Proyecto Final

Búsqueda y optimización

Autora

Laura González Pizarro, 100538522

Noviembre 2024

Índice

1. Problema

Una fábrica recibe todas las mañanas un pedido con las tareas que debe realizar durante el día. Cada una de las tareas pertenece a uno, y sólo uno de los tipos a, b, c o d, y se sabe el tiempo exacto de procesamiento requerido para su realización. Para ello, la fábrica dispone de una cantidad arbitraria de máquinas dispuestas en paralelo, una o más de cada uno de los tipos indicados, de modo que cada máquina sólo puede realizar tareas de su tipo. Se desea determinar el orden y asignación de tareas a máquinas de tal modo que el tiempo total de realización de todas las tareas sea el mínimo.

2. Espacio de estados

Para abordar este problema es fundamental definir el espacio de estados, el cual describe el conjunto de estados, el estado inicial, estado final, así como, el conjunto de acciones para transitar entre estados y el coste asociado a cada una de ellas. En este caso, se ha seguido una orientación basada en programación orientada a objetos, utilizando clases con atributos y métodos para la representar del problema. A continuación, se detallan los componentes que se utilizarán para la representación de un estado de cara a la implementación:

- **Tarea**: cada tarea se representa mediante dos atributos: su tipo (a, b, c o d) y su duración, que es el tiempo necesario para completarla.
- **Máquina**: una máquina está caracterizada por el tipo de tarea que puede ejecutar y un conjunto de tareas asignadas a ella. Para su implementación, se utilizará un conjunto (*set*) para almacenar las tareas, ya que el orden en que se realicen no afecta el tiempo total de ocupación.
- **Fábrica**: se modela como un conjunto de máquinas. En la práctica una fábrica estará modelada como un diccionario donde la clave es el tipo de tarea que realiza la máquina. Los valores serán una tupla de longitud el número de tareas a realizar de ese tipo y tendrán como valor el índice de la máquina a la que será asingada, ya que puede haber más de una máquina que realice el mismo tipo tareas.
- **Demanda**: la demanda de la fábrica se representa como un conjunto de tareas que deben ser realizadas. En la implementación se modelará mediante una lista de tareas.

Por tanto, el estado esta representado por la demanda de tareas a realizar y por la fábrica con las diferentes máquinas en cada momento del proceso.

La única **operación** es asignar una tarea a una máquina. Esta acción tiene como precondición que la tarea debe de ser del mismo tipo que la máquina, es decir, una tarea del tipo a solo puede asignarse a una máquina que procese tareas de tipo a. Y como precondición se tiene que la tarea se ha asignado a

la máquina

La **función de coste** se calcula como la diferencia entre el tiempo de la máquina a la que se asigna la nueva tarea y el tiempo de la máquina que mas tarda en completar todas sus tareas, si al añadir la tarea esta máquina se convierte en la que más tarda; de lo contrario, es cero. El coste total es el tiempo que tarda la máquina con más tareas en completar su trabajo, independientemente de su tipo.

En la Figura 3 se presenta un ejemplo en el que, al añadir una nueva tarea, el *makespan* (el tiempo total necesario para completar todas las tareas) aumenta, ya que la máquina con más carga ahora es aquella a la que se le ha asignado la nueva tarea. Por otro lado, en la Figura 2, se puede observar un caso en el que el *makespan* no se incrementa tras añadir una nueva tarea, ya que no provoca que ninguna máquina exceda el tiempo de la que tenía la mayor carga.

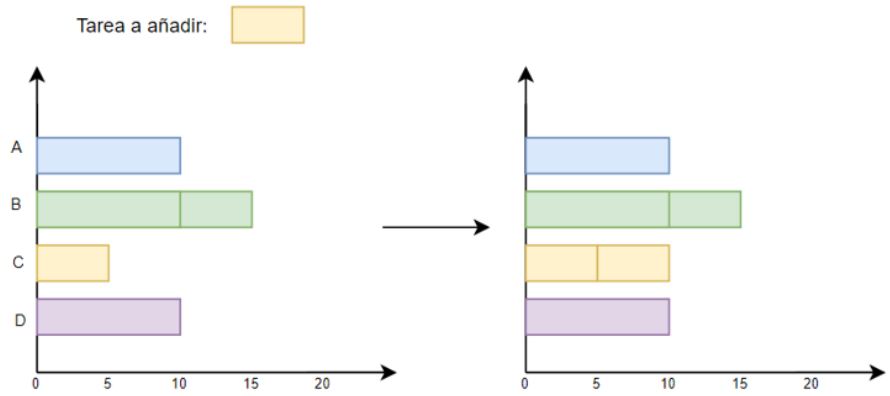

Figura 1: El *makespan* aumenta tras añadir una nueva tarea a una máquina.

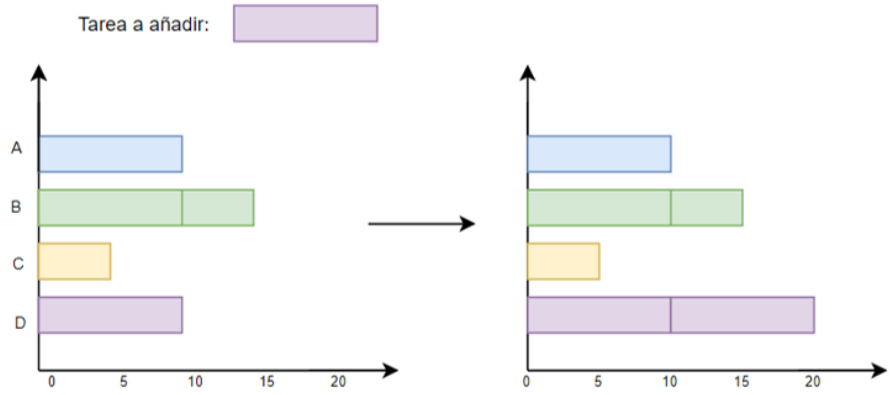

Figura 2: El *makespan* no se ve afectado tras añadir una nueva tarea.

El **estado inicial** del problema se define como la situación en la que la demanda está completa y no se ha asignado ninguna tarea a ninguna máquina. Por otro lado, el **estado final** se caracteriza por una demanda vacía, con todas las tareas asignadas a las máquinas de tal manera que el tiempo total de realización sea el mínimo.

2.1. Heurística para el problema

Para poder resolver este problema mediante algoritmos de búsqueda informada es necesario definir una heurística que guíe la toma de decisiones.

Se conoce el estado de la demanda, tanto la que ya ha sido asignada como la que aún no lo ha sido, así como el número de máquinas disponibles de cada tipo en la fábrica. Sea T_a, T_b, T_c, T_d el tiempo total requerido para completar las tareas pendientes de los tipos a, b, c y d. Además, t_a, t_b, t_c, t_d, representan el tiempo que tarda cada máquina en realizar las tareas asignadas de cada tipo. Mientras que M_a, M_b, M_c, M_d es el número de las máquinas disponibles de cada tipo.

Una heurística basada en relajación de restricciones sobre el espacio de estados para este problema consiste en considerar el tiempo máximo estimado que tardaría cada tipo de máquina en completar las tareas pendientes, dadas las máquinas disponibles y el estado actual de la demanda. Esta fórmula estima el tiempo máximo que se necesitaría para completar todas las tareas pendientes en el sistema, ajustando el tiempo de espera en función de la capacidad de procesamiento efectiva de cada grupo de máquinas. De esta manera, la heurística se puede definir como:

$$h(n) = \max_{i=a\ldots d} \frac{T_i - \max\limits_{j \in M_i} t_j + \min\limits_{j \in M_i} t_j}{M_i}$$

Esta heurística es admisible, es decir, que nunca sobrestima el coste real para completar el problema. Esto se debe a que, al calcular el tiempo mínimo posible asumiendo una distribución óptima de las tareas entre las máquinas, la heurística siempre proporcionará un valor igual o menor que el coste real.

Además, es una heurística informada, es decir, dado que $h(n) \geq h_2(n)$, para cualquier estado, donde h_2 una heurística menos informada. Un ejemplo de heurística menos para este problema podría ser aquella que evalúa el tiempo máximo necesario para completar las tareas pendientes, pero ajustado por el desempeño colectivo de las máquinas en lugar de solo los extremos. Esta nueva heurística se define como:

$$h_2(n) = \max_{i=a\ldots d} \frac{T_i - \sum_{j=1}^{M_i}(T_{\max_i} - t_{ij})}{M_i}$$

Donde $T_{\max_j} = \max_{i \in M_i} t_j$.

En la Figura 3 se presenta un ejemplo en el que, al añadir una nueva tarea, el *makespan* aumenta, ya que la máquina con más carga ahora es aquella a la que se le ha asignado la nueva tarea. Se puede apreciar como la heurística no sobrestima el coste real ya que $f + g = 15 + \frac{5}{3} < h^* = 5 + 15 = 20$

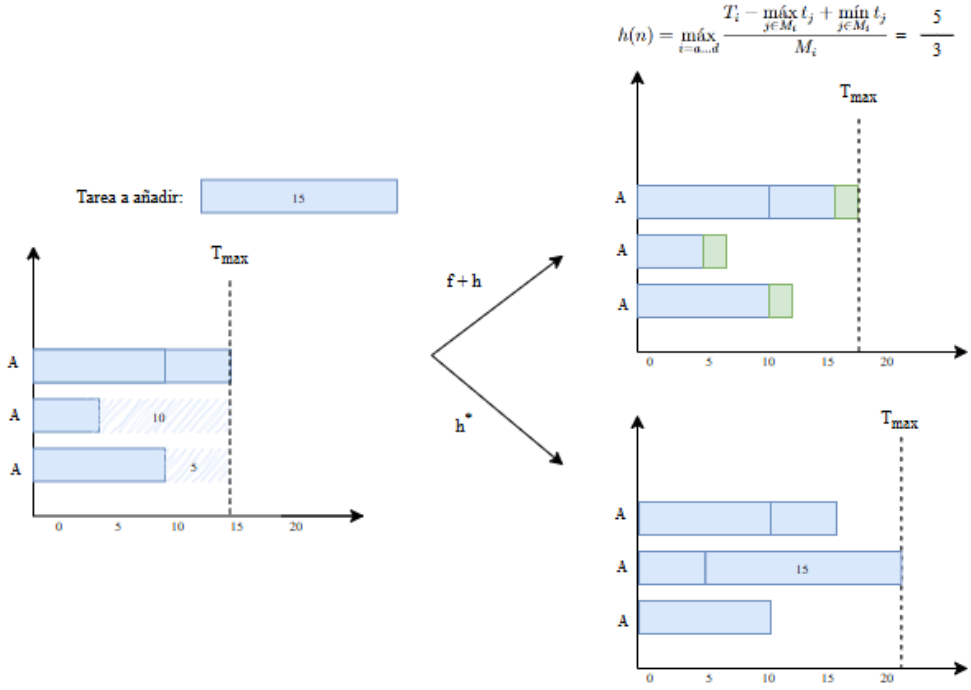

Figura 3: El *makespan* aumenta tras añadir una nueva tarea a una máquina.

3. Implementación del problema

Para la implementación de este problema se ha elegido Python como lenguaje de programación. Se han implementado clases y funciones que representan las distintas entidades del problema, permitiendo organizar el código de manera modular y facilitar su expansión futura. Además, Python ofrece bibliotecas y herramientas que simplifican tanto la implementación del problema como la visualización de resultados.

3.1. Representación del espacio de estados

Este problema puede dividirse en cuatro subproblemas, separando cada uno según el tipo de máquina. Esto se debe a que el tiempo máximo de ocupación depende de la máquina que tarde más en completar sus tareas, sin afectar a máquinas de otros tipos. Así, la solución óptima del problema completo se obtiene como la unión de las soluciones óptimas de los subproblemas.

Para la representación del espacio de estados, se ha seguido el enfoque descrito en la Sección 2, con algunas adaptaciones para ajustarlo a los algoritmos de búsqueda. Un estado se define mediante:

- La demanda restante.
- La configuración actual de las máquinas, es decir, una tupla del tamaño de las tareas ya asignadas donde cada elemento indica en que máquina ha sido asignada.
- El número de máquinas disponibles dado que, como sólo se guarda la configuración de las tareas asignadas, es necesario saber cuantas máquinas disponibles hay.
- El tiempo de la máquina que más tarda en realizar las tareas en ese punto, es decir, el coste en ese instante.
- El tiempo de la máquina que menos tarda en realizar sus tareas.

3.2. Implementación de algoritmos de búsqueda

En este apartado se comentarán los diferentes algoritmos de búsqueda implementados, detallando las decisiones tomadas en su implementación.

3.2.1. Fuerza bruta

La primera aproximación para resolver este problema es fuerza bruta. Este enfoque generan todas las posibles asignaciones de las tareas a las máquinas y se selecciona la de menor coste. Este algoritmo genera m^n donde m el número de máquinas y n es el número de tareas. A medida que aumenta el número de tareas el problema se vuelve intratable debido a la gran cantidad de nodos que se deben generar.

3.2.2. A*

Se ha implementado el algoritmo de búsqueda heurística unidireccional A*. Este algoritmo emplea una lista abierta que ordena los nodos a expandir en orden ascendente según el criterio $f = g + h$, y una lista cerrada para evitar la expansión repetida de nodos ya visitados. Para la lista abierta, se ha optado por utilizar un montículo (heap), mientras que para la lista cerrada se ha empleado un conjunto (*set*), dado que comprobar la pertenencia a un conjunto tiene un coste constante $O(1)$.

Se ha realizado un estudio para determinar qué heurística utilizar en este problema, comparando el tiempo de ejecución y el número de nodos expandidos. Las heurísticas que se han comparado son las siguientes:

$$h_1(n) = \max_{i=a\ldots d} \frac{T_i - \max\limits_{j \in M_i} t_j + \min\limits_{j \in M_i} t_j}{M_i}$$

$$h_2(n) = \max_{i=a\ldots d} \frac{T_i - \sum_{j=1}^{M_i} (T_{\max_i} - t_{ij})}{M_i}$$

A continuación, se presentan unas gráficas comparativas de las dos heurísticas, donde se muestra el tiempo de ejecución y número de nodos expandidos a medida que se ha aumentado el número de tareas por tipo de máquina, considerando que el número de máquinas es cuatro en cada una de las iteraciones.

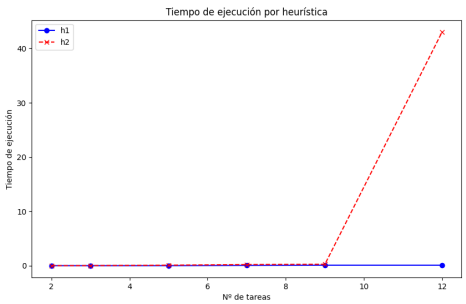

Figura 4: Tiempo de ejecución.

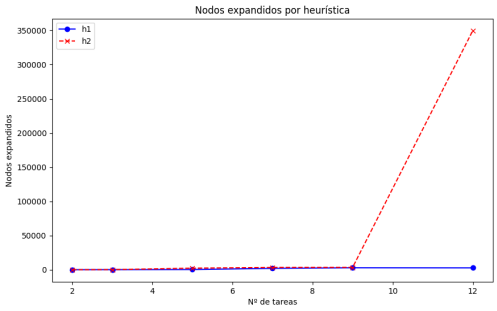

Figura 5: Nodos expandidos.

		Número de tareas por máquina					
		2	3	5	7	9	12
Nodos expandidos	h_1	60	160	304	2024	2872	2736
	h_2	52	196	2308	3416	3480	349764

Tabla 1: Número de nodos expandidos por heurística y número de tareas

Como se puede apreciar en la Tabla 1, el número de nodos expandidos y por tanto el tiempo de ejecución con h_1 es significativamente menor en

comparación con h_2. Esto se debe a que h_1 es más informada, lo que permite encontrar la solución de manera más rápida. Por lo tanto, se ha seleccionado h_1 como heurística para los experimentos posteriores.

3.2.3. Weighted A*

También se ha implementado el algoritmo *weighted* A* (wA*), cuya única diferencia con el algoritmo A* es el peso adicional aplicado al valor de la heurística. A diferencia de A*, el algoritmo wA* no garantiza encontrar soluciones óptimas, ya que el peso aplicado a la heurística puede hacer que el criterio de selección de nodos favorezca soluciones subóptimas en favor de una expansión más rápida.

Se ha realizado un estudio para determinar el parámetro de peso más adecuado para este problema, probando diversos valores de peso y comparando el tiempo de ejecución y el número de nodos expandidos. A continuación se presentan unas gráficas comparativas de los diferentes pesos a asignar a la heurística, donde se muestra el número de nodos expandidos y el error cometido, es decir, la diferencia entre el resultado obtenido menos la solución optima, a medida que se ha aumentado el número de tareas por tipo de máquina, considerando que el número de máquinas es cuatro en cada una de las iteraciones.

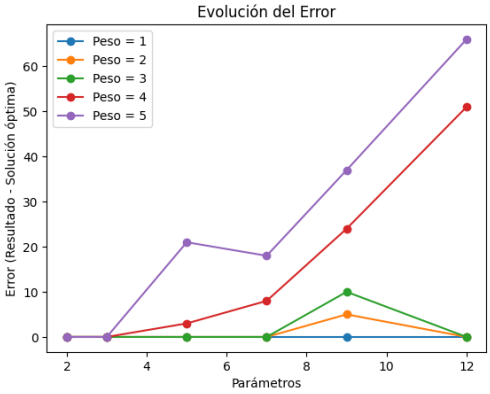

Figura 6: Error cometido.

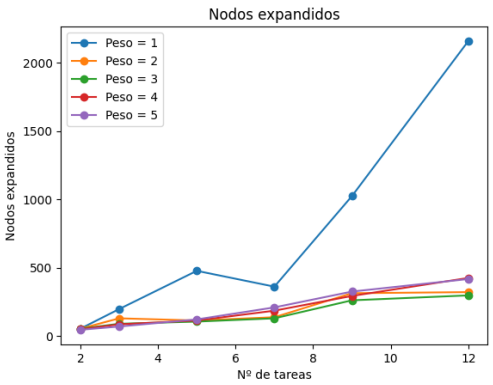

Figura 7: Nodos expandidos.

Como se puede ver en la Figura 7 a media que se va aumentando el peso de la heuristica el error que se comete es mayor. Mientras que en la Tabla 2 se observa que a media que se va aumentando el peso del número de la heurística, el número de nodos decrece y, por tanto, el tiempo de ejecución para el mismo número de tareas a realizar. Por lo tanto, se ha seleccionado un peso de 2 para el algoritmo wA* en los experimentos posteriores, dado que ofrece el mejor compromiso entre el error cometido y el número de nodos expandidos.

		Número de tareas por máquina					
		2	3	5	7	9	12
Peso	1	64	92	164	228	388	2924
	2	60	84	104	176	256	1020
	3	56	84	104	152	256	1032
	4	48	84	112	148	316	512
	5	40	84	108	212	304	524

Tabla 2: Número de nodos expandidos por peso y número de tareas

3.2.4. Monte Carlo

Se ha implementado el algoritmo de búsqueda no óptima Monte Carlo, dado que este problema siempre tiene solución, no se debe comprobar si el estado es meta. Se ha implementado de manera que selecciona una asignación aleatoria de las tareas en las máquinas. A diferencia de otros métodos, este algoritmo expande únicamente un nodo por iteración. El algoritmo es especialmente adecuado cuando el número de tareas es muy grande, sin embargo, su precisión se ve afectada negativamente a medida que aumenta la diferencia en los tiempos de las tareas, lo que puede resultar en un mayor margen de error.

3.3. Experimentos realizados

En este apartado se comentará los experimentos realizados para comparar los diferentes algoritmos de búsqueda.

El primer experimento analiza el tiempo de ejecución y los nodos expandidos a medida que se va incrementando el número de tareas, manteniendo constante el número de máquinas de cada tipo de tarea, que en este caso es cuatro. A continuación se muestran las gráficas comparativas de los diferentes algoritmos:

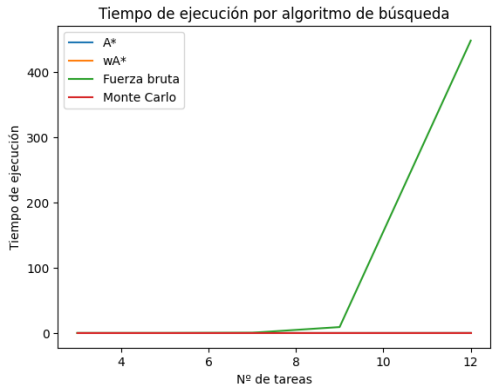

Figura 8: Tiempo de ejecución.

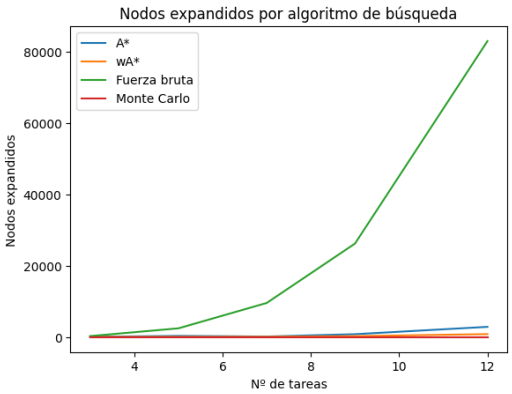

Figura 9: Nodos expandidos.

		Número de tareas por máquina				
		2	3	5	7	9
Algoritmo	A*	120	380	204	856	2916
	wA*	84	108	136	320	876
	Fuerza Bruta	324	2500	9604	26244	82944
	Monte Carlo	4	4	4	4	4

Tabla 3: Número de nodos expandidos por algoritmo y número de tareas

Como se puede observar en las Figuras 8 y 9, a medida que se va aumentando el número de tareas a asignar a las máquinas, el tiempo de ejecución y los nodos expandidos también aumentan. En el caso de fuerza bruta, este incremento es exponencial, mientras que para A* y wA* el aumento es más gradual, y para Monte Carlo, el tiempo es constante.

La Tabla 3 muestra el número de nodos expandidos a medida que se van aumentando el número de tareas. Se observa que el algoritmo A* expande significativamente menos nodos en comparación con el de fuerza bruta. A medida que aumenta el número de tareas, esta diferencia se acentúa, alcanzando órdenes de magnitud superiores. Por otro lado, el algoritmo wA* también expande menos nodos debido a su capacidad para encontrar soluciones más rápidamente (necesariamente óptimas) al priorizar la heurística. Por último, el método Monte Carlo genera únicamente cuatro nodos, que corresponden a las soluciones óptimas de los subproblemas.

El siguiente experimento compara el tiempo de ejecución y el número de nodos expandidos a medida que se incrementa el número de máquinas en la fábrica, manteniendo constante el número de tareas por tipo de máquina, que en este caso es siete. A continuación, se presentan las gráficas comparativas de los distintos algoritmos de búsqueda:

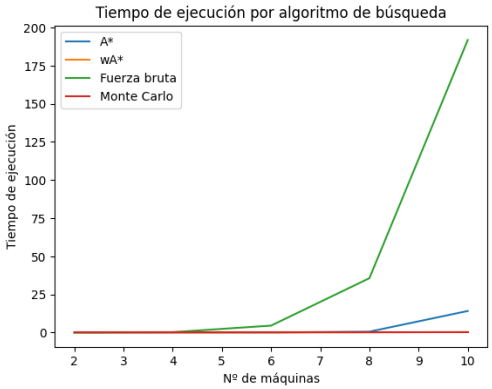

Figura 10: Tiempo de ejecución.

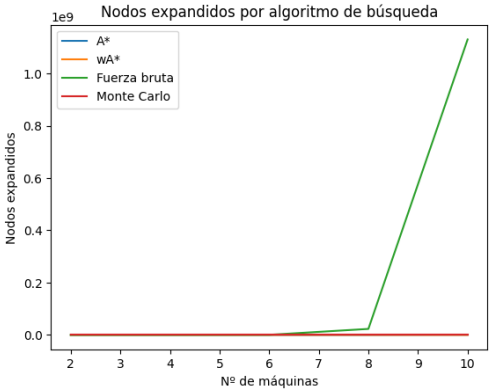

Figura 11: Nodos expandidos.

		Número de máquinas				
		2	4	6	8	10
Algoritmo	A*	232	468	1638	16672	516990
	wA*	98	140	294	3784	11180
	Fuerza Bruta	196	9604	470596	23059204	1129900996
	Monte Carlo	4	4	4	4	4

Tabla 4: Número de nodos expandidos por algoritmo y número de máquinas

Las Figuras 10 y 11 muestran que medida que se va aumentando el número de máquinas al que a asignar las tareas, el tiempo de ejecución y los nodos expandidos también aumentan. En este caso fuerza bruta es significativamente peor a todos los demás algoritmos.

A simple vista puede parecer que las gráficas de este experimento y el anterior son muy similares, la principal diferencia es la unidad del eje y, mientras que en el primer experimento era 10^4, en el segundo es 10^9, lo que representa un incremento de cinco órdenes de magnitud. Esta diferencia se debe a que el número de nodos expandidos en el algoritmo de fuerza bruta se calcula como m^n donde m el número de máquinas y n es el número de tareas. Por lo tanto, aumentar la base incrementa significativamente el número de nodos expandidos en comparación con un aumento en la potencia. Como se puede ver en la Tabla 4 con 10 máquinas, fuerza bruta puede llegar a expandir 1.13 mil millones de nodos.

4. Conslusiones

Los algoritmos de búsqueda unidireccional heurística juegan un papel crucial en la resolución de problemas complejos, especialmente en escenarios donde la cantidad de nodos a explorar puede ser exponencial. La eficiencia

de estos algoritmos no solo depende de su diseño intrínseco, sino que un paso fundamental es la selección de una heurística informada que guíe adecuadamente la búsqueda hacia la solución óptima. Una heurística bien diseñada no solo reduce el tiempo de ejecución, sino que también minimiza la cantidad de nodos expandidos, mejorando así el rendimiento general del algoritmo.

Además, la elección de estructuras de datos para implementar estos algoritmos es fundamental. La correcta selección y utilización de estructuras puede influir significativamente en la eficiencia del algoritmo, afectando tanto la velocidad de búsqueda como el uso de memoria. Optimizar estos aspectos es esencial para abordar problemas de gran escala de manera efectiva.

Otra posible estrategia para abordar el problema sería la implementación del algoritmo IDA* pero no se encontró una manera eficiente de resolver el problema de las trasposiciones en este contexto, lo que limitó la viabilidad de esta opción.

Por último, una forma de optimizar aún más la solución a estos problemas sería implementar el programa en un entorno distribuido o concurrente, lo que permitiría una mejor utilización de los recursos y una reducción del tiempo de procesamiento dado que el problema se puede dividir en problemas. Sin embargo, esta opción queda fuera del alcance de esta asignatura.