Car Sequencing Problem

Propuestas de solución utilizando algoritmos de búsqueda
GBJ y BackTracking

Integrantes: Daniel Martinez
Fernando Delgado
Fernanda Avendaño

Fundamentos del problema

Secuencia de vehículos en una línea de ensamblaje:

- Los vehículos cuentan con una lista de opciones a ser instaladas.
- Cada opción es instalada por una estación diferente cuya capacidad de operación es limitada.
- **Objetivo principal** → Encontrar un orden en la secuencia de vehículos que satisfaga la demanda para cada clase de coches y las restricciones de capacidad para cada estación de trabajo.

Parámetros del problema

- **V** = $\{v_1, v_2, \ldots, v_n\}$ → Lista de vehículos.
  - Vehículos con las mismas opciones a instalar pertenecen a una misma clase.
- **O** = $\{o_1, o_2, \ldots, o_n\}$ → Posibles opciones a instalar en un vehículo.
- $\boldsymbol{p_i / q_i}$ **: O → N.** Proporción que restringe la instalación de opciones.
  - Cada q_i vehículos consecutivos en la línea de ensamblaje, a lo más q_i de ellos podrán tener instalada la opción o_i

BackTracking como solución para CS → Propuesta 1

Parámetros: Número de clases → Demanda y opciones a instalar en cada una.

Variables y sus dominios:

- **Línea de ensamblaje:** Solución parcial actual de la instancia.
- **Clase actual:** Clase siendo agregada al final de la línea de ensamblaje.
- **Instanciados:** Cantidad de vehículos dentro de la línea.
- **No usar:** Lista de clases no utilizables en una dada posición de la línea.

Orden de instanciación: Secuencial, se intentará siempre agregar a la línea la menor clase que no se encuentre “vetada”, y que aún no se haya satisfecho su demanda.

Verificación de restricciones: Verificar últimos q vehículos en la línea luego de haber agregado un vehículo al final de esta.

BackTracking como solución para CS → Propuesta 1

Implementación realizada →

- Al **iniciar** el **programa**, la **clase actual** parte en **0** para iniciar siempre con la **menor clase**.
- **Pasos:**

1. Insertar la clase actual al final de la línea de ensamblaje.

2. Validar, según los últimos *q* vehículos en la línea de ensamblaje (O en toda la línea según sea el caso), si se cumple la proporción *p* / *q* para todas las opciones.

3. **Si se cumplen**, instanciar la clase actual como la menor clase con demanda sin cumplir y **volver al paso 1.**

4. **Si no se cumplen:** Vetar clase actual → Realizar **BackTracking**:
   a. Si en la posición actual de la línea de ensamblaje existen clases no vetadas con demanda que satisfacer, instanciar la clase actual con este valor y **volver al paso 1.**

   b. Si no existen, sacar vehículo anterior al último añadido a la línea, su valor será la nueva clase actual, y volver a repetir el bucle por si en su caso existe otra clase a utilizar.

BackTracking como solución para CS → Propuesta 1

Experimentación realizada:

- Se realiza una prueba sencilla para verificar el funcionamiento del programa.
- Las siguientes 4 instancias buscan medir rendimiento con una demanda constante y aumento de clases de vehículos.
- En todas ellas se utilizan la misma cantidad de opciones (5) y de proporciones *p* / *q.*
    - Las instalaciones realizadas en cada clase son cambiadas en cada instancia.

BackTracking como solución para CS → Propuesta 1

1er experimento:

```
10 5 6
1 2 1 2 1
2 3 3 5 5
0 1 1 0 1 1 0
1 1 0 0 0 1 0
2 2 0 1 0 0 1
3 2 0 1 0 1 0
4 2 1 0 1 0 0
5 2 1 1 0 0 0
```

```
Solucion posible: 0 1 5 2 4 3 3 4 2 5

Solucion posible: 0 2 5 1 4 3 2 4 3 5

Solucion posible: 0 2 5 1 5 3 4 2 3 4

Solucion posible: 4 3 2 4 3 5 1 5 2 0

Solucion posible: 5 2 4 3 3 4 2 5 1 0

Solucion posible: 5 3 4 2 3 4 1 5 2 0

Espacio de busqueda terminado
Tiempo de ejecucion: 0
Sobre calidad:
--- Cantidad maxima de vehiculos instanciados correctamente en algun punto: 10
--- Numero de iteraciones en total: 2265
--- Cantidad de soluciones encontradas: 6
```

BackTracking como solución para CS → Propuesta 1

2do experimento:

```
100 5 18
1 2 1 2 1
2 3 3 5 5
0 5 1 1 0 0 1
1 3 1 1 0 1 0
2 7 1 1 1 0 0
3 1 0 1 1 1 0
4 10 1 1 0 0 0
5 2 1 0 0 0 1
6 11 1 0 0 1 0
7 5 1 0 1 0 0
8 4 0 1 0 0 1
9 6 0 1 0 1 0
10 12 0 1 1 0 0
11 1 0 0 1 0 1
12 1 0 0 1 1 0
13 5 1 0 0 0 0
14 9 0 1 0 0 0
15 5 0 0 0 0 1
16 12 0 0 0 1 0
17 1 0 0 1 0 0
```

```
Solución 1:     0 3 6 14 2 15 1 9 7 14 0 12 1 14 7 8 1 16 2 14 5 9 2 16 4 8 6 10 4 16 0 9 7 14 4
                11 4 9 6 10 4 15 2 9 6 10 4 15 2 9 6 10 4 15 2 14 6 10 0 16 2 14 6 8 4 16 4 10 5
                14 4 16 6 8 7 14 6 10 0 16 7 16 13 10 6 15 6 10 13 16 10 13 16 10 13 16 10 13 16 17

Solución 2:     0 3 6 14 2 15 1 9 7 14 0 12 1 14 7 8 1 16 2 14 5 9 2 16 4 8 6 10 4 16 0 9 7 14 4
                11 4 9 6 10 4 15 2 9 6 10 4 15 2 9 6 10 4 15 2 14 6 10 0 16 2 14 6 8 4 16 4 10 5
                14 4 16 6 8 7 14 6 10 0 16 7 16 13 10 6 15 6 10 13 16 10 13 16 10 13 16 10 | 16 13 17

Solución 3:     0 3 6 14 2 15 1 9 7 14 0 12 1 14 7 8 1 16 2 14 5 9 2 16 4 8 6 10 4 16 0 9 7 14 4
                11 4 9 6 10 4 15 2 9 6 10 4 15 2 9 6 10 4 15 2 14 6 10 0 16 2 14 6 8 4 16 4 10 5
                14 4 16 6 8 7 14 6 10 0 16 7 16 13 10 6 15 6 10 13 16 10 13 16 10 13 16 | 17 13 16 10

Solución 4:     0 3 6 14 2 15 1 9 7 14 0 12 1 14 7 8 1 16 2 14 5 9 2 16 4 8 6 10 4 16 0 9 7 14 4
                11 4 9 6 10 4 15 2 9 6 10 4 15 2 9 6 10 4 15 2 14 6 10 0 16 2 14 6 8 4 16 4 10 5
                14 4 16 6 8 7 14 6 10 0 16 7 16 13 10 6 15 6 10 13 16 10 13 16 10 13 16 | 17 16 13 10

Solución 5:     0 3 6 14 2 15 1 9 7 14 0 12 1 14 7 8 1 16 2 14 5 9 2 16 4 8 6 10 4 16 0 9 7 14 4
                11 4 9 6 10 4 15 2 9 6 10 4 15 2 9 6 10 4 15 2 14 6 10 0 16 2 14 6 8 4 16 4 10 5
                14 4 16 6 8 7 14 6 10 0 16 7 16 13 10 6 15 6 10 13 16 10 13 16 10 | 16 13 10 13 16 17

Tiempo de ejecucion: 0 segundos.
Sobre calidad:
--- Cantidad maxima de vehiculos instanciados correctamente en algun punto: 100
--- Numero de iteraciones en total: 624
--- Cantidad de soluciones encontradas: 5
```

Iter 1 = 585 — Iter 2 = 589 — Iter 3 = 612 — Iter 4 — 616 — Iter 5 = 624

BackTracking como solución para CS → Propuesta 1

3er experimento:

```
200 5 19
1 2 1 2 1
2 3 3 5 5
0 5 0 0 0 0 1
1 7 0 0 0 1 1
2 1 0 0 1 0 1
3 67 0 1 0 0 0
4 7 0 1 0 1 1
5 15 0 1 1 0 0
6 1 0 1 1 1 1
7 26 1 0 0 0 0
8 6 1 0 0 0 1
9 11 1 0 0 1 0
10 5 1 0 0 1 1
11 9 1 0 1 0 0
12 6 1 0 1 1 0
13 1 1 0 1 1 1
14 2 1 1 0 1 1
15 20 1 1 1 0 0
16 3 1 1 1 0 1
17 6 1 1 1 1 0
18 2 1 1 1 1 1
```

```
Solucion sin errores no encontrada :(
Tiempo de ejecucion: 301
Sobre calidad:
--- Cantidad maxima de vehiculos instanciados correctamente en algun punto: 127
--- Numero de iteraciones en total: 559236770
--- Cantidad de soluciones encontradas: 0
```

BackTracking como solución para CS → Propuesta 1

4to experimento:

```
200 5 23
1 2 1 2 1
2 3 3 5 5
0 2 0 0 0 0 1
1 7 0 0 0 1 0
2 1 0 0 0 1 1
3 12 0 0 1 0 0
4 3 0 0 1 0 1
5 2 0 0 1 1 0
6 1 0 0 1 1 1
7 35 0 1 0 0 0
8 7 0 1 0 0 1
9 28 0 1 0 1 0
10 12 0 1 1 0 0
11 7 0 1 1 1 0
12 24 1 0 0 0 0
13 5 1 0 0 0 1
14 9 1 0 0 1 0
15 2 1 0 1 1 0
16 1 1 0 1 1 1
17 8 1 1 0 0 1
18 14 1 1 0 1 0
19 2 1 1 0 1 1
20 10 1 1 1 0 0
21 4 1 1 1 0 1
22 4 1 1 1 1 0
```

```
Solucion sin errores no encontrada :(
Tiempo de ejecucion: 301
Sobre calidad:
--- Cantidad maxima de vehiculos instanciados correctamente en algun punto: 139
--- Numero de iteraciones en total: 340678462
--- Cantidad de soluciones encontradas: 0
```

BackTracking como solución para CS → Propuesta 1

5to experimento:

```
200 5 25
1 2 1 2 1
2 3 3 5 5
0 2 0 0 0 0 1
1 9 0 0 0 1 0
2 4 0 0 0 1 1
3 8 0 0 1 0 0
4 1 0 0 1 0 1
5 6 0 0 1 1 0
6 1 0 0 1 1 1
7 40 0 1 0 0 0
8 5 0 1 0 0 1
9 5 0 1 0 1 1
10 16 0 1 1 0 0
11 1 0 1 1 0 1
12 10 0 1 1 1 0
13 2 0 1 1 1 1
14 22 1 0 0 0 0
15 4 1 0 0 0 1
16 9 1 0 0 1 0
17 1 1 0 1 1 1
18 26 1 1 0 0 0
19 5 1 1 0 0 1
20 4 1 1 0 1 1
21 13 1 1 1 0 0
22 1 1 1 1 0 1
23 4 1 1 1 1 0
24 1 1 1 1 1 1
```

```
Solucion sin errores no encontrada :(
Tiempo de ejecucion: 301
Sobre calidad:
--- Cantidad maxima de vehiculos instanciados correctamente en algun punto: 103
--- Numero de iteraciones en total: 422794940
--- Cantidad de soluciones encontradas: 0
```

BackTracking como solución para CS → Propuesta 2

Representación: S_j = clase de auto k asignado al slot j.

El orden de instanciación de las clases corresponde al orden el en el cual se encuentran las clases en el archivo de entrada.

Movimiento: Cambiar la clase del último slot instanciado que no trasgreda la restricción de demanda de esa clase de acuerdo al orden de instanciación.

Algoritmo

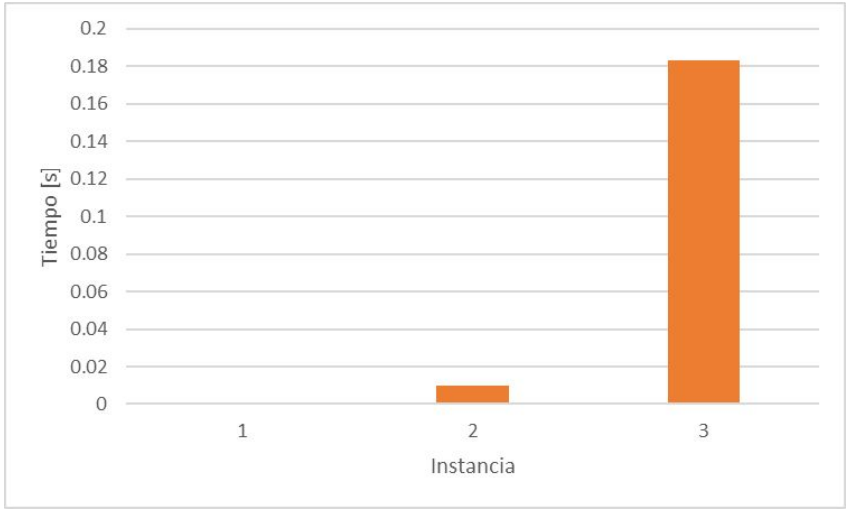
Algorithm 1 Backtracking solver

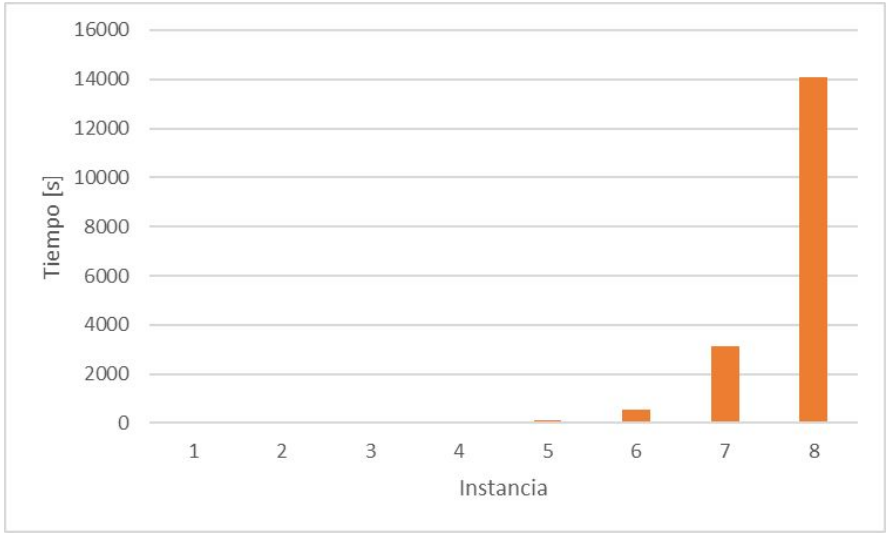
```
1: function BT_SOLVER(S, D, P, Q, profundidad, nAutos, nClases, nOpciones)
2:     if profundidad = nAutos then
3:         nViolations ← calculate_violations(S, P, Q, nAutos, nOpciones)
4:         if nViolations = 0 then
5:             solution ← S
6:             end execution
7:         end if
8:     else
9:         for i ← 0; i < nClases; i ← i + 1 do
10:            if D[i] ≠ 0 then
11:                D[i] ← D[i] − 1
12:                S[profundidad] ← i
13:                BT_SOLVER(S, D, P, Q, profundidad + 1, nAutos, nClases, nOpciones)
14:                D[i] ← D[i] + 1
15:            end if
16:        end for
17:    end if
18: end function
```

Experimentación

Instancia	Demanda total	Cantidad de clases	Cantidad opciones	Tiempo [s]
1	10	6	5	0.001
2	11	6	5	0.010
3	12	6	5	0.180
4	13	6	5	3.150
5	14	6	5	101.590
6	15	6	5	544.650
7	16	6	5	3122.750
8	17	6	5	14064.610

Resultados


0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
Tiempo [s]
1
2
3
Instancia


16000
14000
12000
10000
8000
6000
4000
2000
0
Tiempo [s]
1
2
3
4
5
6
7
8
Instancia

Conflict Directed Backjumping como solución para CS.

Algorithm 1 CBJ

Require: *target*: Solución (vacía), Matriz, cantAutos, cantOpciones, cantClases, cantAutosPorClase

Ensure: Solución

1: $i = 1$
2: $D'_i = D_i$
3: $J_i =$
4: **for** $i = 1, i = n, i++$ **do**
5: $instanciax_i = SelectValueCBJ$
6: **if** $x_i == null$ **then**
7: $iprev = i$
8: $i =$ index de la ultima variable en set conflictos J_i
9: $J_i = J_i \cup J_{iprev} - x_i$
10: **if** $x_i != null$ **then**
11: $i = i + 1$
12: $D'_i = D_i$
13: $J_i =$
14: **if** $i = 0$ **then**
15: return inconsistente
16: **if** $i != 0$ **then**
17: return valores $x_1 ... x_n$

Implementación realizada

1. Se asocia a cada variable X_i su conjunto conflicto, que contiene las variables pasadas para las cuales se ha detectado alguna inconsistencia con X_i .
2. Cuando se detecta una inconsistencia en una instanciación a_i de la variable actual Xi y una instanciación de alguna variable pasada, se añade esta última al conjunto conflicto de X_i .
3. Cuando se ha agotado el dominio de X_i , se retrocede a la variable más profunda del conjunto de conflicto de X_i.
4. En cada invocación de CBJ, el conjunto conflicto de X_i se vacía. Si CBJ encuentra una solución, se añade al conjunto conflicto de X_n la variable X_{n-1} para garantizar que CBJ retrocederá al nivel n-1 cuando se hayan explorado todos los valores del dominio.

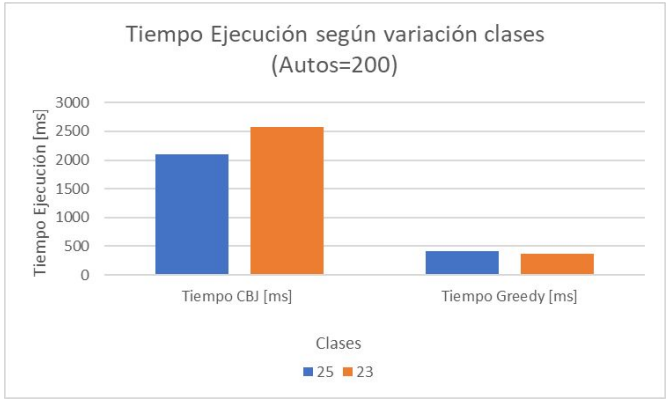
Experimentos

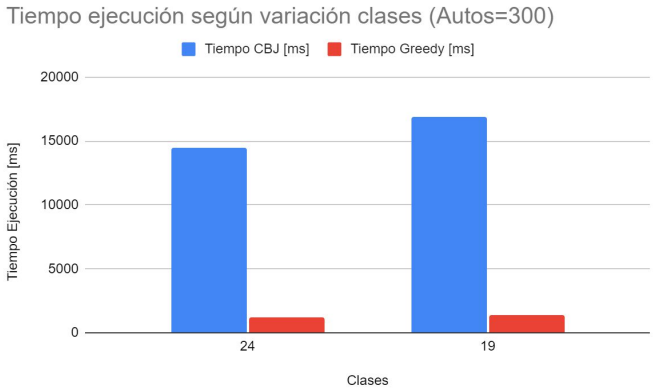
Instancia	Cant. Autos	Clases	Opciones	Tiempo CBJ [ms]	Tiempo Greedy [ms]
1	200	25	5	2107	421
2	200	23	5	2572	368
3	300	24	5	14432	1200
4	300	19	5	16924	1358
5	400	23	5	46885	3293
6	400	20	5	55262	2761

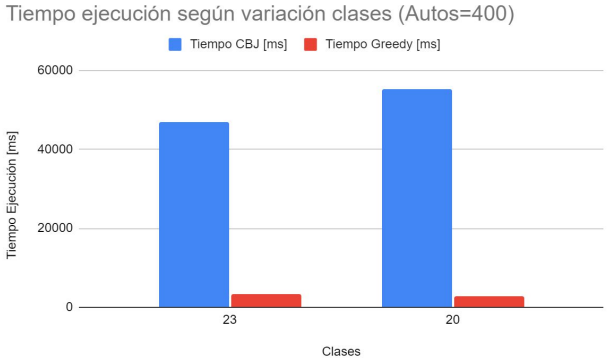
- Se trato de elegir la cantidad más variada de información con respecto al número de clases, opciones y cantidad de autos.
- Se hizo una comparación con el algoritmo Greedy.

Experimentos

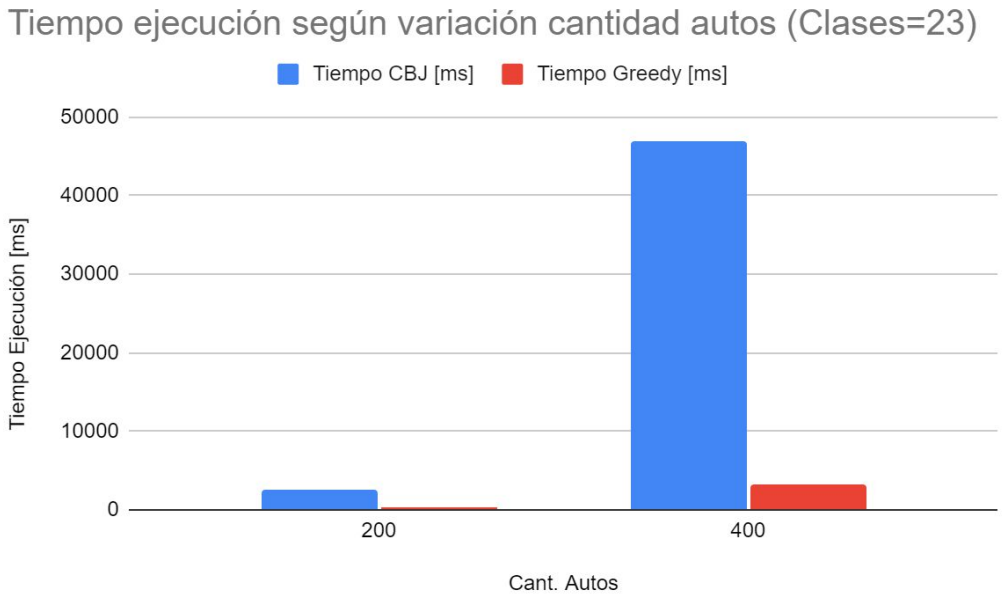
Experimentos







Experimentos



Conclusiones enfoque CBJ para CS

- A medida que aumenta la cantidad de vehículos se puede notar que utilizar CBJ toma un tiempo considerable en comparación al enfoque Greedy.
- El enfoque greedy y combinándolo con una cantidad media de aleatorismo demuestra que puede encontrar rápidamente soluciones óptimas locales para todas las instancias de prueba.
- La desventaja más significativa encontrada en el algoritmo CBJ es la cantidad de tiempo que toma debido a la comprobación de las restricciones.
- Del análisis realizado, siempre se considera que la secuencia tiene un principio y final, y que se conoce la cantidad existente vehículos a secuenciar. Por lo mismo, un buen trabajo a futuro sería desarrollar una metodología que permita resolver el problema con una mejor aproximación a la situación real. Además de poder integrar prioridades en la secuencia de ensamblaje y también la limitante de las cantidades de productos (pintura, vidrio, etc) disponibles.

Conclusiones al utilizar BT (Propuesta 1) para CS:

Ventajas:

- **Permite visualizar todas las soluciones factibles** del problema dado que recorre todo su espacio de búsqueda.
- **Sencillo de implementar.**

Desventajas:

- Es bastante **ineficiente** dado que es un método de **búsqueda exhaustiva**, y **CS es NP-Difícil.**
  - A medida que aumenta la complejidad de la instancia (En este caso aumento de demandas y clases) se tarda demasiado en buscar una “posible” solución.
- Realiza **trabajo redundante** → Incluso si se detecta un conflicto entre variables al ser agregadas a la línea, no es recordado para detectarlo en las siguientes iteraciones.

Posibles arreglos: Utilizar heurísticas para el orden de selección de clases, como podría ser el intentar siempre agregar primero aquellas clases con mayores opciones y luego de estas aquellas con menores para permitir un orden correcto.

Conclusiones al utilizar BT para CS: Propuesta 2

- Es fácil de implementar
- Usar backtracking no es la mejor aproximación para tratar de resolver el CS, debido a que recorrer todo el espacio de búsqueda de este problema para instancias de más de 20 autos es difícil.
- Como trabajo a futuro sería prometedor usar backtracking junto con heurísticas greedy para así reducir el espacio de búsqueda del algoritmo.