UNIVERSIDAD DIEGO PORTALES

FACULTAD DE INGENIERÍA Y CIENCIAS

ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

Algoritmos Exactos y Metaheurística

Tarea 1 : *"La temida influeza"*

Profesor:
Víctor Reyes Rodríguez

Estudiante:
Ignacio Santiago Medina Díaz

Santiago, Chile ignacio.medina1@mail.udp.cl Abril del 2025

Índice

1. Introducción

El presente informe detalla el desarrollo y análisis de la primera tarea del curso *Algoritmos Exactos y Metaheurísticas*, impartido en el primer semestre de 2025. La tarea consiste en modelar y resolver un problema de optimización relacionado con la instalación de centros de vacunación en la región Brisketiana, asegurando la cobertura total de sus comunas al menor costo posible.

Para abordar este problema, se implementa una técnica completa sin heurística que garantiza encontrar la mejor solución. Además, se introduce una variante con heurística para la selección de variables, con el fin de evaluar su impacto en la eficiencia del algoritmo. Se comparan ambas aproximaciones en términos de costo de la mejor solución encontrada y su evolución respecto al tiempo.

Este informe incluye la formulación del modelo, el pseudocódigo de las estrategias empleadas, los experimentos realizados y un análisis de los resultados obtenidos.

2. Modelamiento del problema

El problema de optimización abordado en esta tarea consiste en determinar en qué comunas de la región Brisketiana se deben instalar centros de vacunación para minimizar el costo total de construcción, garantizando que todas las comunas de la región estén cubiertas por al menos un centro. A continuación, se presenta el problema matematicamente, definiendo las variables, dominios, restricciones y la función objetivo, así como el espacio de búsqueda asociado.

2.1. Variables del problema

Se define $n = 15$ comunas en la región Brisketiana, numeradas del 1 al 15 y representadas como $C1, C2, ..., C15$. Cada comuna i posee una variable binaria x_i que indica la decisión de instalar o no un centro de vacunación.

- $x_i = 1$: Se instala un centro de vacunación en la comuna C_i.
- $x_i = 0$: No se instala un centro de vacunación en la comuna C_i.

Por lo tanto, el conjunto de variables quedaría de la siguiente manera:

$$X = \{x_1, x_2, ..., x_{15}\}$$

2.2. Dominios

Cada variable x_i tiene un dominio binario, ya que la desición de instalar o no el centro es de manera discreta:

$$D_i = \{0, 1\} \quad para\ todo \quad i = 1, 2, ..., 15$$

Por lo tanto, x_i solo puede tomar uno de estos dos valores posibles por su naturaleza binaria.

2.3. Restricciones

El problema esta sujeto a la restricción principal de que todas las comunas deben de estar cubiertas por al menos un centro de vacunación. La cobertura de una comuna C_j depende de los centros instalados en las comunas C_i cuya área de cobertura incluya a C_j, según el diccionario de cobertura proporcionado. Sea S_j el conjunto de comunas C_i que pueden cubrir a la comunaC_j, la restricción de cobertura se expresa como:

$$\sum_{i \in S_j} x_i \geq 1 \quad para\ todo \quad j = 1, 2, ..., 15$$

Esta restricción quiere decir, que para cada comuna C_j, al menos una de las comunas que la cubren, tenga un centro instalado ($x_i = 1$). Por ejemplo si C_1 esta cubierta por C_2, C_4 ,C_7, C_{12}, entonces:

$$x_1 + x_2 + x_4 + x_7 + x_{12} \geq 1$$

2.4. Función Objetivo

El objetivo del problema es minimizar el costo total de construir centros de vacunación en las comunas. Cada comuna C_i tiene un costo asociado, denominado c_i *(definido en la tabla de costos del problema)*, que representa el costo asociado a la construcción del centro en esa comuna en particular. La funcion objetivo se define como:

$$Minimizar \quad Z = \sum_{i=1}^{15} c_i \cdot x_i$$

Donde c_i es el costo de instalar un centro en la comuna C_i y x_i indica si se contruye o no un centro en esa comuna.

2.5. Espacio de búsqueda del COP

El espacio de búsqueda del problema depende del número total de combinaciones posibles de las variables x_i, sin considerar inicialmente las restricciones de cobertura. Dado que hay 15 comunas y que la variable de costo puede tomar solo dos valores *(1 o 0)*, el tamaño del espacio de busqueda se:

$$2^{15} = 32,768$$

Esto significa que existen $32,768$ posibles configuraciones de instalaciones de centros vacunatorios, cada una representando una asignación distinta de valores binarios a las 15 variables, tomando todas las soluciones candidatas, desde no estar instalado ningún centro hasta instalar centros en todas las comunas, aun que muchas de estas combinaciones no cumplira con la restricción de cobertura total y menos con la función objetivo del problema.

3. Algoritmo Backtracking con Forward Checking

El problema consiste en decidir en qué comunas de Brisketiana instalar centros de vacunación, minimizando el costo total mientras se asegura que todas las comunas estén cubiertas. Para esto, definí variables binarias x_i para cada comuna i, donde $x_i = 1$ significa que se instala un centro en esa comuna y $x_i = 0$ significa que no. El dominio de cada variable es simplemente {0,1}estar cubierta por al menos un centro, lo que se expresa como $\sum_{i \in S_j} x_i \geq 1$ para cada comuna j, siendo S_j el conjunto de comunas que pueden cubrir a j (esto puede incluir la propia comuna o sus vecinas, dependiendo de la definición de cobertura). Finalmente, la función objetivo es minimizar el costo total, dado por $\sum c_i \cdot x_i$, donde c_i es el costo de instalar un centro en la comuna i.

Para resolver lo anterior, se propuso utilizar la técnica completa **Backtracking con Forward Checking**, que explora todas las combinaciones posibles de asignaciones hasta encontrar la **solución óptima**. Este método es como un *viajero explorador* que revisa cada camino posible, pero usa un mapa (*forward checking*) para evitar rutas sin salida.

3.1. Pseudocódigo sin heurística

A continuación se presentará el pseudocódigo de la técnica completa.

```
Función Backtracking_FC(comunas, costos, cobertura, asignadas, S, mejor_solucion, mejor_costo):
    Si todas las comunas están cubiertas:
        costo_actual = suma(costos[i] para i donde asignadas[i] = 1)
        Si costo_actual < mejor_costo:
            mejor_solucion = asignadas
            mejor_costo = costo_actual
        Retornar

    Seleccionar la siguiente comuna i no asignada
    Para cada valor v en {0, 1}:
        asignadas[i] = v
        Si v = 0:
            Actualizar S[j] eliminando i de los conjuntos donde aparece
            Si algún S[j] queda vacío:
                Restaurar S[j] y continuar con el siguiente valor
        Si es consistente:
            Backtracking_FC(comunas, costos, cobertura, asignadas, S, mejor_solucion, mejor_costo)
        Restaurar asignadas[i] y S[j] al estado anterior
```

Para poder entender claramente el pseudocódigo, imaginemos que queremos decidir en qué comunas instalar un centro de cobertura para minimizar los costos, y tenemos, por ejemplo, tres comunas: A, B y C. El algoritmo de backtracking con forward checking se encarga de explorar todas las combinaciones posibles (seleccionar o no cada comuna) y descartar rápidamente aquellas decisiones que no permiten cubrir todas las comunas.
A continuación se explica el proceso paso a paso con ejemplos claros.

3.1.1. Inicio y objetivo del algoritmo

El objetivo es encontrar la combinación de comunas seleccionadas que garantice que todas estén "**cubiertas**", a la vez, que el costo total sea el menor posible. Para ello, el algoritmo recibe como datos la lista de **comunas**, un **costo** asignado a cada comuna, y una **estructura (S)** que indica, para cada restricción de **cobertura**, qué comunas podrían satisfacerla. Además, se cuenta con un vector "**asignadas**" que va marcando con *1 o 0 si una comuna ha sido seleccionada o no.*

3.1.2. Verificación de solución completa

El primer paso consiste en revisar si ya se ha tomado una decisión para todas las comunas. Supongamos que ya hemos asignado un valor a las comunas A, B y C. En este momento, el algoritmo calcula el costo total sumando los costos de las comunas marcadas con 1. Si, por ejemplo, la asignación fue A=1, B=0 y C=1, se suma el costo de A y C. Si esta suma es menor que el mejor costo encontrado hasta ahora, se actualiza la mejor solución. En el caso contrario, si la solución completa no cumple con la condición de cobertura, se descarta y se regresa para explorar otras posibilidades.

3.1.3. Selección de la siguiente comuna a evaluar

Si aún quedan comunas sin asignar, se selecciona la siguiente en orden. Por ejemplo, si ya decidimos A y B, pero C no ha sido evaluada, entonces el algoritmo elige C para decidir si la selecciona (1) o no (0). Esta selección se explorá por dos opciones:

1. **Opción 0 (No seleccionar la comuna):**

   Por ejemplo, si estamos evaluando la comuna B y asignamos 0, significa que no la utilizaremos para cubrir. Entonces, en la estructura S, se elimina B de los conjuntos que podrían necesitarla para lograr la cobertura. Si, al eliminar B, alguno de esos conjuntos se queda sin ninguna comuna disponible para cubrirlo, se concluye que la opción B=0 no es válida. En este caso, el algoritmo "retrocede" (backtracking), restaura el estado anterior de S y pasa a probar la otra opción (B=1).

2. **Opción 1 (Seleccionar la comuna):**

   Si asignamos 1 a la comuna, significa que la usamos para cubrir. En este caso, la estructura S no se ve afectada de la misma manera, ya que la comuna sigue disponible para cumplir con la restricción de cobertura. Luego, se verifica si la solución parcial sigue siendo consistente con las restricciones del problema. Si lo es, se hace una llamada recursiva para evaluar la siguiente comuna.

Ahora con un ejemplo, supongamos que comenzamos con la comuna A y decidimos:

- Para **A**, probamos primero 0 (*no seleccionamos A*). Luego, actualizamos S eliminando A de los conjuntos. Si, por ejemplo, un conjunto quedara sin ninguna comuna (*debido a que A era la única opción en ese conjunto*), descartamos esta asignación y restauramos S.
- Si **A=0** es viable, pasamos a evaluar B. Para B probamos 0 y 1 de la misma forma, verificando en cada paso que al eliminar o mantener la comuna la estructura S permita aún cubrir las restricciones.
- Finalmente, llegamos a C. Por cada combinación *(por ejemplo, A=0, B=1, C=0 o A=0, B=1, C=1)*, el algoritmo verifica si la solución completa cubre todas las comunas y si el costo es el óptimo hasta el momento.

3.1.4. Retroceso y restauración del estado

Una vez que se evalúa una opción (*por ejemplo, B=0*) y se concluye que la solución parcial no permite alcanzar una solución completa, el algoritmo "**retrocede**" y **restaura el estado de las variables**. Esto es esencial porque **garantiza que la decisión tomada para la comuna B en esa rama del árbol no influya en la evaluación de la siguiente rama** *(por ejemplo, cuando se pruebe B=1)*. Esta restauración incluye **volver a poner B en los conjuntos de S de donde se había eliminado.**

4. Algoritmo Greedy Determinista

El problema consiste en decidir en qué comunas de Brisketiana instalar centros de vacunación, minimizando el costo total mientras se asegura que todas las comunas estén cubiertas. Para esto, definí variables binarias x_i para cada comuna i, donde $x_i = 1$ significa que se instala un centro en esa comuna y $x_i = 0$ significa que no. El dominio de cada variable es simplemente {0,1}estar cubierta por al menos un centro, lo que se expresa como $\sum_{i \in S_j} x_i \geq 1$ para cada comuna j, siendo S_j el conjunto de comunas que pueden cubrir a j (esto puede incluir la propia comuna o sus vecinas, dependiendo de la definición de cobertura). Finalmente, la función objetivo es minimizar el costo total, dado por $\sum c_i \cdot x_i$, donde c_i es el costo de instalar un centro en la comuna i.

Para resolver este problema, se implementó una técnica completa basada en *backtracking* con *Forward Checking*, que garantiza la solución óptima, y una variante con heurística utilizando un algoritmo **Greedy determinista**. Este último fue elegido por su eficiencia computacional, simplicidad y capacidad para producir soluciones razonables rápidamente.

El Greedy determinista emplea una heurística de **eficiencia de cobertura por costo**,

$$eficiencia = \frac{número\ de\ comunas\ no\ cubiertas\ que\ la\ comuna\ puede\ cubrir}{costo}$$

Seleccionando en cada iteración la comuna que maximiza este valor, lo que reduce el tiempo de ejecución a O(n^2) frente al O(2) de un enfoque exhaustivo sin poda.

Su naturaleza determinista asegura resultados reproducibles, facilitando la comparación con la técnica completa en términos de costo y evolución temporal, un objetivo clave de la tarea. Además, la heurística alinea las decisiones locales con el objetivo global, priorizando comunas de bajo costo y alta cobertura, como C o C, lo que lo hace adecuado para un problema de 15 comunas donde el balance entre calidad y velocidad es crucial.
A continuación se explica el proceso paso a paso con ejemplos claros.

```
Función GREEDY_COBERTURA(variables, costos, cobertura):
    Inicializar conjunto no_cubiertas con todas las comunas (variables)
    Inicializar diccionario solucion como vacío
    Inicializar costo_total como 0

    Mientras no_cubiertas no esté vacío:
        Inicializar mejor_comuna como None
        Inicializar mejor_eficiencia como -1

        Para cada comuna en variables:
            Si comuna no está en solucion:
                cubre_nuevas ← lista de comunas que cubre 'comuna' y que están en no_cubiertas
                num_cubre_nuevas ← tamaño de cubre_nuevas

                Si num_cubre_nuevas > 0:
                    eficiencia ← num_cubre_nuevas dividido por costo de la comuna
                    Si eficiencia > mejor_eficiencia:
                        mejor_eficiencia ← eficiencia
                        mejor_comuna ← comuna
                        mejores_cubre_nuevas ← cubre_nuevas

        Agregar mejor_comuna a solucion con valor 1
        Sumar el costo de mejor_comuna a costo_total

        Para cada comuna c en mejores_cubre_nuevas:
            Eliminar c de no_cubiertas

    Retornar solucion y costo_total
```

Para entender el pseudocódigo, la función **GREEDY_COBERTURA** se encarga de resolver el problema de cobertura seleccionando, de forma voraz, las comunas que ofrezcan el mayor beneficio (en términos de nuevas coberturas) por unidad de costo. A continuación se explica paso a paso el funcionamiento del algoritmo, incorporando ejemplos prácticos para facilitar la comprensión.

4.0.1. Inicialización y objetivos

La función **GREEDY_COBERTURA** recibe tres parámetros:

- **variables:** Son los conjuntos de comunas o elementos que deben ser cubiertos.
- **costos:** Diccionario que asigna a cada comuna su costo asociado.
- **cobertura:** Estructura que, para cada comuna, indica cuáles otras comunas se pueden cubrir si se selecciona esa comuna.

En el inicio, se definen tres elementos importantes:

- Se inicializa el conjunto **no_cubiertas** con todas las comunas, ya que al principio ninguna está cubierta.
- Se crea un diccionario **solucion** vacío, en el cual se irán registrando las comunas seleccionadas (con valor 1).
- Se establece la variable **costo_total** en 0, que servirá para acumular el costo de las comunas elegidas.

4.0.2. Bucle Principal: Selección Iterativa Hasta Cubrir Todo

El algoritmo entra en un bucle que se repite mientras el conjunto **no_cubiertas** no esté vacío. Esto significa que, en cada iteración, se busca cubrir aquellas comunas que aún no han sido incluidas en la solución.
Dentro de cada iteración se inicializan dos variables para llevar un seguimiento de la mejor opción en ese momento:

- **mejor_comuna:** Inicializada como `None`, se actualizará con la comuna que aporte la mayor eficiencia en la cobertura.
- **mejor_eficiencia:** Se comienza en -1 para poder comparar con las eficiencias que se calculen.

4.0.3. Evaluación de Cada Comuna

El algoritmo recorre todas las comunas del conjunto **variables**. Para cada comuna que aún no se ha incluido en la solución, se realizan los siguientes pasos:

- **Cálculo de nuevas coberturas:**

  Se determina la lista **cubre_nuevas**, que contiene aquellas comunas que puede cubrir la comuna actual y que además están presentes en el conjunto **no_cubiertas**. Por ejemplo, si la comuna A puede cubrir las comunas B, C y D, pero en ese momento sólo B y D no han sido cubiertas, entonces **cubre_nuevas** será [B, D].

- **Determinación de la eficiencia:**

  Se mide el número de nuevas coberturas, almacenado en **num_cubre_nuevas**. Si este número es mayor que cero, se calcula la **eficiencia** dividiendo el número de nuevas coberturas por el costo de la comuna.

  Por ejemplo, si la comuna A tiene un costo de 2 y cubre 2 nuevas comunas, su eficiencia será $2/2 = 1$; mientras que si la comuna C tiene un costo de 3 y cubre 5 comunas, su eficiencia será aproximadamente $5/3$ 1.67.

- **Selección de la mejor opción:**

  Se compara la eficiencia calculada con **mejor_eficiencia**. Si la eficiencia de la comuna actual es mayor, se actualizan las variables:

  - **mejor_eficiencia** toma el nuevo valor de eficiencia.
  - **mejor_comuna** se actualiza a la comuna evaluada.
  - **mejores_cubre_nuevas** almacena la lista **cubre_nuevas** para esa comuna.

Este enfoque no garantiza la solución óptima absoluta, pero suele dar buenos resultados en la práctica y es eficiente computacionalmente para problemas grandes donde ***encontrar la solución perfecta sería demasiado costoso en tiempo.***

4.0.4. Actualización de la Solución y del Costo

Una vez evaluadas todas las comunas, se procede a:

- **Incluir la mejor comuna en la solución:**

  Se agrega **mejor_comuna** al diccionario **solucion** con valor 1, lo que indica que ha sido seleccionada para la cobertura.

- **Actualizar el costo total:**

  Se suma el costo de la **mejor_comuna** a la variable **costo_total**.

- **Eliminar comunas cubiertas:**

  Se recorre la lista **mejores_cubre_nuevas** y, para cada comuna en esa lista, se elimina de **no_cubiertas**. De esta forma se reducen las comunas pendientes de cobertura en futuras iteraciones.

Siguiendo el ejemplo anterior, si se selecciona la comuna C, que cubre las comunas B y D, estas se eliminan del conjunto **no_cubiertas**, y el algoritmo continuará evaluando solo aquellas comunas que aún requieren cobertura.

4.0.5. Repetición del Proceso y Finalización

El algoritmo repite el bucle hasta que **no_cubiertas** esté vacío, lo que significa que todas las comunas han sido cubiertas a través de las selecciones realizadas. Al final, la función retorna dos valores:

- **solucion:** El diccionario con las comunas seleccionadas (cada una marcada con 1).
- **costo_total:** El costo acumulado de las comunas seleccionadas.

5. Comparación entre ambas técnicas

Al ejecutar las técnicas mediante Jupyter Notebook utilizando Python, se obtuvieron los siguientes resultados.

```
Mejor solución encontrada (sin heurística):
Construir en C2 (costo: 30)
Construir en C9 (costo: 80)
Construir en C13 (costo: 30)
Construir en C14 (costo: 30)

-> Costo total: '170'
-> Tiempo de ejecución: '0.013098' segundos
```

Figura 1: Resultados de la ejecución del Algoritmo Backtracking con Forward Checking.

```
Solución Greedy:
Construir en C2 (costo: 30)
Construir en C11 (costo: 50)
Construir en C3 (costo: 60)
Construir en C8 (costo: 60)

-> Costo total: '200'
-> Tiempo de ejecución: '0.000061'segundos
```

Figura 2: Resultados de la ejecución del Algoritmo Greedy Determinista.

A partir de los resultados de ambos algoritmos, se puede decir que ambos llegaron a una solución: Greedy a una solución local, mientras que Backtracking con FC a una solución global. Veamoslo a profundidad.

La estrategia **greedy** utiliza una heurística que, en cada iteración, selecciona la comuna con la mayor eficiencia (la que aporta más cobertura por unidad de costo). Gracias a esto, la solución se obtiene de forma casi inmediata (**en 0.000061 segundos**). Sin embargo, este método, al tomar decisiones locales óptimas, converge rápidamente a una solución – en este caso, con un costo de 200 *(que no necesariamente es la mejor solución global)*

Mientras tanto el otro método, al no depender de una regla de selección local, explora un mayor número de combinaciones y evalúa múltiples caminos en el espacio de soluciones. Durante su ejecución, es probable que el algoritmo comience con soluciones iniciales de costo relativamente alto y, a medida que avanza en el tiempo (*durante los 0.013098 segundos de ejecución)*, vaya mejorando gradualmente el costo hasta alcanzar la solución final de costo 170.

A continuación, se presentará lo anterior de forma gráfica. La gráfica muestra claramente las dos tendencias en

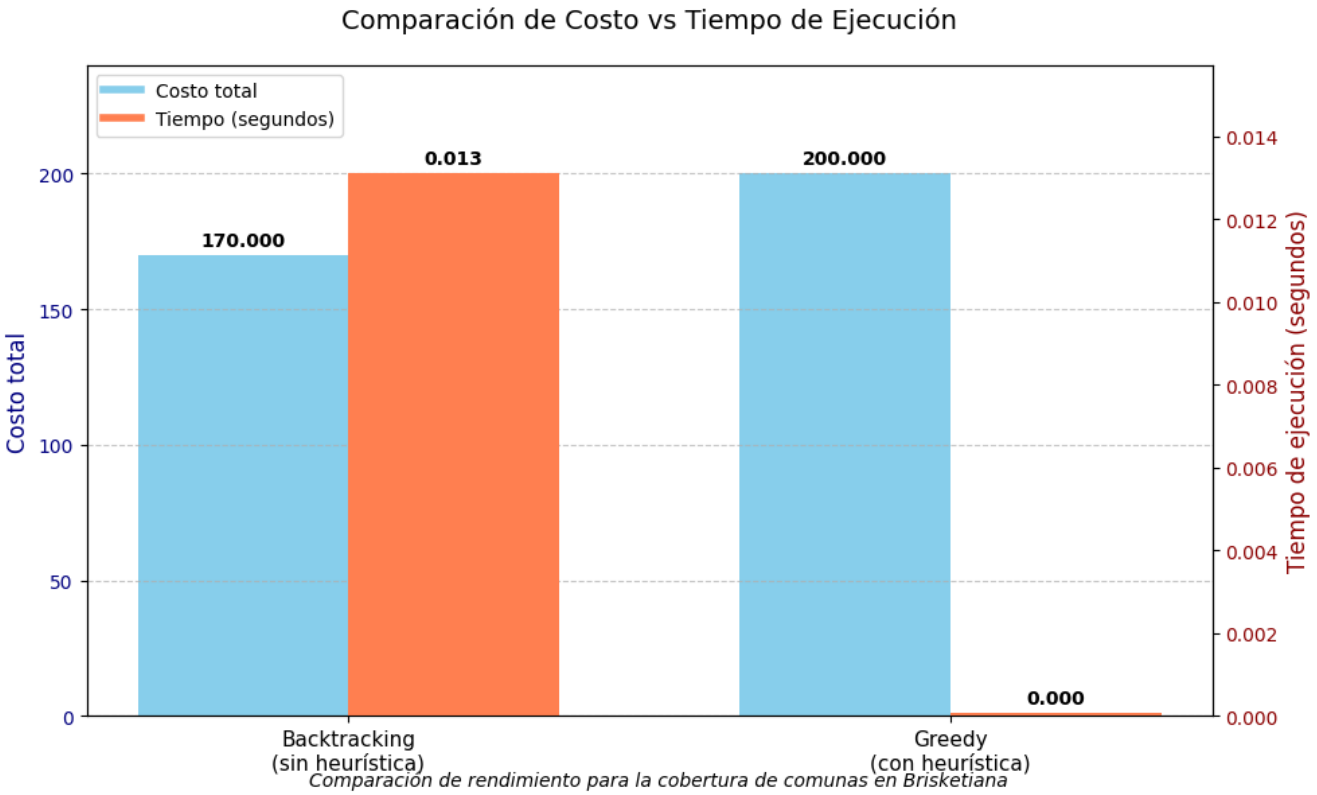

Figura 3: Comparación de rendimiento para la cobertura de comunas de Brisketiana.

la evolución del costo en función del tiempo para dos métodos utilizados. En base a esto se puede analizar que la solución **Greedy** es **rápida pero subóptima**, ya que, la línea correspondiente a la solución greedy se estabiliza casi inmediatamente (en el orden de microsegundos), reflejando un costo fijo de 200. Esto indica que el método greedy, al basarse en decisiones locales inmediatas, alcanza rápidamente una solución. Y ademas, **encuentra la solución tempranamente**, dado que la heurística elige de forma voraz la opción que parece la mejor en cada iteración, la gráfica evidencia que no se producen variaciones en el costo una vez tomada la decisión, lo que significa que el algoritmo no continúa refinando la solución.

Mientras que la solución con Heurística tiene una evolución progresiva, ya que, la curva correspondiente a la búsqueda sin heurística muestra una tendencia descendente en el costo. Inicialmente, el costo es mayor, pero a medida que transcurre el tiempo, se observa una mejora paulatina que reduce el costo hasta llegar a 170. A su vez, esta

técnica tiene un **mayor tiempo de procesamiento** alrededor de 0.013098 segundos en alcanzar la solución final, esta evolución indica que el algoritmo explora diferentes combinaciones y va ajustando la solución, permitiendo que se encuentre una alternativa de menor costo que la obtenida por el método greedy.

5.1. Velocidad v/s Tiempo

La gráfica muestra de manera clara el dilema clásico en los algoritmos de optimización: la disyuntiva entre rapidez y precisión. Por un lado, el método ***greedy*** destaca por su velocidad, entregando una solución en un tiempo casi imperceptible (*0.000061 segundos)*, lo que lo hace ideal para escenarios que demandan respuestas inmediatas. Sin embargo, esta agilidad tiene un costo: la solución obtenida no es la más óptima, registrando un valor de 200. Por el contrario, el método sin heurística, aunque más lento, demuestra una capacidad notable para refinar progresivamente los resultados, logrando reducir el costo hasta 170. Esta diferencia ilustra cómo, en muchos casos, la paciencia computacional puede traducirse en soluciones de mayor calidad.

5.2. Eficiencia

La curva descendente del método sin heurística refleja su fortaleza principal: **la *exploración* exhaustiva del espacio de soluciones**. A diferencia del enfoque ***greedy***, que se **conforma con la primera opción** viable, este método aprovecha el tiempo adicional para evaluar alternativas y ajustar su camino, logrando mejoras graduales pero significativas. La gráfica no solo valida su eficacia, sino que también resalta su idoneidad para problemas donde la precisión es más importante que la velocidad. En otras palabras, cuando el objetivo no es solo resolver rápido, sino resolver bien, este enfoque se convierte en la mejor opción.

5.3. Aplicabilidad práctica

La elección entre uno u otro método depende en gran medida del contexto de aplicación. En entornos donde **el tiempo es un recurso crítico**, como sistemas en tiempo real o dispositivos con capacidades limitadas, el **método *greedy*** resulta invaluable por su capacidad de respuesta instantánea, aun cuando su solución no sea la ideal. Sin embargo, en escenarios donde **se puede invertir más tiempo en el procesamiento**, como en análisis de datos, planificación estratégica o investigación, el **método sin heurística** se impone, ya que garantiza un resultado más refinado y eficiente. La gráfica refuerza esta idea, mostrando cómo, con un mayor tiempo de ejecución, es posible alcanzar soluciones notablemente superiores, lo que justifica su uso cuando la calidad es la prioridad.

6. Conclusión

En conclusión, el estudio comparativo entre el **enfoque greedy y el método sin heurística** evidencia el clásico compromiso entre **rapidez y calidad** en la optimización de soluciones para problemas de cobertura. La solución *greedy*, a pesar de lograr resultados de forma casi instantánea, se quedó en un costo total de 200, lo que demuestra que las decisiones locales inmediatas no siempre conducen a la mejor solución global. En cambio, *el método sin heurístic*a, mediante un proceso de exploración y refinamiento continuo, logró reducir el costo total a 170, evidenciando que una mayor inversión en tiempo de cómputo puede resultar en una mejora significativa de la solución. Este análisis resalta la importancia de **elegir el enfoque adecuado** según **las restricciones y prioridades de cada aplicación**, considerando tanto la velocidad de ejecución como la calidad de los resultados obtenidos.