Practica 01

Análisis Comparativo: Simulated Annealing vs Hill Climbing en la Optimización de Portafolios Financieros

Alberto Valentín Velásquez Santos[†], **Rodolfo Morocho Caballero**[†], **Max Houston Ramirez Martel**[†] **and Harold Mondragon Tavara**[†]

[†]*Estos autores contribuyeron igualmente a este trabajo.*

Este archivo fue compilado el 26 de Octubre del 2024

Abstract

Este trabajo busca analizar y comparar la efectividad de dos algoritmos de optimización – Simulated Annealing y Hill Climbing – en el contexto de la optimización de portafolios financieros. Se utilizará el Sharpe Ratio como función fitness para medir el rendimiento ajustado al riesgo de los portafolios generados por cada algoritmo. El objetivo es identificar las ventajas y limitaciones de ambos métodos en términos de su capacidad para maximizar el rendimiento ajustado al riesgo en un conjunto de datos de activos financieros. Comparar Simulated Annealing y Hill Climbing en el contexto de la optimización de portafolios resulta relevante, ya que ambos algoritmos ofrecen enfoques distintos para la búsqueda de soluciones óptimas. Simulated Annealing explora el espacio de soluciones con una mayor capacidad para evitar los óptimos locales debido a su proceso de "enfriamiento", lo que podría ofrecer soluciones potencialmente superiores para problemas complejos como la optimización de portafolios. Por otro lado, Hill Climbing es conocido por su mayor velocidad, aunque puede quedar atrapado en óptimos locales, lo cual es una desventaja en contextos de alta variabilidad y múltiples picos en la función objetivo. La comparación de estos dos enfoques proporcionará una perspectiva sobre la eficiencia y aplicabilidad de cada algoritmo en la gestión de portafolios ajustada al riesgo.

Keywords: *optimización, heurísticas, Simulated Annealing, Hill Climbing , Machine Learning, Markowitz*

1. Marco Teórico

La La optimización de portafolios es una práctica central en las finanzas, enfocada en maximizar el rendimiento ajustado al riesgo de una cartera para un nivel de riesgo dado, o en minimizar el riesgo para un rendimiento específico. Este proceso se fundamenta en los principios de diversificación y equilibrio entre riesgo y retorno, tal como se describe en la teoría de portafolios de Markowitz (1952), que sentó las bases para el análisis moderno de inversiones. En este modelo, la minimización de la varianza de los retornos se combina con la maximización de su media para construir una cartera eficiente, definiendo el riesgo como la volatilidad de los retornos de una cartera y permitiendo que los inversores tomen decisiones informadas en función de su tolerancia al riesgo. Complementando la teoría de Markowitz, el Sharpe Ratio, introducido por William F. Sharpe en 1966, evalúa la eficiencia de una cartera ajustando su rendimiento al riesgo. Este ratio es calculado dividiendo el exceso de retorno de la cartera sobre la tasa libre de riesgo por la desviación estándar de sus retornos. De esta manera, el Sharpe Ratio permite una comparación directa entre carteras al normalizar los retornos en términos del riesgo asumido, consolidándose como una herramienta fundamental para la optimización de portafolios en el ámbito financiero, especialmente cuando se busca maximizar la recompensa obtenida por cada unidad de riesgo. La optimización de portafolios implica complejidades adicionales cuando se introducen restricciones no lineales, como los costos de transacción fijos o los límites en la inversión en unidades discretas. Estas restricciones convierten el problema en uno no convexo, limitando la efectividad de los métodos clásicos como la programación cuadrática y lineal. En este contexto, los algoritmos heurísticos como Hill Climbing (HC) y Simulated Annealing (SA) han surgido como alternativas viables para enfrentar la naturaleza de estas funciones objetivo. HC es un algoritmo de búsqueda local que optimiza la función objetivo mediante ajustes incrementales a una solución inicial. Aunque es rápido en converger hacia un óptimo local, su enfoque determinista lo hace vulnerable a quedar atrapado en estos óptimos, lo que puede ser un obstáculo significativo en problemas complejos de optimización de portafolios, donde las múltiples restricciones generan una superficie de solución con múltiples picos y valles. Simulated Annealing (SA), que permite una exploración más amplia del espacio de soluciones al aceptar soluciones subóptimas en las primeras etapas del proceso. Este enfoque de exploración adaptativa evita que el algoritmo se quede atrapado en óptimos locales, permitiendo una búsqueda más robusta de soluciones óptimas globales, especialmente en problemas de optimización no convexos, como aquellos que incluyen costos fijos de transacción. En el estudio de Rubio-García, García-Ripoll y Porras (2022), se evidencia que SA es particularmente adecuado para la optimización de portafolios en presencia de restricciones complejas. Su estructura estocástica y adaptativa permite reducir el tiempo de búsqueda al explorar eficientemente soluciones subóptimas al inicio, lo que contribuye a la eficacia de este algoritmo en problemas de portafolios discretos. Además, el estudio introduce una métrica de rendimiento denominada "time to target" (tiempo para alcanzar el objetivo), que cuantifica el tiempo requerido para obtener una solución de calidad en términos de una solución que maximiza el Sharpe Ratio, proporcionando una referencia para comparar la eficiencia de SA frente a otros métodos. En la comparación entre Hill Climbing y Simulated Annealing, este último muestra una capacidad superior para maximizar el Sharpe Ratio al ser más resistente a los óptimos locales. Esto es esencial en escenarios financieros donde la complejidad de las restricciones y los costos fijos de transacción agregan un nivel adicional de dificultad. Hill Climbing, si bien eficiente en tiempo, resulta limitado en su exploración y podría no encontrar la solución óptima global en problemas de portafolios que incluyen estos componentes no convexos. En este sentido, el uso de SA ofrece una exploración del espacio de soluciones más completa, mientras que Hill Climbing es preferible en situaciones donde se prioriza la rapidez de convergencia sobre una búsqueda exhaustiva.

Sharpe Ratio

$$\text{Sharpe Ratio} = \frac{R_x - R_f}{\sigma_{R_x}}$$

Donde:

R_x = Expected portfolio return
R_f = Risk free rate of return
σ_{R_x} = Standard deviation of portfolio return / volatility

2. Metodología fitness restricciones

Para evaluar el desempeño de Simulated Annealing y Hill Climbing en la optimización de portafolios, este estudio emplea el Sharpe Ratio como función fitness. El Sharpe Ratio, definido anteriormente como la diferencia entre el rendimiento esperado del portafolio y la tasa libre de riesgo, dividido por la desviación estándar de los rendimientos, es un indicador clave en la gestión de inversiones, pues permite medir la eficiencia de una cartera en términos de retorno ajustado al riesgo. Utilizar el Sharpe Ratio como objetivo de optimización permite que ambos algoritmos ajusten la composición de los activos para maximizar esta medida, evaluando la capacidad de cada método para alcanzar soluciones eficaces en función del riesgo asumido. En cuanto a la configuración de los algoritmos, Simulated Annealing se ha ajustado con una temperatura inicial alta (x definir) y una tasa de enfriamiento controlada, factores que facilitan la exploración del espacio de soluciones y evitan caer en óptimos locales en las primeras iteraciones. A medida que la temperatura disminuye, el algoritmo se vuelve más. conservador, favoreciendo soluciones cada vez más óptimas y menos dispuestas a aceptar cambios que no mejoren el Sharpe Ratio. Para Hill Climbing, el criterio de terminación está definido por la la ausencia de mejoras significativas en iteraciones sucesivas. La naturaleza de Hill Climbing lo lleva a realizar cambios incrementales en la composición del portafolio, en busca de maximizar el Sharpe Ratio de manera rápida pero sin considerar saltos que podrían llevar a un mejor óptimo global, lo cual contrasta con el enfoque de SA. Los datos utilizados en este análisis incluyen rendimientos históricos y desviaciones estándar de varios activos (simulados random), permitiendo la simulación de un portafolio diversificado. Este conjunto de datos se elige para representar un horizonte de tiempo específico, capturando la volatilidad y los retornos esperados de cada activo. Los datos se alimentan a ambos algoritmos, lo que permite una comparación homogénea y una composición de activos a elegir iguales con la finalidad de que busquen maximizar el Sharpe Ratio. La implementación de estos algoritmos se realizó en Python, con diferencias estructurales en el código para adaptarse a las particularidades de cada algoritmo. En el caso de Simulated Annealing, se configuraron funciones para el enfriamiento gradual y la aceptación de soluciones subóptimas en etapas iniciales, mientras que en Hill Climbing se estableció un enfoque de búsqueda directa, permitiendo modificaciones sólo cuando mejoraban el Sharpe Ratio. Este diseño permite una comparación justa y precisa del rendimiento de ambos métodos en condiciones similares.

2.1. Fitness

El fitness es una medida que evalúa la calidad de cada solución, es decir, de cada combinación de pesos asignados a los activos en el portafolio con su respectivo retorno esperado ajustado al riesgo que esta conlleva. Es importante mencionar que la función objetivo de nuestro problema es el Sharpe Ratio mencionado anteriormente y se busca maximizar esta métrica.La interpretacion es cuanto mayor sea el Sharpe Ratio, mayor será el retorno ajustado al riesgo, y por lo tanto, mejor será la solución en términos de optimización del portafolio. Por cuestiones de practicidad y para lograr tener una comparación de portafolios que lleguen hasta n=500 se optó por generar valores aleatorios controlando el rango que estos pueden optar, asignamos valores que en el mundo financiero son los más comunes. El retorno esperado y la definición del riesgo se presentan a continuación:

```
# Retornos esperados de cada activo
returns = np.random.uniform(0.1, 0.2, n)

# Matriz de covarianza
cov_matrix = np.random.uniform(0.001, 0.002, (n, n))

```

Code 1. Generación de retornos y matriz de covarianza

A continuación, se detalla el proceso para calcular el Sharpe Ratio en el código:

1. Cálculo del retorno del portafolio: Dado un vector de pesos que define la proporción invertida en cada activo (Porcentaje dentro del portafolio), el retorno esperado del portafolio se calcula como el producto escalar entre estos pesos y los retornos esperados individuales de los activos:

```
portfolio_return = np.dot(weights, returns)

```

Code 2. Cálculo del retorno del portafolio

   Esto representa el retorno ponderado del portafolio, calculado en función de la proporción de inversión en cada activo.
2. Cálculo de la varianza del portafolio: Para calcular el riesgo del portafolio, se utiliza la matriz de covarianza de los retornos de los activos, **cov_matrix**. La varianza del portafolio se obtiene multiplicando la matriz de covarianza por el vector de pesos, y luego realizando un segundo producto escalar con el mismo vector de pesos:

$$\sigma_p^2 = \mathbf{w}^T \cdot \mathbf{cov\_matrix} \cdot \mathbf{w} \tag{1}$$

```
portfolio_variance = np.dot(weights, np.dot(
 cov_matrix, weights))

```

Code 3. Cálculo de la varianza del portafolio

   La varianza es una medida del riesgo total del portafolio. La raíz cuadrada de esta varianza nos da la desviación estándar del portafolio, que representa el riesgo asumido al invertir en esta combinación de activos.
3. Cálculo del Sharpe Ratio (fitness): Finalmente, el Sharpe Ratio se calcula dividiendo el retorno del portafolio entre la desviación estándar del mismo:

```
harpe_ratio = portfolio_return / np.sqrt(
 portfolio_variance)

```

Code 4. Cálculo de la varianza del portafolio

   Este Sharpe Ratio es el valor de la función de fitness. Maximizar este valor es el objetivo del problema, pues refleja una mejor relación entre el rendimiento obtenido y el riesgo asumido.

2.2. Definición de la Solución

La solución en el contexto de este estudio es un vector de pesos que define la proporción de inversión asignada a cada activo en el portafolio. En el código, esta solución se inicializa mediante un vector aleatorio generado con la función np.random.dirichlet, que asegura que los valores asignados a cada activo sumen 1. Esta condición es fundamental, ya que en un portafolio realista, todos los activos combinados representan el 100% de la inversión. En Python, el vector inicial de pesos se obtiene de la siguiente manera:

```
weights = np.random.dirichlet(np.ones(n), size=1)[0]

```

Code 5. vector inicial

donde n es el número de activos en el portafolio. Este vector weights es, por tanto, la representación de la solución inicial para ambos algoritmos.

Ser “solución” en este problema implica cumplir con ciertas propiedades que hacen que una combinación de pesos sea válida y útil en la optimización. En primer lugar, debe asegurar la **composición total de la inversión**, de modo que los pesos en el vector sumen 1, garantizando así que el capital se distribuye completamente entre los activos, sin exceder ni dejar por fuera parte del total invertido; esto refleja la realidad de una cartera en la que el total de los recursos está completamente asignado.

Además, cada solución representa una **configuración específica de riesgo y rendimiento esperados**; el objetivo es encontrar aquella combinación de pesos que maximice el Sharpe Ratio, el cual mide el retorno ajustado al riesgo. De este modo, una "buena" solución es aquella que logra un equilibrio adecuado entre el riesgo y el rendimiento esperado, maximizando el Sharpe Ratio.

Asimismo, la **adaptabilidad a diferentes combinaciones de activos** es fundamental, pues la solución debe ajustarse a distintas configuraciones, ya que los pesos pueden variar para adaptarse a los riesgos y retornos específicos de cada activo, lo cual hace que la solución sea flexible y se adapte a distintas situaciones y estrategias de inversión.

La combinación específica de pesos es considerada la solución óptima porque optimiza el objetivo del problema, es decir, maximiza el Sharpe Ratio, logrando así el mayor valor posible dentro del conjunto de soluciones evaluadas por el algoritmo. Finalmente, esta solución también **resuelve el problema bajo restricciones**, cumpliendo con la restricción fundamental de que los pesos sumen 1, respetando la realidad de que el portafolio debe estar completamente invertido.

2.3. Vecinos

Para explorar el espacio de soluciones y mejorar la asignación de activos, ambos algoritmos (Hill Climbing y Simulated Annealing) generan **vecinos** de la solución actual. Los vecinos son nuevas configuraciones de pesos que se obtienen realizando pequeñas variaciones en la combinación actual, lo cual permite explorar alternativas cercanas para encontrar mejores soluciones.

2.3.1. Generación de Vecinos en Hill Climbing

En Hill Climbing, el proceso de generar un vecino se basa en realizar ajustes incrementales en los pesos de los activos, manteniendo la suma total en 1. El algoritmo modifica ligeramente un peso aleatorio en el vector de pesos, aumentando o disminuyendo su valor en una pequeña cantidad aleatoria. Esto permite encontrar un vecino cercano a la solución actual:

```
    def get_neighbors(weights):
        idx = np.random.randint(len(weights))
        change = np.random.uniform(-0.1, 0.1)
        new_weights = weights.copy()
        # Aplica el cambio
        new_weights[idx] = np.clip(new_weights[idx] + change, 0,
     1)
        # Normaliza para que sumen 1
        new_weights /= np.sum(new_weights)
        return new_weights
```

Code 6. Función para generar vecinos en Hill Climbing

Esta función asegura que el nuevo vector de pesos representa un portafolio válido y que el cambio sea lo suficientemente pequeño para que el algoritmo explore de forma incremental. Hill Climbing evalúa la fitness del vecino y lo compara con la fitness actual. Si el vecino tiene un Sharpe Ratio mayor, el algoritmo cambia a esta nueva solución.

2.3.2. Generación de Vecinos en Simulated Annealing

Simulated Annealing permite cambios más amplios en la composición del portafolio para evitar quedarse atrapado en óptimos locales. La función `random_neighbor` genera un vecino intercambiando los pesos de dos activos aleatorios, lo cual puede dar lugar a una combinación de pesos muy distinta:

```
def random_neighbor(weights):
    # Selecciona dos indices aleatorios
    i, j = np.random.choice(len(weights), 2, replace=False)
    new_weights = weights.copy()
    # Intercambia los pesos
    new_weights[i], new_weights[j] = new_weights[j], new_weights[i
 ]
    return new_weights
```

Code 7. Función para generar vecinos en Simulated Annealing

Este intercambio permite explorar combinaciones radicalmente diferentes, lo que ayuda al algoritmo a escapar de los óptimos locales.

2.3.3. Proceso de Evaluación en Simulated Annealing

En Simulated Annealing, la aceptación de un vecino no solo depende de si su fitness es mejor, sino también de un factor de probabilidad que disminuye con el tiempo (temperatura).

2.3.4. Cálculo del cambio en fitness (delta)

Primero, el algoritmo calcula la diferencia delta entre la fitness de la solución actual y la fitness del vecino:

```
delta = current_fitness - new_fitness
```

Code 8. Hallando fitnes actual

2.3.5. Aceptación del vecino

Si el vecino tiene un Sharpe Ratio menor, aún puede ser aceptado con cierta probabilidad, definida como $\exp(-\delta/T)$, donde T es la temperatura. Esta probabilidad permite al algoritmo aceptar soluciones subóptimas en las primeras etapas, evitando caer en óptimos locales demasiado rápido.

```
def accept(delta, T):
    if delta < 0:  # Si el nuevo fitness es mejor, aceptar
        return True
    else:  # Acepta con probabilidad
        r = np.random.rand()  # Valor aleatorio entre [0, 1]
        return r < np.exp(-delta / T)
```

Code 9. Aceptación del vecino

3. Casos reales

3.1. Detección de abuso y fraude en servicios de streaming mediante Machine Learning con reconocimiento heurístico

La Plataforma de streaming Netflix desarrollo un framework para la detección de fraudes en su aplicación empleando algoritmos heurísticos desarrollados en la compañía y Machine Learning (ML), usando diferentes fuentes de datos como la trazabilidad de los usuarios, conexiones, contenido visualizado, ubicaciones, entre otros. Los autores Esmaeilzadeh et al. (2022) [3] expresan en su artículo que desarrollar un modelo ML para la detección de anomalías en el contexto de su aplicación implica muchos retos, entre ellos están, el análisis en tiempo real que puede llegar a ser costoso en términos de infraestructura y poca escalable, la definición de anomalía depende del contexto de negocio y aplicación y la cantidad de datos para alimentar al modelo, ya que al tratarse de casos no recurrentes se presenta una disparidad en la cantidad de datos. Debido a estas razones los autores plantean las siguientes soluciones. Primero, una solución basada en reglas que permita identificar irregularidades tomando como base el conocimiento y experiencia de los expertos de negocio, brindando características esenciales y contexto sobre los incidentes que permitan elaborar algoritmos heurísticos. Segundo, aplicar otra solución basada en modelo que permita identificar los casos anormales en la plataforma de forma automática. Para ejecutar este marco de trabajo primero se separaron en tres tipos de fraude, de cuentas, servicios y contenido, entonces la primera solución planteada complementa a la segunda, ya que ayuda a etiquetar y limpiar los datos que ingesta a la segunda solución, con la finalidad de identificar cual es el fraude que más se comete en la aplicación y poder identificar la cuenta asociada al cliente que infringen las políticas de la aplicación. Por último, con algoritmos heurísticos desarrollados con base a la experiencia y conocimiento de los expertos, complementa un modelo

ML incrementando su precisión hasta en un 86% en el análisis de fraude en tiempo real.

3.2. Uso de Machine Learning para predecir el próximo archivo

Un segundo caso de éxito lo aplica la empresa de servicio de almacenamiento en la nube Dropbox, en su artículo de ingeniería el autor Kumar (2019) [1] presenta como una funcionalidad de predicción de archivos que parece simple para el usuario tiene una gran complejidad por detrás, esto se debe a que para poder desarrollarlo realizaron algunos algoritmos heurísticos tomando como punto de inicio variables como la frecuencia con la que se accede a un archivo y el acceso reciente, sobre esto implementaron los algoritmos para obtener la probabilidad de cuál sería el próximo archivo que el usuario necesitaría, sin embargo, algunos obstáculos en ciertos escenario y el incremento en la complejidad computaciónal es que optaron por construir un primer modelo de Machine Learning que pueda analizar el comportamiento del usuario sobre los archivos que crea y modifica dentro de la aplicación.

4. Conclusiones

- Las soluciones clásicas son óptimas siempre y cuando no se tengan que analizar grandes cantidades de datos de ser el caso pasamos a un tiempo computaciónal fuera de las posibilidades y entendimiento humano por lo que se tiene que explorar otras soluciones que estén referenciadas en tiempo humano.
- Los algoritmos heurísticos se han mostrado efectivos para resolver problemas complejos en diversos contextos, ofreciendo soluciones escalables y eficientes. Al aplicar estas técnicas, es posible superar limitaciones de costo computaciónal y disponibilidad de datos, creando modelos que se adaptan bien a condiciones cambiantes y que permiten optimizar tanto el rendimiento como la capacidad de respuesta ante desafíos específicos. Estos casos de éxito subrayan la importancia de la heurística como herramienta fundamental en la búsqueda de soluciones prácticas y de alta eficiencia.
- De los casos de éxito se llega a la conclusión que se puede aplicar heurística en diferente entornos y problemas, mediante el desarrollo de algoritmos heurísticos se puede desarrollar soluciones escalables y eficientes, a modo de ejemplo el caso de Netflix que mediante estos algoritmos pudo entrenar un modelo de ML mucho más grande y solucionar el problema de fraude en su plataforma, también el caso de Dropbox que pudo desarrollar una funcionalidad para el usuario final brindando facilidad en la navegación entre los archivos de su plataforma.

Tabla de Contenidos

References

[1] N. Kumar, *Using machine learning to predict what file you need next*, May 2019. [Online]. Available: https://dropbox.tech/machine-learning/content-suggestions-machine-learning.

[2] A. Mari, “Aplicación de métodos greedy en problemas np-hard”, *Journal of Optimization*, 2020.

[3] S. Esmaeilzadeh, N. Salajegheh, A. Ziai, and J. Boote, *Abuse and fraud detection in streaming services using heuristic-aware machine learning*, Mar. 2022. [Online]. Available: https://arxiv.org/abs/2203.02124.