % [Vanguard Mexico, sf].
- Vanguard S&P 500 ETF: este instrumento busca replicar el rendimiento del S&P 500 Index. Ambos cerraron el 2023 con un rendimiento anual del 24.25 % y del 26.29 %, respectivamente. Para el 31 de marzo de 2024 cada uno presentaba una desviación estándar de 17.60 % [Vanguard Mexico, sf].

3. Implementación

En esta sección se describen los elementos más importantes del proyecto: la arquitectura del algoritmo evolutivo, la construcción del DataSet con datos reales del mercado financiero, para evaluar el rendimiento del algoritmo, y los algoritmos empleados para medir el rendimiento.

3.1. Algoritmo Evolutivo

- **Representación:** se optó por una representación real en el intervalo $[0, 1]$, donde cada gen indica el porcentaje del capital destinado a la inversión en el activo respectivo. Esta representación es la más natural en el contexto del problema, y permite expresar las restricciones así como la función de aptitud mediante expresiones matemáticas sencillas. Además, todas las posibles carteras de inversión pueden ser expresadas mediante esta representación.
- **Aptitud (fitness):** la función de aptitud seleccionada, como se describió en la sección 2.4, expresa una diferencia ponderada entre los rendimientos esperados de la cartera y su riesgo intrínseco. La ponderación se realiza mediante el parámetro δ, el cual expresa la disposición del inversionista a asumir un mayor o menor riesgo. Un valor de δ=0 indica que el inversionista está dispuesto a afrontar cualquier riesgo con tal de maximizar los retornos esperados, mientras que δ=1 indica que lo único que importa es minimizar el riesgo, sin importar el retorno esperado de la cartera. Para facilitar la implementación en el código, se expresó la proporción del capital asignado a cada activo, así como los rendimientos esperados, mediante los vectores columna w, y ξ respectivamente, de modo

que la función de aptitud se puede escribir como sigue:

$$f(w;r) = (1-\delta)w^T\xi - \delta\sqrt{252w^T\sigma_{ij}w} \ , \tag{16}$$

con σ_{ij} la matriz de covarianza entre los rendimientos diarios de los activos en el portafolio.

Finalmente, para prevenir violaciones de la restricción $w_i < w_{max} \quad \forall i$, dónde w_i es el valor del i-ésimo gen, se añadió una penalización de $20(w_i - w_{max})$ para cada $w_i > w_{max}$. El coeficiente de 20 se determinó de manera experimental.

Esta penalización, al ser proporcional al valor excedente de w_i, permite que las soluciones que se exceden ligeramente sigan siendo consideradas, ya que valores cercanos al límite suelen generar mejores resultados. Por ejemplo, si un activo tiene una excelente relación retorno-riesgo, debería invertirse la mayor cantidad posible en ese activo. En contraste, los valores que superan ampliamente el límite se penalizan de manera proporcional, reduciendo su probabilidad de reproducción.

- **Inicialización:** se inicializa la población de forma aleatoria, con una distribución uniforme entre 0 y el valor asignado como peso máximo, w_{max}. Luego se normalizan los cromosomas dividiendo cada gen entre la suma total de los genes en el cromosoma. De este modo se garantiza que la suma de los genes es igual a 1 y se reproduce el mecanismo que sucedería en un escenario real en el que, si se desea aumentar el porcentaje de inversión a un activo en específico, se necesita reducir los porcentajes de otros activos. Adicionalmente se añadió un filtro en la inicialización el cual descarta cualquier cromosoma que tenga algún gen con un valor mayor a w_{max}. Esto implica que se debe realizar una elección cuidadosa del valor de w_{max} según el número de activos disponibles, pues un valor demasiado bajo podría resultar en ciclo infinito debido a que no hay activos suficientes para lograr que los genes del cromosoma sumen 1.
  Por último, se añade un gen adicional para controlar el tamaño del torneo de selección de sobrevivientes, cuyo valor se elige de manera aleatoria con una distribución uniforme entre 0 y 1.

- **Selección de padres:** se optó por una selección de padres uniforme, donde cada individuo tiene la misma probabilidad de ser seleccionado. De esta manera, la presión de selección se delega completamente a la etapa de selección de sobrevivientes. Se eligió la selección uniforme por encima de la selección por "ranking" o basada en fitness debido a que en la pruebas se observó que estas otras generaban presión de selección demasiado alta, dada la elección de la selección de sobrevivientes. Por otro lado, la población sigue un modelo generacional, utilizando un mecanismo de torneo y selección (μ, λ), donde se generan $\lambda(> \mu)$ hijos a partir de μ padres. Por esta razón, se seleccionan $\lambda > \mu$ padres, lo que implica que hay padres que se seleccionan múltiples veces.

- **Recombinación:** la recombinación se efectúa mediante el operador "BLX-α", con un parámetro $\alpha = 0{,}5$, de acuerdo a las siguientes expresiones:

$$\gamma = (1-2\alpha)u - \alpha \ ,$$

$$x_1 = (1-\gamma)p_1 + \gamma p_2 \ ,$$
$$x_1 = (1-\gamma)p_2 + \gamma p_1 \ ,$$

donde u es una variable aleatoria uniforme entre 0 y 1, p_1 y p_2 son los padres y x_1 y x_2 los hijos. Los hijos resultantes de este operador son sometidos a una normalización como se describió anteriormente.
Este operador de cruce, a diferencia de muchos otros, introduce una mayor diversidad en la población al generar valores de genes que antes no estaban presentes, lo cual ayudó a lograr un mejor equilibrio entre la exploración y la explotación en el proceso evolutivo. Además, a diferencia de otras recombinaciones aritméticas comúnmente utilizadas en representaciones reales, bajo esta configuración los genes de los hijos tienen la misma probabilidad de estar dentro o fuera del rango definido por los genes de los padres, por lo que el espacio alcanzable no está limitado por la población actual.

- **Mutación:** la mutación se divide en dos partes. Para los genes que representan el porcentaje de inversión, se utiliza una perturbación Gaussiana con una probabilidad de mutación p_m y una desviación estándar σ. Esto busca principalmente realizar cambios pequeños, pero con la posibilidad de generar cambios más significativos. Por motivos que se expondrán más adelante se eligieron los valores $p_m = 0{,}02$ y $\sigma = 0{,}1$.
Para el gen adicional, se utiliza una fórmula específica:

$$k' = \left(1 + \frac{1-k}{k} \cdot e^{-\gamma \cdot N(0,1)}\right)^{-1} \quad , \tag{17}$$

donde k es el parámetro que determina el tamaño del torneo a utilizar durante la selección de sobrevivientes, $N(0,1)$ es una distribución normal con media 0 y desviación estándar 1, y $\gamma = 0{,}22$ es el tamaño de paso que controla la adaptación, según resultados previos obtenidos en [Bäck and Schütz, 1996] y [Eiben et al., 2006] . De esta manera, se obtienen valores en el rango de 0 a 1, favoreciendo los cambios pequeños sobre los grandes. Es importante mencionar que el algoritmo aplica el operador de recombinación al cromosoma extendido en su totalidad, mientras que la mutación se aplica exclusivamente al gen adicional correspondiente al tamaño de torneo. Esto significa que el hijo creado por la recombinación hereda un valor inicial de sus padres, pero el valor definitivo, k', se determina mediante la mutación.

- **Selección de sobrevivientes:** el reemplazo combina criterios de edad y aptitud: todos los padres son descartados después de una generación y ningún individuo se mantiene por más de una. La selección se lleva a cabo de la siguiente manera:

  1. Se obtiene el tamaño de torneo, K, a utilizar según la siguiente expresión
  $$K = \left\lceil \sum_{i=1}^{N} k_i \right\rceil \ ,$$
  donde k_i es el valor del gen adicional del i-ésimo padre seleccionado.
  2. Se llevan a cabo μ torneos de tamaño K, eligiendo en cada uno a los K participantes de manera aleatoria y sin reemplazo de entre los λ padres seleccionados.

3. En cada torneo gana el individuo con mayor aptitud.
4. Los μ ganadores de torneos conforman la nueva población.

Se optó por la selección de tipo (μ, λ) en vez de $(\mu+\lambda)$ ya que es preferible si el algoritmo tiene parámetros autoadaptativos [Eiben and Smith, 2015], como es en este caso el tamaño del torneo.
Este tipo de selección tiende a generar una gran presión de selección, la cual es regulada mediante el tamaño de torneo. Por esta razón se optó por una selección de padres uniforme (sin presión de selección) la cual permite al tamaño de torneo autoadaptativo regular la presión de selección según se necesite.

3.2. Construcción del DataSet

El algoritmo requiere como datos entrada los rendimientos diarios de cada activo que forma parte del portafolio. Estos rendimientos se calculan a partir de los precios de cierre, los cuales se pueden obtener de manera gratuita en diversas fuentes. En este caso, se empleó la API (Application Programming Interface) de Yahoo Finance, la cual proporciona datos financieros. Para poder acceder a las funciones de esta API se debe importar la biblioteca yfinance a nuestro entorno. Una vez instalada, es posible obtener los precios de cierre de cualquier activo en un mercado determinado, simplemente se debe indicar el ticker (símbolo bursátil o código de cotización) del activo y la información que se desea obtener: el precio de apertura ('Open'), el precio máximo ('High'), mínimo ('Low'), el precio de cierre ('Close') y el precio de cierre ajustado ('Adj Close') del activo para cada día, entre muchos otros datos. El mercado financiero donde cotiza cada unos de los bienes se especifica con un sufijo al final del ticker, los activos que cotizan en la BMV se identifican con el sufijo .MX, .L para la Bolsa de Valores de Londres, .TO para la Bolsa de Toronto, etc.

Del sitio oficial de la Bolsa Mexicana de Valores se obtuvieron los tickers que identifican a los activos nacionales e internacionales que se pueden comerciar en el mercado mexicano, empleando estas etiquetas se descargaron los datos desde Yahoo Finance. Del mercado nacional se consideraron acciones y FIBRAS, y de los activos extranjeros se consideraron acciones únicamente. Para completar nuestro espacio muestral de activos, se agregaron los productos financieros proporcionados por el Gobierno de México: CETES, BONDDIA y ENERFIN. Los precios de cierre de estos instrumentos se obtuvieron del sitio oficial del Banco de México [BANXICO, sf]. Los datos obtenidos van desde el 1 de enero de 2021 hasta el 4 de abril de 2024.

Después de filtrar los datos, nuestro espacio muestral quedó de la siguiente manera: 91 acciones nacionales, 14 FIBRAS y 290 acciones internacionales; además de los tres instrumentos gubernamentales, dando un total de 398 activos. El código correspondiente a este apartado puede consultarse en el Apéndice B.

3.3. Tamaño mínimo del portafolio

Para definir el conjunto de activos potenciales a incluir dentro del portafolio se determinó previamente que el riesgo máximo a tolerar sería de 0.18, lo cual corresponde aproximadamente al benchmark que se utilizó como referencia para medir el desempeño del modelo. Este valor corresponde en general a las preferencias individuales del inversor y a sus actitud respecto a seguir estrategias más arriesgadas con rendimientos potencialmente más altos o más seguras con menores rendimientos. Una vez definido el riesgo máximo, se realizaron gráficas del retorno esperado contra el riesgo para conjuntos de activos de distintos tamaños y utilizando distintos valores de w_{max}, los cuales establecen el número mínimo de activos que serán incluidos en la cartera ($\lceil 1/w_{max} \rceil$) y por lo tanto determinan su grado mínimo de diversificación. Los resultados obtenidos, que se muestran en las gráficas siguientes, muestran que un w_{max} más pequeño genera rendimientos inferiores de manera consistente. Esto corresponde a una diversificación excesiva del portafolio. Al contrario, una menor diversificación tiende a generar mayores rendimientos. Por otro lado, por lo general se considera que una menor diversificación conlleva un mayor riesgo, cosa que no se ve particularmente reflejada en la figura, quizás por la naturaleza estocástica del método. Sin embargo, siguiendo la sabiduría de los expertos, se optó por w_{max} ni muy alto ni muy bajo, de 0.07, el cual corresponde a un mínimo de 15 activos en el portafolio. Este valor parece generar riesgos inferiores (o muy cercanos, en el caso de los 100 activos) al límite establecido de 0.18 (la gráfica muestra en rojo un riesgo de 0.2).

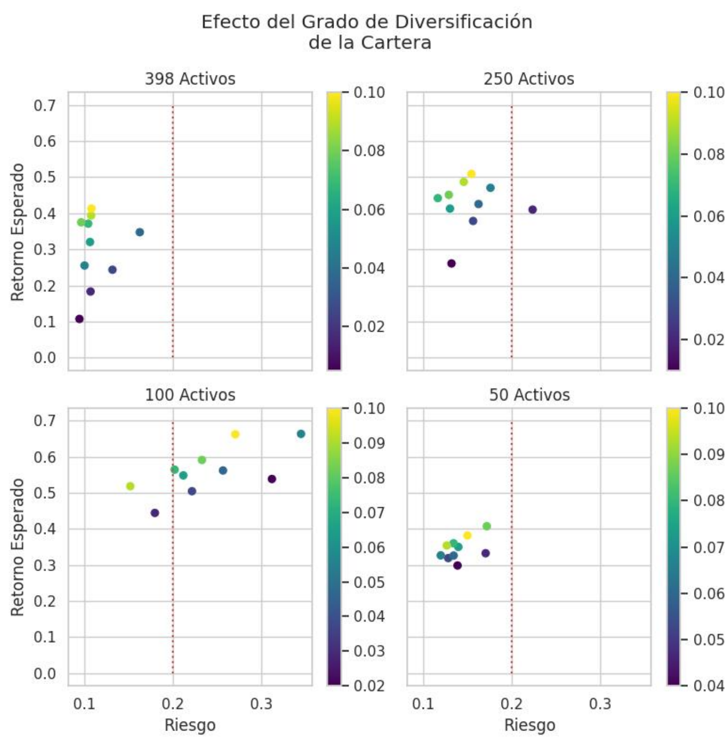

Figura 1: *Mayores valores de w_{max}, correspondientes a una menor diversificación, generan mayores rendimientos. Contrario a la sabiduría de los expertos, en este caso no se observó un incremento del riesgo al disminuir la diversificación.*

Por otro lado, la figura 1 sugiere que al considerar únicamente 100 activos se pueden obtener mejores rendimientos. A continuación exploramos esta idea.

3.4. Espacio muestral

Una vez definido el valor de w_{max}, se comparó el MBF (Mean Best Fitness) obtenido por el algoritmo para distintos subconjuntos de los 398 activos disponibles de la siguiente manera:

- Se calculó para cada activo la siguiente cantidad

$$ponderacion = \frac{rendimiento\ promedio}{riesgo}\ , \tag{18}$$

  es decir, para cada activo se realizó el cociente entre su rendimiento promedio y su riesgo, dando una mejor 'calificación' o *ranking* a activos cuyo rendimiento promedio es grande en relación al riesgo que suponen.

- Se ordenaron los activos en orden descendente según la ponderación obtenida en el punto anterior.

- Se obtuvo el MBF al considerar los 398 activos, así como subconjuntos de los mejores 250, 100 y 50, según el criterio mencionado.

- Se obtuvo el MBF para muestras aleatorias de tamaño 250, 100 y 50.

Los resultados del proceso anterior se muestran en la siguiente tabla, dónde el valor de cada celda representa el MBF obtenido.

# de activos	Selección: Determinista	Selección: Aleatoria
398	0.1703	0.1684
250	0.1889	0.1449
100	0.1927	0.1028
50	0.1119	0.0769

De acuerdo a los resultados obtenidos, se optó por utilizar el subconjunto de los mejores 100 activos de acuerdo a su tasa de rendimiento contra riesgo. De estos 100, por lo menos 15 estarán presentes en el portafolio, de acuerdo al valor de w_{max} elegido.

3.5. Retornos vs. Riesgo

Utilizando los parámetros obtenidos ($w_{max} = 0{,}07$ y espacio muestral de los 100 'mejores' activos) se corrió el algoritmo utilizando distintos valores de δ, el cual se encarga de asignar mayor o menor importancia a los retornos esperados o al riesgo asociado en la función de aptitud. Los resultados de la figura 2 muestran que a mayor δ, menor fitness, alcanzando incluso valores negativos cuando se le asigna mucha mayor importancia al riesgo en lugar de al rendimiento.

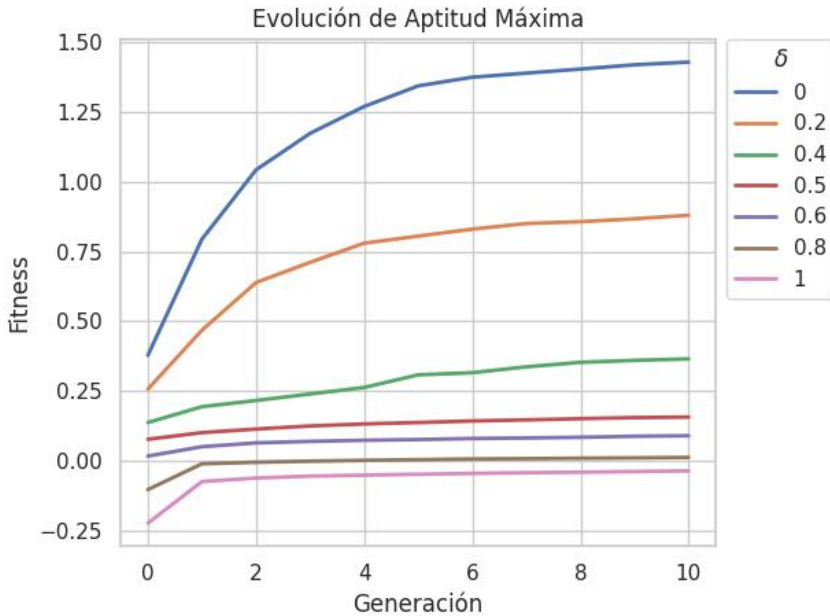

Figura 2: *El fitness disminuye conforme se le da más importancia al manejo del riesgo. Esto se debe al factor de $(1-\delta)$ que multiplica al término asociado con el rendimiento de la cartera.*

Sin embargo, lo que nos interesa realmente es maximizar el rendimiento, sin exceder cierto valor de riesgo. En este sentido, la figura 3 muestra que al darle mayor importancia al rendimiento este tiende a aumentar significativamente, pero lo mismo sucede con el riesgo. De igual manera, al darle mayor importancia al riesgo este alcanza valores muy bajos, a costa de disminuir el rendimiento. El valor de $\delta = 0{,}5$ resultó en el mejor equilibrio entre retornos esperados y riesgo, por lo que se utilizó este valor en el resto del trabajo.

3.6. Tasa de mutación

La tasa de mutación es otro parámetro que se debe ajustar o controlar. Hasta ahora se había optado por $p_m = 0{,}01$ de acuerdo al número de activos en el espacio muestral (100) con la intención de que ocurriera en promedio una mutación por cromosoma, promoviendo la diversidad de la población sin ser demasiado destructiva. La figura 4 muestra que, en efecto, valores muy bajos de p_m generan un crecimiento muy lento de la aptitud, probablemente debido a una alta homogeneización de la población, la cual conlleva a una convergencia prematura. Por otro lado, valores demasiado altos no permiten que buenas soluciones produzcan influencias positivas sobre la población ya que tanta mutación destruye los patrones que podrían favorecer a nuevos individuos, previniendo así la convergencia.

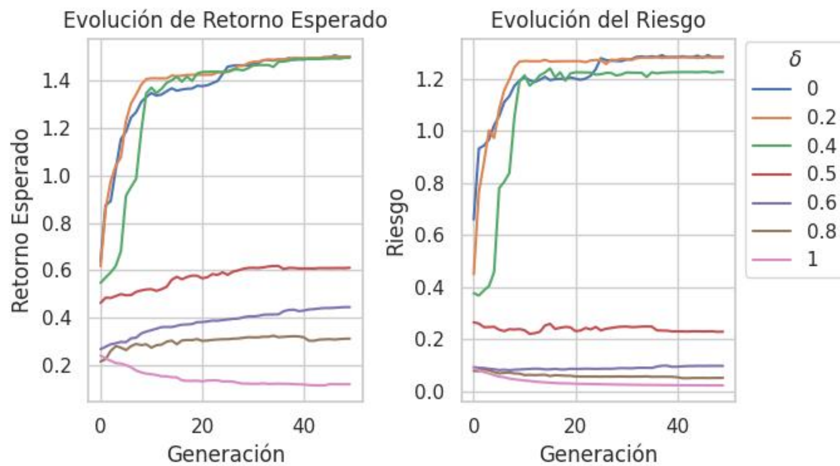

Figura 3: *El rendimiento y el riesgo crecen y decrecen de manera similar al variar la importancia que se le asigna a cada uno.*

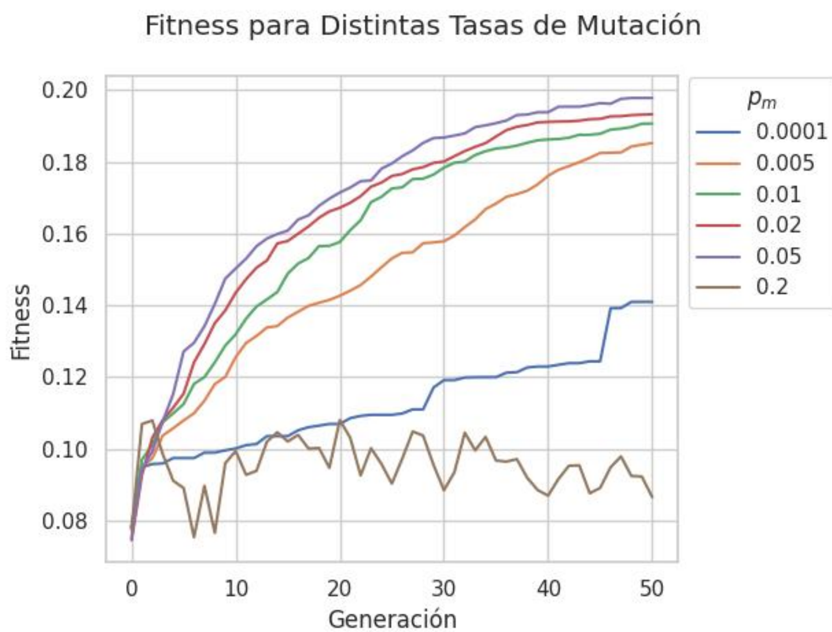

Figura 4: *Valores muy bajos de p_m no producen suficiente diversidad, por lo que el algoritmo puede converger prematuramente a soluciones subóptimas. Por otro lado, valores muy altos son demasiado destructivos y no permiten la convergencia a buenas soluciones.*

Valores de entre 0.005 y 0.05 parecen producir resultados similares, pero a lo largo de repetidas pruebas se notó una ligera superioridad para $p_m = 0{,}05$.

3.7. Tamaño de Población

La solución que se está buscando vive en el espacio $[0, w_{max}]^{100}$. Al ser un espacio tan grande, cabe esperar que para lograr una buen exploración en la mayor cantidad posible de regiones se requiere de una población también bastante grande. En la figura 5 se observa que, efectivamente, tamaños de población mayores logran en general encontrar mejores soluciones. Además, lo logran en un menor número de generaciones, pues para cada punto de la gráfica se corrió el algoritmo por $100/log(\mu)$ generaciones. No obstante, el número de evaluaciones incrementa linealmente con el tamaño de la población; además, se tomó $\lambda = 6\mu$, por lo que también aumenta el número de recombinaciones, mutaciones, torneos, etc. Por todas estas razones, el tiempo de ejecución aumenta considerablemente.

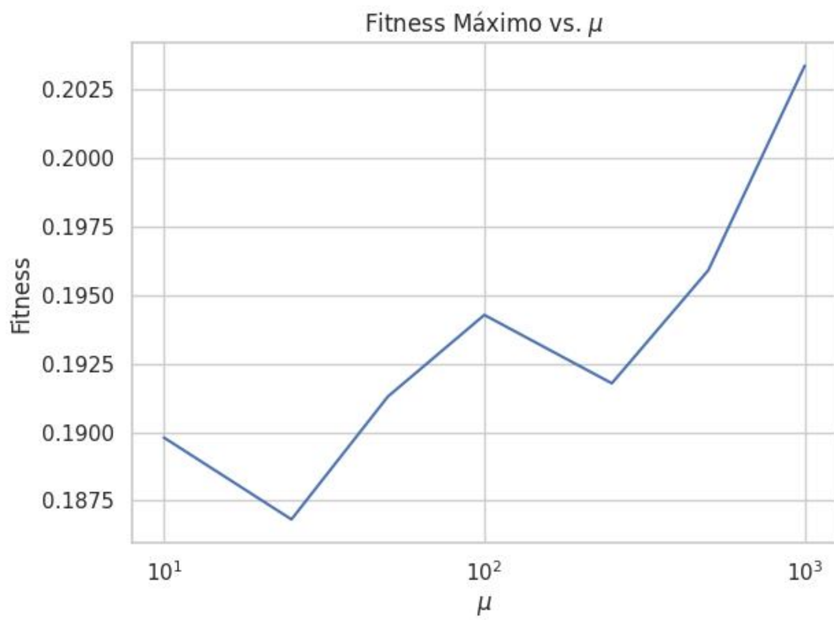

Figura 5: *El fitness máximo obtenido aumenta con el tamaño de la población, μ. Esto se debe a que hay una mayor exploración del espacio de soluciones.*

Puesto que la mejora en los resultados con $\mu = 1,000$ y $\mu = 10,000$ es pequeña, y la diferencia en el tiempo de ejecución es grande, el resto de las pruebas de este trabajo se realizaron con $\mu = 1,000$. Sin embargo, puesto que en una aplicación real del método se cuenta con al rededor de 16 horas para ejecutar el algoritmo, convendría utilizar tamaños de población incluso mayores, de hasta 100,000 o más individuos.

3.8. Evaluación del modelo

El desempeño de un algoritmo evolutivo puede evaluarse según diversos criterios, dependiendo del objetivo específico del problema que busca resolver.

En el contexto de la optimización de portafolios de inversión, generalmente no existen restricciones de tiempo demasiado estrictas. La Bolsa Mexicana de Valores (BMV) está cerrada casi 16 horas al día, sin incluir los fines de semana, lo que permite ejecutar el algoritmo durante el horario de cierre con datos completamente actualizados. No obstante, se puede tomar este periodo de tiempo (o uno similar) como límite para la ejecución del algoritmo.

Para este trabajo se realizó una prueba más pequeña, tomando un límite de 50 generaciones, y se calculó el MBF de 10 ejecuciones cada una en los mismos 100 activos seleccionados según su ranking. Se utilizaron los parámetros $w_{max} = 0{,}07$, $p_m = 0{,}05$, $\sigma = 0{,}1$, $\alpha = 0{,}5$, $\delta = 0{,}51$ (ya que demostró reducir lo suficiente el riesgo sin afectar demasiado al rendimiento), $\mu = 1,000$ y $\lambda = 6,000$. El resultado obtenido fue un MBF de 0.194, el cual corresponde a un rendimiento promedio de 0.645 (con desviación estándar 0.0383) con un riesgo promedio de 0.257 (desviación estándar 0.0).

Por otro lado, la existencia de Benchmarks como lo son el VMEX y el Vanguard S&P 500 ETF ofrecen posibles definiciones de *éxito* que se pueden utilizar para medir el AES (Average Evaluations to Success) y el SR (SR).

El SR es la métrica más valiosa en el contexto del problema, pues existe un límite de tiempo (aunque sea relativamente largo) e interesa obtener buenos resultados de manera consistente. No sirve de mucho que el algoritmo ocasionalmente encuentre un portafolio excepcional si mayoría de las veces el portafolio generado resulta en rendimientos deficientes o incluso negativos.

Para medir el SR tomamos como referencia el VMEX, que tiene por rendimiento anual de 20.8 % con un riesgo anualizado de 18.1 %. Ambas condiciones se consideraron como criterio para el éxito. Bajo estas condiciones, con los mismos parámetros utilizados para medir el MBF pero con un total de 30 ejecuciones del algoritmo, se obtuvo un SR de 0.767 con un AES de 7784. La figura 6 muestra que en todos los casos se superó el rendimiento del VMEX. Sin embargo, sólo en el grupo que aparece abajo a la izquierda lo logró sin exceder el riesgo permitido. Una característica interesante, para la que no encontramos una explicación, es el hueco que hay entre los dos grupos de puntos.

Aunque un SR de 76.7 % puede parecer relativamente alto, confiar ciegamente en los resultados de este algoritmo podría llevar a grandes pérdidas de capital si no se utiliza con cuidado. Para mitigar este riesgo se podrían utilizar valores mayores de δ y seguir una estrategia más conservadora, o correr el algoritmo varias veces hasta que se encuentre una solución que satisfaga ambas condiciones (altos retornos a bajo riesgo) de manera simultánea. Esto, no obstante, podría no ser factible si se desea correr el algoritmo con una población extremadamente grande para intentar encontrar la mejor solución posible dentro de las 16 horas en que está cerrada la BMV. El usuario debería considerar todos estos factores, así como sus recursos y su tiempo para decidir cuál estrategia es la mejor en su caso específico.

Por otro lado, las medidas de desempeño de un algoritmo son útiles para cuantificar su habilidad para resolver un problema específico, pero no ofrecen información sobre la capacidad

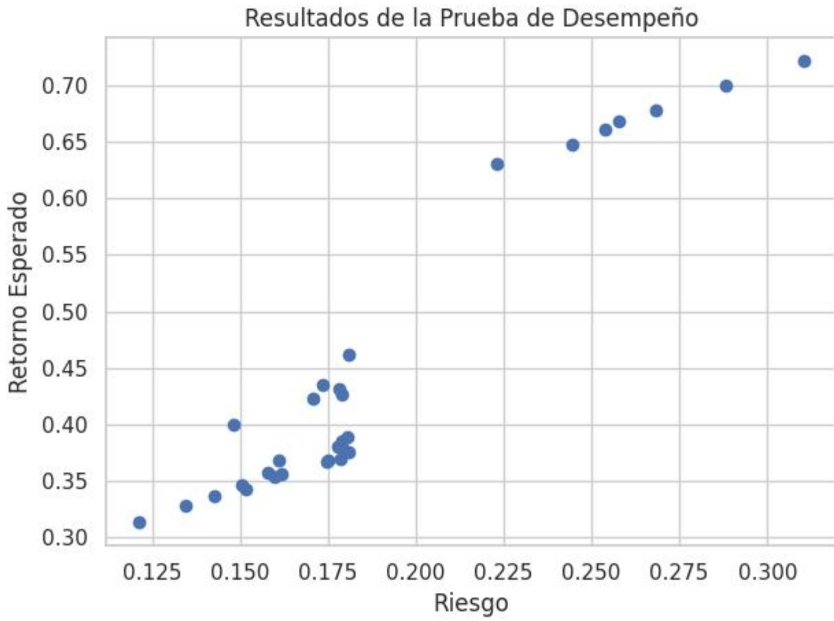

Figura 6: *Aunque todas las corridas del algoritmo superaron el rendimiento del VMEX, sólo 76.7 % lo lograron sin exceder el riesgo límite de 18.1 %.*

del algoritmo para generalizar a diferentes instancias del mismo problema o a problemas distintos. Tampoco proporcionan detalles sobre cómo cambia el desempeño del algoritmo al modificar sus parámetros, ya sean numéricos o simbólicos. Para esto sirven las medidas de robustez.

Al igual que las medidas de desempeño, la manera en que se mide la robustez siempre dependerá del problema: en algunos casos los datos son fijos, mientras que en otros pueden variar. Por otro lado, un problema puede considerarse como un problema "tipo", donde ciertos parámetros pueden cambiar sin alterar el esquema general del problema, como sucede en el caso de las ecuaciones cuadráticas.

En el contexto de la optimización de portafolios de inversión, el problema mantiene una estructura constante, pero los datos varían dependiendo de la bolsa en la que se planea invertir. Por ejemplo, los activos disponibles en la Bolsa Mexicana difieren de aquellos en la Bolsa Italiana. Además, los precios de los activos se actualizan diariamente durante los días hábiles. Por estas razones, la medida de robustez más apropiada se relaciona con la base de datos. Esto se puede evaluar de dos maneras: en términos del conjunto de activos seleccionados y en relación con las fechas de los precios de cierre considerados.

La figura 7 muestra la evolución del fitness al utilizar como conjunto de activos disponibles 5 distintas muestras aleatorias de 100 activos cada una, obtenidas a partir del total de 398 activos descargados.

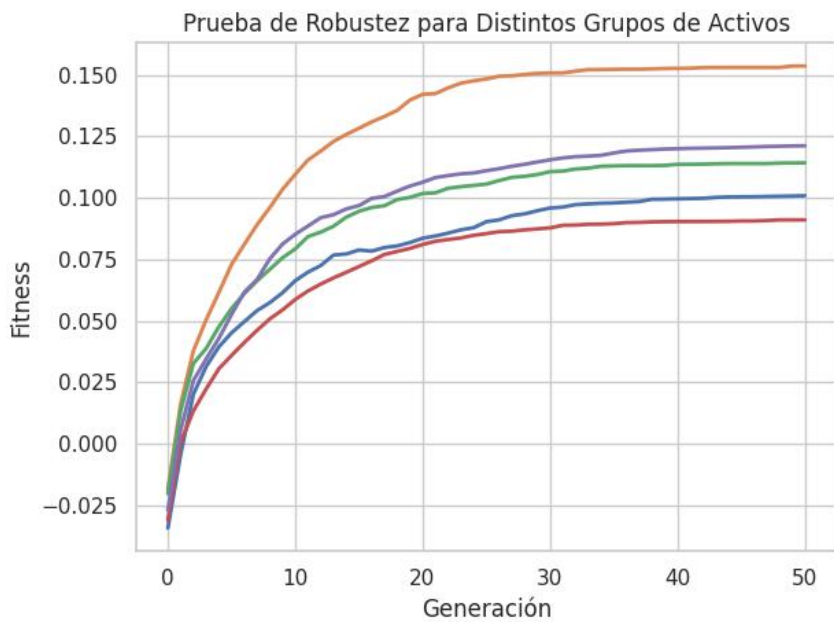

Figura 7: *El comportamiento del fitness es muy similar para distintos conjuntos de activos. Esto sugiere que el algoritmo es generalizable a distintos conjuntos de activos, como pudiera ser al aplicarse en distintas bolsas de valores.*

Resulta claro que, a pesar de que la calidad del resultado varía para distintas muestras de activos, el comportamiento general es muy similar en todos los casos. Esto sugiere que el algoritmo funciona de manera adecuada independientemente de los activos disponibles, y que un mayor o menor rendimiento puede ser obtenido dependiendo de las particularidades de cada conjunto de activos.

Por otro lado, un escenario más realista es el de un inversionista que cotiza regularmente en una bolsa en específico y en distintos momentos del año (incluso diariamente). La figura 8 muestra la evolución del mejor fitness al aplicar el algoritmo al mismo conjunto de activos, pero considerando sus rendimientos en distintos periodos de tiempo. En cada caso se realizó previamente a la aplicación del algoritmo un ranking de los 100 mejores activos según su relación rendimiento-riesgo, como se describió anteriormente, con lo cual en cada periodo no sólo se consideran distintas fechas, sino también potencialmente distintos conjuntos de activos, siendo estos los mejores en su respectivo periodo de tiempo.

Los resultados de la figura 8 muestran que los rendimientos obtenidos varían más con el momento en el que se decide invertir que con variaciones en el conjunto de activos disponibles, lo cual refleja el hecho de que, en la vida real, hay momentos mejores que otros para comprar y vender activos financieros.

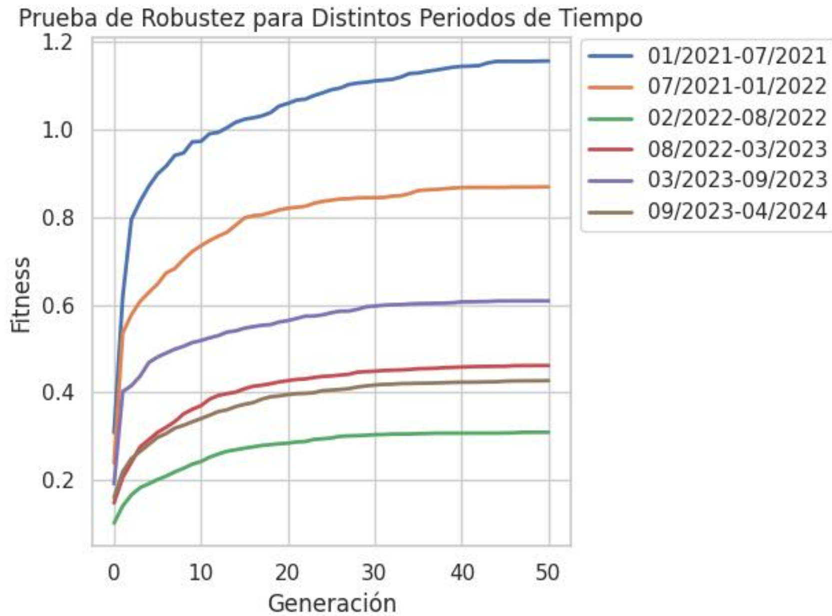

Figura 8: *El rendimiento máximo obtenido depende mucho del periodo de tiempo considerado. Esto refleja el hecho de que las bolsas de valores tienen temporadas mejores que otras. La similitud entre las evoluciones de los fitness sugieren que el algoritmo está funcionando adecuadamente en todos los casos.*

3.9. Ejemplo

Para simular un ejemplo como si se estuviera utilizando el algoritmo en un escenario real se ejecutaron 50 generaciones en el conjunto de los 100 mejores activos, con $w_{max} = 0{,}07$, $\alpha = 0{,}5$, $p_m = 0{,}05$, $\sigma = 0{,}1$ y $\delta = 0{,}51$, con una población de tamaño $\mu = 2,000$ y $\lambda = 12,000$.

Se obtuvo un fitness final de 0.155, correspondiente a un retorno esperado de 45.78 % con un riesgo del 13.50 %. La evolución del fitness se muestra en la figura 9

Por otro lado, como se muestra en la figura 10 en escala logarítmica, la diversidad decae casi a un 0.5 % de su valor original en las primeras pocas generaciones, tras lo cual decae gradualmente, con incrementos espontáneos causados por los operadores de mutación y recombinación. Este comportamiento sugiere que sigue habiendo cierto grado de exploración incluso en generaciones avanzadas, aunque el patrón general es de convergencia, pues la diversidad tiende a ir en descenso. Este hecho justifica la selección de padres uniforme, pues otra alternativa intensificaría esta pérdida de diversidad.

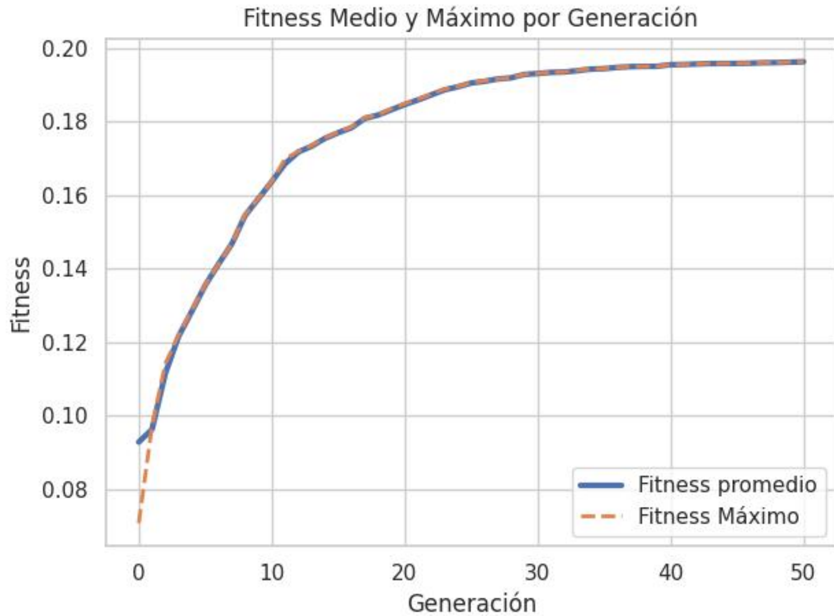

Figura 9: *Ejemplo típico de la evolución del fitness medio y máximo en cada iteración.*

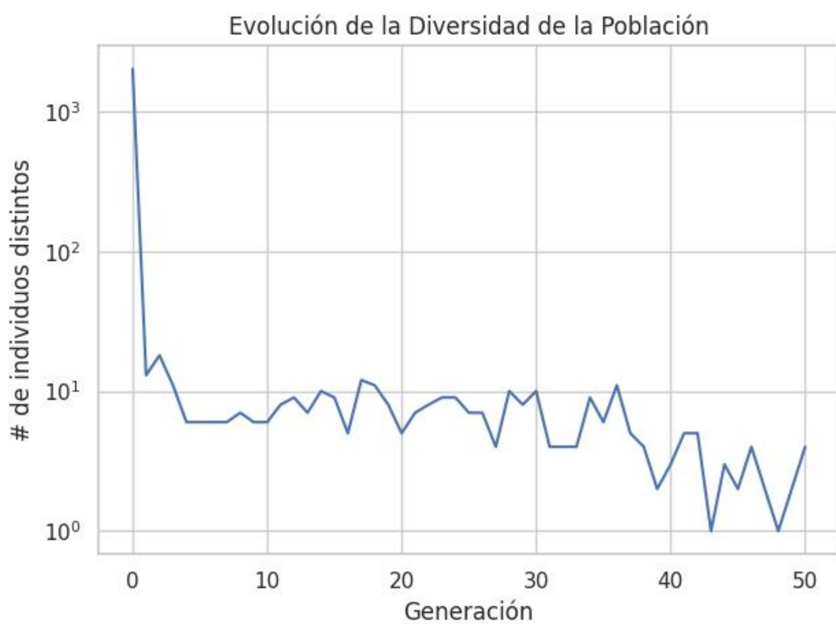

Figura 10: *Disminución de la diversidad durante el proceso de evolución.*

El portafolio resultante se muestra en la tabla 1, dónde se ve que todos los pesos están entre 0 y $w_{max} = 0{,}07$, y que muchos de los activos seleccionados tienen pesos cercanos a este valor máximo.

FIHO12.MX	0.021298	GFINBURO.MX	0.039467
CIEB.MX	0.069857	ALSEA.MX	0.066133
GENTERA.MX	0.069214	VGK.MX	0.013783
MEDICAB.MX	0.069255	BBAJIOO.MX	0.064605
CHDRAUIB.MX	0.069718	FAZ.MX	0.035317
LAMOSA.MX	0.069963	LIT.MX	0.013422
BAFARB.MX	0.069739	VISTAA.MX	0.069639
MRO.MX	0.068933	FRAGUAB.MX	0.033399
IAU.MX	0.048970	APA.MX	0.024810
DELLC.MX	0.069660	ALSEA.MX	0.066133

Tabla 1: *Portafolio de inversión generado por el algoritmo evolutivo. Cada valor representa la proporción del capital total que se ha de invertir en su respectivo activo.*

4. Conclusiones

El algoritmo propuesto ha demostrado en todas las pruebas realizadas ser flexible y robusto al adaptarse a distintos escenarios y mercados financieros (generados artificialmente a partir de datos históricos reales). Más aún, logró superar el benchmark de VMEX en repetidas ocasiones en un tiempo bastante reducido dado por la evolución de 50 generaciones (al rededor de 5 minutos).

Es, además, adaptable a las preferencias del usuario, pues mostró ser capaz de priorizar el aumento del rendimiento esperado o la disminución del riesgo según el valor del parámetro δ. Finalmente, la diversificación del portafolio se puede ajustar de manera simple al fijar el valor de w_{max}.

A pesar de los logros obtenidos, la elección de la mayoría de los parámetros se realizó mediante criterios heurísticos o mediante pruebas experimentales. Estos experimentos se llevaron a cabo para ajustar distintos parámetros uno por uno. Sin embargo, es bien sabido que los distintos componentes y parámetros de un algoritmo evolutivo interaccionan de maneras altamente no lineales, por lo que en un trabajo futuro se podrían diseñar experimentos o mecanismos con los cuales se pudieran determinar buenos valores para los parámetros de manera simultánea. En nuestro caso, por limitaciones de tiempo y de recursos computacionales no fue posible realizar dichas búsquedas, a pesar de lo cual se obtuvieron resultados satisfactorios respecto a los objetivos planteados inicialmente.

Referencias

[BANXICO, sf] BANXICO (s.f.). Valores gubernamentales. Recuperado de `https://www.banxico.org.mx/SieInternet/consultarDirectorioInternetAction.do?accion=consultarCuadro&idCuadro=CF107&sector=22&locale=es`.

[Bodie et al., 2018] Bodie, Z., Kane, A., and Marcus, A. J. (2018). Risk, return, and the historical record. In *Investments*, chapter 5, pages 117–156. McGraw-Hill Education.

[Brealey et al., 2011] Brealey, Myers, and Allen (2011). Portfolio theory and the capital asset pricing model. In *Principlesf of Corporate Finance*, chapter 8, pages 185–212. McGraw-Hill/Irwin.

[Bäck and Schütz, 1996] Bäck, T. and Schütz, M. (1996). Intelligent mutation rate control in canonical genetic algorithms. *Foundations of Intelligent Systems*, 1079. https://doi.org/10.1007/3-540-61286-$6_1$41.

[Cetesdirecto, sf] Cetesdirecto (s.f.). ¿qué es cetes? Recuperado de `https://www.cetesdirecto.com/sites/portal/inicio#queEsCetes`.

[de Trending, sf] de Trending, E. (s.f.). Notas técnicas sobre el cálculo de la volatilidad. Recuperado de `https://estrategiastrading.com/calcular-volatilidad/`.

[Eiben et al., 2006] Eiben, A. E., Schut, M. C., and de Wilde, A. R. (2006). Boosting genetic algorithms with self-adaptive selection. *International Conference on Evolutionary Computation*, 1079:477–482.

[Eiben and Smith, 2015] Eiben, A. E. and Smith, J. E. (2015). *Introduction to evolutionary computing*. Springer.

[Grupo BMV, sfa] Grupo BMV (s.f.a). Calendario de días festivos. Recuperado de `https://www.bmv.com.mx/es/Grupo_BMV/Calendario_de_dias_festivos/_rid/662/_mod/TAB_HORARIOS_NEG`.

[Grupo BMV, sfb] Grupo BMV (s.f.b). Mercado global. Recuperado de `https://www.bmv.com.mx/es/mercados/mercado-global`.

[Markowitz, 1952] Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, 7(1):77–91. https://doi.org/10.1111/j.1540-6261.1952.tb01525.x.

[Vanguard Mexico, sf] Vanguard Mexico (s.f). Productos. Recuperado de https://www.vanguardmexico.com/es/productos/productos-financieros/etf-de-renta-variable/VMEX.

5. Apéndices

5.1. Apéndice A: Algoritmo Evolutivo

El algoritmo utilizado se programó en forma de clase, dónde a cada operador le corresponde un método distinto, y varios mecanismos fueron implementados para facilitar el proceso de ajuste y evaluación. El parámetro `testing` se añadió para decidir si se desea generar un historial de valores de aptitud, de riesgos y retornos esperados, etc, o si se desea utilizar el algoritmo únicamente para obtener un portafolio de inversiones (en este caso se debería utilizar `testing=False`).

Además, está programado de forma que se pueda retomar el proceso evolutivo desde el punto en el que se quedó si se observa que la tendencia del fitness sigue siendo creciente. Esto permite ejecutar el algoritmo por un número pequeño de generaciones, y tomar decisiones según el comportamiento de las curvas resultantes.

```
class EA:
    """Evolutionary Algorithm for investment portfolio optimization."""

    def __init__(self, data, pop_size=100, lambda_=600, p_m=0.01, sigma=0.1,
                 max_w=0.1, delta=0.5, alpha=0.5, testing=True):
        """
        Initialize the EA with the given parameters and data.

        Args:
        - data (pd.DataFrame): Historical closing prices.
        - pop_size (int, optional): Population size for the EA. (default 100)
        - lambda_ (int, optional): Number of offspring per generation.
          (default 600)
        - p_m (float, optional): Mutation rate. (default 0.01)
        - sigma (float, optional): Mutation step size. (default 0.1)
        - max_w (float, optional): Maximum weight per asset. (default 0.1)
        - delta (float, optional): Fitness weight constant. (default 0.5)
        - alpha (float, optional): Blend crossover constant. (default 0.5)
        """
        self.stats_(data) # Expected anual returns returns and covariance matrix
        self.pop_size = pop_size # Population size
        self.lambda_ = lambda_ # Number of offspring per generation
        self.p_m = p_m # Mutation rate
        self.sigma = sigma # Mutation step size
        self.delta = delta # Fitness weight constant
        self.alpha=0.5
        self.max_w = max_w # Max weight per asset
        self.assets = data.columns.values # Asset names
        self.n_assets = data.shape[1] # Number of assets
        self.population = self.initialize_population_()
        self.fitness_evaluations = 0
        self.pop_fitness = self.get_fitness(self.population[:,:-1])

        # For plotting and evaluating model performance and robustness.
        self.testing = testing
        if self.testing:
            self.diversity = len(np.unique(self.population, axis=0))
            self.diversity_history = [self.diversity]
            self.max_fit_history = [self.pop_fitness.mean()]
            self.mean_fit_history = [self.pop_fitness.max()]
```

```python
            self.best_fitness = self.pop_fitness.max()

    def stats_(self, data):
        """
        Compute and store expected annual returns and covariance matrix.

        Args:
        - data (pd.DataFrame): Historical asset prices.
        """
        # Daily closing price % change.
        daily_change = data.pct_change()
        # Expected anual return.
        self.expected_returns = (daily_change.mean()*252).values
        # Covariance matrix.
        self.cov_matrix = daily_change.cov().to_numpy()

    def volatility(self, chromosome):
        """
        Calculate the volatility (risk) of the portfolio.

        Args:
        - chromosome (np.ndarray): Portfolio weights.

        Returns:
        - float: Portfolio volatility.
        """
        return np.sqrt(252*chromosome.T @ self.cov_matrix @ chromosome)

    def weight_penalty_(self, chromosome, w=20):
        """
        Compute penalization for weights exceeding the maximum allowed.

        Args:
        - chromosome (np.ndarray): Portfolio weights.
        - w (float, optional): Penalty weight.

        Returns:
        - float: Penalty value.
        """
        penalization = 0
        for gene in chromosome[:-1]:
            if gene > self.max_w: # If too high...
                penalization += gene - self.max_w # penalize proportionally.
        return w*penalization

```

```python
    def fitness(self, chromosome):
        """
        Calculate the fitness of the portfolio.

        Args:
        - chromosome (np.ndarray): Portfolio weights.

        Returns:
        - tuple: Fitness value (f), expected returns for the portfolio (f1),
        and portfolio volatility (f2).
        """
        self.fitness_evaluations += 1 # For performance evaluation (AES, SR)
        f1 = chromosome @ self.expected_returns # Expected portfolio returns.
        f2 = self.volatility(chromosome) # The risk of the portfolio.
        f = (1 - self.delta)*f1 - self.delta * f2 # Fitness.
        if np.any(chromosome > self.max_w): # weight penalization
            f -= self.weight_penalty_(chromosome)
        return f, f1, f2

    def get_fitness(self, group):
        """
        Calculate fitness values for a group of portfolios.

        Args:
        - group (np.ndarray): Group of portfolios.

        Returns:
        - np.ndarray: Fitness values.
        """
        return np.array([self.fitness(x)[0] for x in group])

    def normalize(self, chromosome):
        """
        Normalize a chromosome so that its genes sum up to 1.

        Args:
        - chromosome (np.ndarray): Portfolio weights.

        Returns:
        - np.ndarray: Normalized portfolio weights.
        """
        for i, gene in enumerate(chromosome):
            if gene < 0: # Portfolio cannot have negative weights.
                chromosome[i] = 0
        return chromosome/chromosome.sum()
```

```python

    def initialize_population_(self):
        """
        Initialize the population satisfying problem constraints.

        Returns:
        - np.ndarray: Initial population.
        """
        population = np.empty([self.pop_size, self.n_assets + 1])
        for k in range(self.pop_size):
            while True: # Generate until valid portfolio is found.
                chromosome = np.random.uniform(0, self.max_w, self.n_assets)
                chromosome = self.normalize(chromosome)
                good_weights = np.all(chromosome <= self.max_w)
                if good_weights:
                    break
            ki = np.random.uniform(0,1) # Tournament size self-adaptive gene.
            chromosome = np.append(chromosome, ki) # Add self-adaptive gene.
            population[k] = chromosome # Add chromosome to population.
        return population

    def parent_selection(self):
        """
        Perform uniform parent selection.

        Returns:
        - np.ndarray: Selected parent population.
        """
        # Selection is performed WITH replacement.
        selected = np.random.randint(0, self.pop_size, self.lambda_)
        return self.population[selected]

    def recombination(self, p1, p2):
        """
        Perform blend crossover (BLX-alpha) recombination.

        Args:
        - p1 (np.ndarray): First parent chromosome.
        - p2 (np.ndarray): Second parent chromosome.

        Returns:
        - tuple: Two child chromosomes.
        """
        u = np.random.uniform()
        gamma = (1 - 2*self.alpha) * u - self.alpha
```

```
        child1 = (1 - gamma) * p1 + gamma * p2
        child2 = (1 - gamma) * p2 + gamma * p1
        return child1, child2

    def mutation(self, chromosome, eta=0.22):
        """
        Perform Gaussian perturbation mutation.

        Args:
        - chromosome (np.ndarray): Input chromosome.
        - eta (float, optional): Tournament size mutation parameter
          (default is 0.22).

        Returns:
        - np.ndarray: Mutated chromosome.
        """
        ### Mutation for portfolio weights.
        for i, gene in enumerate(chromosome[:-1]):
            if np.random.random() < self.p_m: # P(mutation) = p_m per gene.
                chromosome[i] += np.random.normal(scale=self.sigma)
        x = self.normalize(chromosome[:-1])

        # Mutation for (self-adaptive) tournament size gene.
        k = chromosome[-1]
        if k < 0 or k >= 1: # k has to be in [0, 1].
          k = np.random.uniform(0, 1)
        new_k = ( 1 + ( (1 - k) / k ) * np.exp(-eta*np.random.normal()) )**(-1)

        return np.append(x, new_k)

    def survival_selection(self, mutated, mutated_fitness, k):
        """
        Perform tournament replacement for survival selection.

        Args:
        - mutated (np.ndarray): Mutated population.
        - mutated_fitness (np.ndarray): Fitness values of mutated population.
        - k (int): Tournament size.

        Returns:
        - tuple: New generation and fitness values.
        """
        new_generation = np.empty([self.pop_size, self.n_assets+1])
        new_fitness = np.empty(self.pop_size)
        for i in range(self.pop_size):
```

```
            # Select k contestants.
            indices = np.random.choice(self.lambda_, k, replace=False)
            tourn = mutated[indices] # Their chromosomes.
            fit_tourn = mutated_fitness[indices] # And their fitness.
            win = tourn[np.argmax(fit_tourn)] # Highest fitness wins.
            win_fit = fit_tourn.max()
            new_generation[i] = win
            new_fitness[i] = win_fit
        return new_generation, new_fitness


    def run(self, iters):
        """
        Run the EA for a specified number of generations.

        Args:
        - iters (int): Number of generations to run.
        """
        for _ in range(iters):
            parents = self.parent_selection() # Select parents.
            offspring = np.empty([self.lambda_, self.n_assets+1])
            mutated = np.empty(offspring.shape)

            ### Produce offspring.
            for i in range(0, self.lambda_, 2):
                offspring[i], offspring[i+1] = self.recombination(parents[i],
                                                                  parents[i+1])
                mutated[i], mutated[i+1] = self.mutation(offspring[i]), \
                                           self.mutation(offspring[i+1])
            mutated_fit = self.get_fitness(mutated[:,:-1]) # Get their fitness.

            ### Update population.
            k = int(np.ceil(np.sum(mutated[:, -1]))) # Tournament size.
            self.population, self.pop_fitness = self.survival_selection(mutated,
                                                                 mutated_fit, k)

            ### Update and store historical attributes.
            if self.testing:
                self.best_fitness = self.pop_fitness.max()
                self.diversity = len(np.unique(self.population[:,:-1], axis=0))
                self.diversity_history.append(self.diversity)
                self.max_fit_history.append(self.pop_fitness.max())
                self.mean_fit_history.append(self.pop_fitness.mean())

    def plot_diversity(self, label=None):
```

```
        """
        Plot the evolution of population diversity.
        """
        plt.plot(self.diversity_history, label=label)
        plt.title('Population Diversity')
        plt.ylabel('# of unique individuals')
        plt.xlabel('Generation')

    def plot_fitness(self):
        """
        Plot the evolution of mean and max fitness.
        """
        plt.plot(self.mean_fit_history, label='mean population fitness', lw=3)
        plt.plot(self.max_fit_history, '--', label='max population fitness',
                                                                         lw=2)
        plt.legend()
        plt.title('Mean and Max Fitness Evolution')
        plt.ylabel('Fitness')
        plt.xlabel('Generation')

    def plot_max_fitness(self, label=None):
        """
        Plot the evolution of max fitness.
        """
        plt.plot(self.max_fit_history, lw=2, label=label)
        plt.title('Max Fitness Evolution')
        plt.ylabel('Fitness')
        plt.xlabel('Generation')

    def portfolio(self):
        best_portfolio = self.population[np.argmax(self.pop_fitness)][:-1]
        not_null = best_portfolio > 0
        df = pd.DataFrame(best_portfolio[not_null], self.assets[not_null]).T
        return df
```

5.2. Apéndice B: Construcción del DataSet

La siguiente función se utilizó para descargar y preparar los datos históricos de los activos para su uso por el algoritmo.

```
def get_historical_data(tickers, start, end):
    """
    Downloads and processes historical data from Yahoo Finance.
```

```
    Args:
    tickers (list): List of asset symbols.
    start (str): Start date in 'YYYY-MM-DD' format.
    end (str): End date in 'YYYY-MM-DD' format.
    Returns:
    pd.DataFrame: DataFrame with adjusted close prices per asset.
    """

    # Download data from Yahoo Finance
    data = yf.download(tickers, start=start, end=end)['Adj Close']

    # Handle cases with a single asset
    if not isinstance(data, pd.DataFrame):
        data = pd.DataFrame(data)
        data.columns = [tickers[0]]

    # Remove assets with more than 10 NaN values
    data = data.dropna(axis=1, thresh=data.shape[0]-10)

    # Fill NaN values with previous or next value
    data = data.fillna(method='ffill').fillna(method='bfill')

    return data

# Mount Google Drive
drive.mount('/content/drive')

# Time range to consider in the historical data
start = '2021-01-01'
end = '2024-04-05'

## National capitals
# Load the CSV file with the names of the national assets
assets_NC = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Proyecto_AE/ \
CapitalesNacionales.csv')

# Extract the names
tickers_NC = assets_NC['Tickers'].tolist()

# Get historical data for national capitals
closing_prices_NC = get_historical_data(tickers_NC, start, end)

## FIBRAS
# Load the CSV file with the names of the FIBRAS
assets_F = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Proyecto_AE/FIBRAS.csv')
```

```

# Extract the names
tickers_F = assets_F['Tickers'].tolist()

# Get historical data for FIBRAS
closing_prices_FIBRAS = get_historical_data(tickers_F, start, end)

## International capitals
# Load the CSV file with the names of the international assets
assets_IC = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Proyecto_AE/ \
CapitalesExtranjeros.csv')

# Extract the names
tickers_IC = assets_IC['Tickers'].tolist()

# Get historical data for international capitals
closing_prices_IC = get_historical_data(tickers_IC, start, end)

## Bonddia
# Get historical data for Bonddia
closing_prices_BONDDIA = get_historical_data('BONDDIAPF2.MX', start, end)

## ENERFIN
# Get historical data for ENERFIN
closing_prices_ENERFIN = get_historical_data('ENERFINPF2.MX', start, end)

## CETES
# Set the same number of closing prices for CETES (compared to other assets)
num_days = len(closing_prices_ENERFIN)

# Initialize with 1
values = [1]

for i in range(1, num_days):
    previous_value = values[i-1]
    new_value = previous_value * ((.1103/252) + 1)
    values.append(new_value)

# Create DataFrame for CETES
closing_prices_CETES = pd.DataFrame(values, columns=['CETES365'])

closing_prices_CETES.set_index(closing_prices_ENERFIN.index, inplace=True)

## Combine closing price DataFrames
closing_prices = pd.concat([
```

```
    closing_prices_NC,
    closing_prices_FIBRAS,
    closing_prices_IC,
    closing_prices_BONDDIA,
    closing_prices_ENERFIN,
    closing_prices_CETES
], axis=1)
```

5.3. Apéndice C: Medidas de Desempeño

Se definieron funciones para evaluar de maneras diferentes el desempeño del algoritmo evolutivo.

5.3.1. MBF

El MBF (Mean Best Fitness) fue implementado de modo que regresara no sólo el fitness promedio, sino también los retornos esperados y sus riesgos, ya que estas cantidades son más fáciles de interpretar en el contexto del problema.

```
def MBF(algorithm, data, runs=10, generations=50, **kwargs):
"""
Calculate and print the Mean Best Fitness (MBF) over multiple runs.

Args:
- algorithm (class): Evolutionary algorithm class.
- data (pd.DataFrame): Historical closing prices.
- runs (int, optional): Number of runs (default is 10).
- generations (int, optional): Num. of generations per run (default is 50).
- **kwargs: Additional keyword arguments for the algorithm.

Returns:
- best_fitness (np.ndarray): Array of best fitness values from each run.
- returns (np.ndarray): Array of best expected returns obtained on the last
  generation of each run.
- risks (np.ndarray): Array with the risks associated to the best individual
  if the las generation for each run.
"""
best_fitness = np.empty(runs)
returns = np.empty(runs)
risks = np.empty(runs)
for i in range(runs):
    ea = algorithm(data, **kwargs)
    ea.run(generations)
```

```
            best = ea.population[np.argmax(ea.pop_fitness)][:-1]
            f, r, risk = ea.fitness(best)
            best_fitness[i] = f
            returns[i] = r
            risks[i] = risk
    print('MBF: ', best_fitness.mean())
    return best_fitness, returns, risk
```

5.3.2. AES y SR

El AES (Average Evaluations to Success) se implementaron de modo que se calcularan de manera simultánea. Igual que para el MBF, se regresa también el historial de retornos esperados y riesgos para una mejor interpretación.

```
def AES_SR(algorithm, data, solution=0.26, max_risk=0.18,
           runs=10, max_gens=50, **kwargs):
    """
    Calculate the Average Fitness Evaluations to Solution (AES) and the Success
    Rate (SR) over a series of runs.

    Args:
    - algorithm (class): Evolutionary algorithm class.
    - data (pd.DataFrame): Historical closing prices.
    - solution (float, optional): Target solution value (default is 0.26).
    - runs (int, optional): Number of runs (default is 10).
    - max_gens (int, optional): Maximum number of generations to run. If the
      solution is not found within this number of generations, that run's
      evaluations are not included in the calculation of AES. (default is 100).
    - **kwargs: Additional keyword arguments for the algorithm.

    Returns:
    - evaluations (np.ndarray): Array of fitness evaluations required to reach
      the solution in each run. Runs where no solution was found have a nan value.
    - returns (np.ndarray): Array of best expected returns obtained on the last
      generation of each run.
    - risks (np.ndarray): Array with the risks associated to the best individual
      if the las generation for each run.
    """
    evaluations = np.empty(runs)
    returns = np.empty(runs)
    risks = np.empty(runs)
    fails = 0
    for i in range(runs):
```

```python
        ea = algorithm(data, **kwargs)
        gen = 0
        best = ea.population[np.argmax(ea.pop_fitness)][:-1]
        _, r, risk = ea.fitness(best)
        while r < solution or risk > max_risk: # while no solution is found
            ea.run(1)
            best = ea.population[np.argmax(ea.pop_fitness)][:-1]
            _, r, risk = ea.fitness(best)
            gen += 1
            if gen == max_gens: # If no solution found in max_gens generations
                ea.fitness_evaluations = np.nan # don't count this run.
                fails += 1 # Keep trak of failures of success.
                break
        returns[i] = r
        risks[i] = risk
        evaluations[i] = ea.fitness_evaluations
    successes = runs - fails
    print('AES: ', np.nanmean(evaluations))
    print('SR: ', successes/runs)
    return evaluations, returns, risks
```

5.4. Apéndice D: Medidas de Robustez

Se optó por medir la robustez del algoritmo de dos maneras diferentes.

5.4.1. Respecto al Conjunto de Activos

En este caso cada corrida utiliza un conjunto distinto de activos.

```python
    def asset_robustness(algorithm=EA, data=all_data, sample_size=100,
                         runs=5, generations=50, **kw):
    """
    Helps evaluate the robustness of the evolutionary algorithm to different
    groups of assets, i.e. to different problem instances.

    Args:
    - algorithm (class, optional): Evolutionary algorithm class (default is EA).
    - data (pd.DataFrame, optional): All asset closing data
      (default is all_data).
    - frac (float, optional): The percentage of all assets to use in each run
      (default is 0.1).
    - runs (int, optional): Number of runs (default is 5).
```

```python
    - generations (int, optional): Number of generations per run
      (default is 100).
    - **kw: Additional keyword arguments for the algorithm.

    Returns:
    - np.ndarray: Array of best fitness values from each run.
    """
    fits = np.empty(runs)
    for i in range(runs):
        print('run: ', i+1)
        data_i = data.sample(sample_size, axis=1)
        ea = algorithm(data_i, **kw)
        ea.run(generations)
        ea.plot_max_fitness()
        fits[i] = ea.best_fitness
    return fits
```

5.4.2. Respecto al Periodo de Tiempo

En este caso se toman periodos de tiempo diferentes, y en cada caso se eligen los mejores 100 activos para considerar en el portafolio.

```python
def time_robustness(algorithm=EA, data=all_data, periods=5,
                    generations=50, n_assets=100, **kw):
    """
    Helps evaluate the robustness of the evolutionary algorithm to different
    time periods, given a fixed group of assets. This situation resembles a
    real-life situation, where every day the data is slightly different from the
    previous one and, over a long period of time, it can vary significantly.

    Args:
    - algorithm (class, optional): Evolutionary algorithm class (default is EA).
    - data (pd.DataFrame, optional): All asset closing data
      (default is all_data).
    - periods (int, optional): The number of time periods the data will be slit
      on (default is 5).
    - generations (int, optional): Number of generations per run
      (default is 100).
    - **kw: Additional keyword arguments for the algorithm.

    Returns:
    - np.ndarray: Array of best fitness values from each run.
    """
```

```
    fits = np.empty(periods)
    data = np.array_split(data, periods)
    for i, period_data in enumerate(data):
        period = rank_data(period_data, n_assets)
        print(f'period #{i+1}')
        start = str(period.index[0])
        end = str(period.index[-1])
        print('start: ', start)
        print('end: ', end)
        ea = algorithm(period, **kw)
        ea.run(generations)
        ea.plot_max_fitness(label=f'{start[5:7]}/{start[:4]}-{end[5:7]}/{end[:4]}')
        fits[i] = ea.best_fitness
        print('best fitness: ', fits[i])
        print('------------------------')
    plt.legend(bbox_to_anchor=(1.0, 1.02))
    return fits
```