Informe del Trabajo Práctico n 1: Búsqueda y optimización

Agüero, Emanuel Bustillo, Carlos Lezcano, Agustín

25 de abril de 2020

1. Elección de algoritmos como solución a distintos tipos de problemas

1.1. Diseño de un proceso de manufactura

La selección de un proceso de manufactura es un problema de satisfacción de restricciones, por lo tanto se no buscará retener la "historia"del proceso si no los resultados puntuales, entonces se puede utilizar un algoritmo de búsqueda local, que aunque puede no ser óptimo se utiliza mucho en la industria porque da una buena aproximación a la solución. Para este proceso es común buscar minimizar el tiempo de fabricación total mediante por ejemplo el ordenamiento de las piezas a fabricar según su tiempo de demora, tratando de paralelizar la mayor cantidad de piezas posibles.

Como variable se pueden usar las tareas dadas para la fabricación de cada pieza y como valores posibles el orden de manufactura de las mismas. Como restricción puedo definir por ejemplo que en un mismo tiempo no se fabriquen dos tipos diferentes de piezas porque ambas usan la misma máquina, o que para la fabricación de una pieza primero debe fabricarse otra. Se pueden usar algoritmos genéticos para su resolución

1.2. Planificación de órdenes de fabricación

Al ser un problema de planificación puedo tener en cuenta distintos factores como por ejemplo el tiempo de demora para la fabricación de determinado producto o la prioridad asignada según el cliente, etc. Para este tipo de problema se puede usar un algoritmo de búsqueda local como por ejemplo temple simulado, estableciendo como energía el tiempo total de la orden de fabricación.

1.3. Ubicación óptima de aerogeneradores

Este problema de optimización tiene en cuenta varios factores, como por ejemplo la influencia del viento, la distancia entre los aerogeneradores, el te-

rreno, etc. Se puede utilizar el concepto de algoritmos genéticos para resolver este problema. Se buscará obtener el mejor resultado posible teniendo en cuenta los factores previamente nombrados, por lo tanto el sistema se puede modelar usando como gen a cada uno de estos valores y mediante una función idoneidad evaluar su calidad para cada uno de los aerogeneradores, que serían la población en cuestión. Mediante el algoritmo se puede evaluar la mejor disposición posible para cada uno de los generadores, buscando un criterio de convergencia conveniente o simplemente cuando transcurra cierto tiempo, dependiendo del criterio de parada previamente elegido.

1.4. Planificación de trayectorias de un brazo robotizado con 6 grados de libertad

Dependiendo del tipo de planificación de las trayectorias, podría usar un algoritmo de búsqueda informada como el caso del algoritmo A* o un algoritmo de búsqueda local como algoritmos genéticos. En el primer caso se puede obtener una solución óptima, el segundo caso es posible usarlo en el caso de planificación en tiempo real ya que no requiere tanta memoria al no expandir el árbol completo.

Para la resolución del problema usando el algoritmo A* se puede usar el espacio articular, asignándole a cada grado de libertad un número determinado de movimientos posibles, a mayor cantidad de movimientos posibles incrementa la precisión (cada movimiento es de menos grados) pero a su vez esto aumenta la cantidad de nodos aumentando así el costo computacional.

1.5. Diseño de un generador

Para el diseño de un generador se tienen en cuenta varias características, como por ejemplo la cantidad de polos del motor, el factor de potencia del generador, la clase de sistema de aislamiento, el amperaje y tensión máximas, el tamaño del motor, etc. Se busca optimizar los recursos y funciones del generador. Cada una de estas características puede ser modelada como un gen determinado, entonces una solución posible es usar algoritmos genéticos.

Siguiendo lo dicho anteriormente, se puede tipificar cada característica tomándola como un gen del genoma Generador. La función de idoneidad que se puede tomar depende del objetivo que se busca, como por ejemplo el precio o el tamaño óptimo del generador con las mejores prestaciones posibles.

1.6. Definición de una secuencia de ensamblado óptima

Definir una secuencia de ensamblado óptima es un problema de optimización, o sea mejorar la configuración. Para tal fin se puede usar un algoritmo de Recocido Simulado, usando como energía el tiempo que insume cada posible secuencia, tomando como estado inicial una secuencia aleatoria.

1.7. Planificación del proyecto de una obra

Al involucrar tiempos de duración de cada tarea, cierto orden que se debe respetar (no se puede empezar desde el final) es un problema de planificación. Para tal objetivo se puede usar algún algoritmo de búsqueda hacia atrás (la búsqueda hacia adelante es inviable), podríamos usar A* para conseguir una heurística admisible.

2. Aplicación del algoritmo A* para la obtención de un camino óptimo

2.1. Cálculo de la trayectoria óptima de un brazo robótico de 6 grados de libertad

Para la trayectoria del brazo robótico se usa una función llamada *generate map*, que admite un parámetro (*tam*: cantidad de puntos por cada dirección), genera un mapa de 6D con esa cantidad de puntos por cada grado de libertad. Luego se generan los obstáculos de forma aleatoria, se definió arbitrariamente que el 10 % del espacio articular esté ocupado por obstáculos y la cantidad de estos se calcula como

$$\frac{cant.depuntos^6}{10} \tag{1}$$

Con la figura 1 se puede deducir que a medida que aumenta la cantidad de puntos, el tiempo de ejecución aumenta en gran medida. Se han adoptado 15 puntos para que la ejecución del código sea en un tiempo considerablemente corto para la complejidad del problema, aún así podrían adoptarse muchos más teniendo en cuenta el costo computacional que conlleva. Como se indica en la figura 2, el código se ha programado orientado a objetos, de tal forma que existe una clase llamada *Nodo*, la cual en el momento de generarse un determinado nodo, se inicializan los atributos *g, h, f* a 0, mientras que *padre* y *pos* a una variable del tipo *None* por defecto, a menos que se pasen parámetros para que adopten éstos. Además, la clase contiene métodos para el calculo de sus atributos *g, h* y *f*. La heurística H se calcula con la distancia de Manhattan: la suma de la diferencia entre iguales componentes de dos vectores posición dentro del espacio articular de 6D; el costo G se calcula como el camino recorrido hasta la posición actual +1 y el costo F se calcula como la suma de G y H.

El estado inicial es la posición inicial del brazo robótico. Se generaron posiciones iniciales y finales entre 0 y la cantidad de puntos menos 1 y luego se aplicó el algoritmo A*, en el que se hace uso de un método utilizando una llamada *lista abierta* y *lista cerrada*. En la lista abierta serán guardados los nodos abiertos pero no explorados (vecinos del nodo actual) para luego ser eliminados de ésta y pasar a la lista cerrada (nodos explorados). Se guardan los vecinos de la posición actual en la variable *neighbours*, y éstos se pueden entender como dirigirse desde la posición actual hacia cada grado de libertad en las dos direcciones (hacia atrás y hacia adelante), con un solo desplazamiento por vecino,

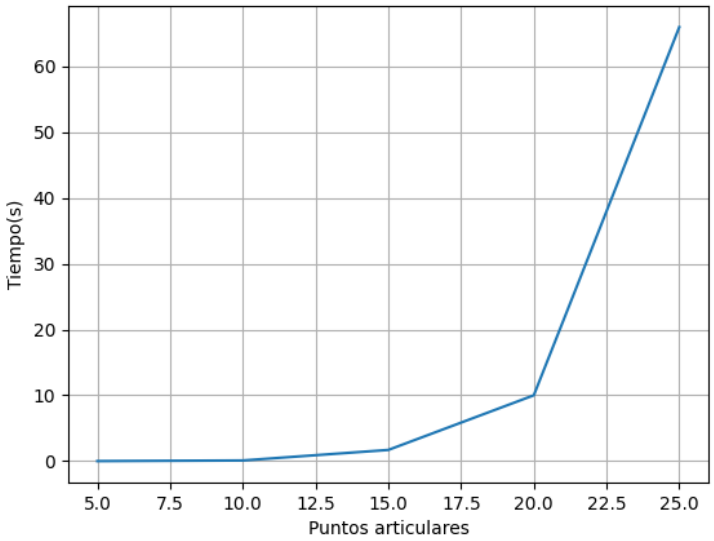


Figura 1: Tiempo de cálculo respecto a la cantidad de puntos de cada GDL

es decir que solo habrá una componente distinta de cero. Luego a través de un condicional se evalúa si la posición está en los obstáculos, no se lo agrega como vecino de la posición actual. Se guarda el camino en la variable *path* que recorre la lista cerrada desde nodo final al nodo inicial de padre en padre, para finalmente devolver el mismo e imprimirlo. La comprobación de si el nodo actual es igual al nodo objetivo (teniendo en cuenta las posiciones angulares) es la prueba de meta.

2.2. Obtención de un camino óptimo dentro de un almacén

En primera instancia se crea el mapa del almacén mediante una función que admite tres parámetros: el orden del almacén (números del 1 al 48 de manera ordenada, ya creados por defecto aunque se puede modificar), la cantidad de filas y columnas de las estanterías, que en este caso están dadas según la consigna pero pueden ser modificadas. La función puede crearse de manera más dinámica dependiendo de lo que se necesite, ya sea un almacén de mayores medidas o de un estado especifico de orden de los productos dentro de éste.

Para la realización de este algoritmo se utilizó el método explicado anteriormente de lista abierta-lista cerrada llamados *OPEN* y *CLOSED* respectivamente. Se trabajó con clases, siendo los atributos de cada nodo sus costes g, la heurística h y el valor f y un apuntador a su padre; como métodos se usan

```
class Nodo:
    def __init__(self,padre=None,pos=None):
        self.padre = padre
        self.pos = pos
        self.g = 0
        self.h = 0
        self.f = 0
    def calculate_h(self,end):
        aux = 0
        for i in range(6): aux += abs(self.pos[i]-end[i])
        self.h = aux
    def calculate_g(self):
        self.g += 1
    def calculate_f(self):
        self.f = self.g+self.h
```

Figura 2: Código

funciones con las cuales se calculan f, g y h. Se comprueba en cada nodo el costo

$$f(n) = g(n) + h(n) \tag{2}$$

y se avanza expandiendo el nodo hijo de menor costo. Si para un nodo ubicado en la lista cerrada se comprueba que el costo de la ruta a través del nodo explorado es menor al del costo que tenía anteriormente, se lo agrega a la lista abierta.

La heurística utilizada en este algoritmo es similar a la distancia de Manhattan pero elevada al cuadrado, esto es, la distancia teniendo en cuenta solo movimientos verticales y horizontales en cuanto a la posición del nodo actual elevada al cuadrado.

Para localizar las posiciones entre las cuales realizar la búsqueda se usa la función *search_position_of*, la cual admite un número como parámetro, *valor*, y el mapa a usar para buscar éste; dependiendo de si el valor se encuentra a izquierda o derecha del pasillo central, devuelve el valor de la columna anterior o siguiente respectivamente. El problema encontrado con este diseño fue al buscar el numero 0, adoptado como "Picking Point.[en] ejercicios subsiguientes, para lo cual la función devolvía (0,-1) porque el primer 0 que encuentra es el de posición (0,0); este problema se pudo solucionar agregando un condicional a esta función para retornar la posición (0,0) siempre que se busque un 0 o algún numero que no esté dentro del mapa.

Cada vez que el nodo avanza se guarda el camino generado por el mismo de la siguiente manera: luego de llegar al nodo objetivo, se recorre ese nodo en sentido inverso a través del padre de cada nodo hasta llegar al nodo raíz.

3. Aplicación de Temple Simulado para la obtención del orden óptimo de picking de un almacén

El objetivo de este problema es obtener el orden óptimo para la operación de *picking* en un almacén, utilizando el método de Temple simulado.

Elementos del algoritmo

- Algoritmo: A continuación se hace mención de los parámetros que intervienen en el algoritmo:
- Estado inicial: En este caso el estado inicial es una lista que contiene las posiciones en donde se debe realizar el picking (las posiciones por las que debo pasar). Estas están ubicadas de manera aleatoria en la lista.
- Temperatura inicial: Es el estado inicial con el que se cuenta, es la lista de elementos a seleccionar en un determinado orden; el mismo se da *harcodeando* en el código las posiciones e ingresando las en una lista (por ejemplo: [37,47,11,48,24,15,2]).
- Temperatura final: Es el valor final obtenido luego de realizar el algoritmo. *Nota:* El estado final obtenido puede no ser el mejor, por lo tanto se guarda el menor estado obtenido cada vez que se actualiza el estado.
- Perfil de descenso de la temperatura: indica la razón a la cual va a disminuir la temperatura en cada iteración. La variación de este perfil condiciona la obtención de los resultados. Se ha experimentado con el descenso lineal de la temperatura, es decir decrementar el valor de la misma en una cantidad fija al transcurrir cada iteración. También se ha experimentado con el descenso .exponencial"de la temperatura, usando una constante entre 0 y 1 para multiplicar por el valor de T en cada iteración.
- Energía: como valor de energía se ha usado el algoritmo A* utilizado en el problema anterior para poder calcular la distancia óptima entre cada posición de la orden de productos a despachar. La suma de cada una de las distancias entre coordenadas consecutivas en la orden da como resultado la energía del estado actual (cada estado es una lista con distinto orden para realizar el picking).

Para buscar estados vecinos simplemente se toma una posición al azar de la lista y intercambia de lugar el pedido de esa posición con otro pedido de la lista. Luego de calcular la energía para el mismo se procede a aceptar o no el estado sucesor como estado actual; para ello se comparan las energías de ambos estados, si el estado sucesor tiene una energía mayor (en este caso una energía mayor es una menor distancia recorrida para la misma orden) se lo acepta como

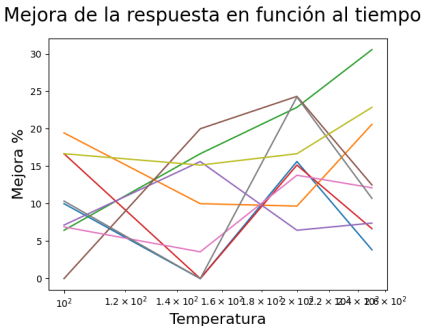
nodo actual, por el contrario si es menor se acepta como estado actual solo con una probabilidad de exp(-delta/T).

3.1. *Tunning* de parámetros

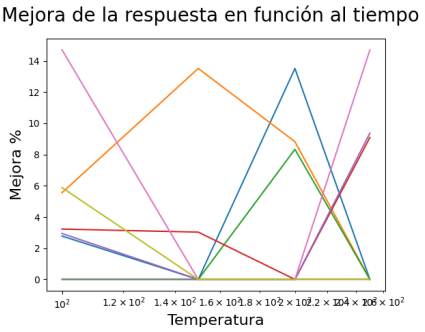
Nota: el algoritmo utilizado en la carpeta principal, si bien devuelve estados iniciales mejores que los iniciales en su mayoría, a veces carece de consistencia en cuanto a la mejora de los resultados obtenidos (los gráficos de la mejora de la relación Estado final-Estado inicial), por lo tanto se optó por considerar un segundo algoritmo de Temple Simulado (adjuntado en la carpeta .ªuxiliares") el cual si presenta mayor consistencia en cuanto a la respuesta aunque el tiempo de ejecución es mayor.

Se han determinado los parámetros por medio de prueba y error, variando las temperaturas iniciales desde T=100 hasta T=1000 o T=2500, dependiendo del caso. También se analizó el tipo de decrecimiento de la temperatura. Se consideraron los casos de decrecimiento lineal (disminuyendo el valor de la temperatura de 1 en 1), exponencial (cada vez que se actualiza la temperatura lo hace con el valor T/2), o también multiplicada por el valor 0,92 (valor arbitrario) cada vez que se actualiza el valor de la variable.

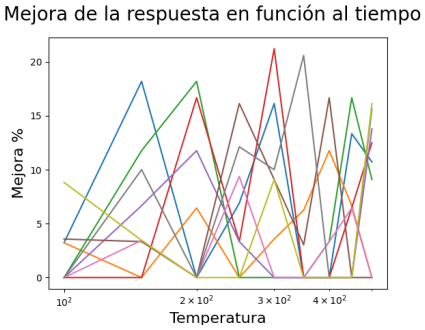
En base a los resultados obtenidos anteriormente, se ha llegado a la conclusión de que para el modelo asignado los parámetros que mejor ajustan son los de Temperatura inicial 250 (T=250) y decrecimiento lineal, ya que al asignar un valor máximo mayor a 250 no se obtienen mejores resultados en general y el tiempo de cómputo es mayor.



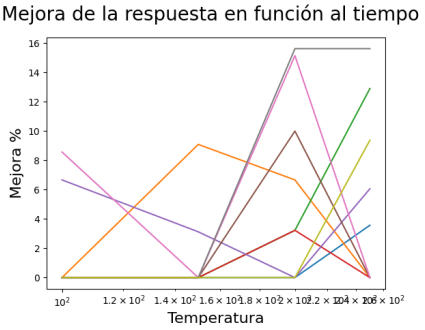
(a) Decrecimiento lineal: de 100 a 250



(b) Decrecimiento exponencial: de 100 a 250



(c) Decrecimiento con factor de 0,92 con T=100 hasta T=500



(d) Decrecimiento con factor 0,92 con T=100 hasta T=250

4. Algoritmo Genético

Se implementó un algoritmo genético con el fin de resolver el problema de optimizar la ubicación de los productos del almacén para optimizar el picking de los mismos. Para resolver este algoritmo consideramos:

- Individuo: almacén con cierta disposición de productos.
- Genes: disposición de los 48 productos.
- Fitness: distancia calculada con recocido simulado.

El algoritmo fue programado orientado a objetos, haciendo uso de una clase que fue llamada *Individuo*(Figura 4), que contiene de atributos los genes y el fitness de un determinado individuo. La función *algoritmo_genetico* comienza ejecutando *crear_lista_picking* que crea una lista de 5 listas de picking, que simulan órdenes ficticias a satisfacer. Cada una de éstas tiene longitud variable de 4 a 8 productos, que serán elegidos al azar y no podrán repetirse 2 productos en una lista. Se inicializa la población con 40 individuos en la función

crear_primer_poblacion, donde se crea la lista *poblacion* que contiene individuos de la clase *Individuo* generados aleatoriamente.

```
class Individuo():
    def __init__(self,genes=[],fitness=0):
        self.genes = genes
        self.fitness = fitness
```

Figura 4: Clase individuo

Una vez generada la primer población, se entra en un bucle en el que se va a comenzar con la población inicial de individuos para ir pasando por los procesos de selección, evolución y mutación hasta cumplir con el método de parada.

4.1. Selección

El criterio utilizado es el de selección de los k mejores, siendo

$$k = 0{,}2 * n \tag{3}$$

(n: tamaño de la población). Se adopta un número tan reducido debido a que si fuese cercano al tamaño de la población, podría evolucionar lentamente ya que prácticamente todos los individuos de la población actual intervendrán en la siguiente generación. En el programa se calcula el fitness de cada individuo de la población para cada lista de picking, sumándolos para generar un fitness total que es guardado como fitness del individuo en cuestión. Luego se calculan el número de individuos seleccionados, adoptando un número entero y par para que pueda realizarse de forma correcta. Finalmente se calculan los mejores individuos comparando sus valores de fitness entre sí: se comienza con un valor semilla del fitness del primer individuo, y quien tenga menor fitness que éste (en este caso, ya que recorrió menos camino de acuerdo a cierta disposición) será seleccionado a menos que se encuentre uno mejor. El proceso se realiza de forma iterativa hasta que se complete la cantidad de seleccionados.

4.2. Crossover

Se utilizó crossover por cruce de orden debido a que es un tipo de permutación en la que los genes se representan con variables discretas y deben aparecer una sola vez en cada individuo. Para esto, se seleccionan entre pares a los individuos seleccionados, se generan dos números aleatorios entre 1 y 46 (inclusive) para realizar los cortes, de forma que se asegure que existe al menos un gen en cada extremo que haya quedado fuera del corte (recordar que cada individuo tiene 48 genes, por lo que el segundo índice será 1 y el penúltimo, 46). Luego se crean dos nuevas listas llamadas *newA* y *newB* que guardarán las nuevas listas. En éstas se copian los genes entre los puntos de cruce del padre opuesto y finalmente, de acuerdo al método, se copian los valores restantes a partir del segundo corte, en orden y evitando duplicar valores.

4.3. Mutación

Un individuo puede llegar a mutar con una probabilidad del 15 %. Para esto, por cada individuo se calcula un número al azar entre 0 y 1, y se lo compara con 0.15: en caso de ser menor el individuo muta, realizando un intercambio entre 2 genes al azar.

4.4. Método de parada

Se adoptó como criterio de parada al tiempo transcurrido, es decir, una vez que se alcanzó un número determinado de generaciones, el bucle termina y el mejor individuo es seleccionado de la población de la última generación.

4.5. Selección del mejor

De la última población, se selecciona el individuo de menor fitness, se lo adopta como el mejor de ésta y se retorna para luego imprimirlo junto con el mapa.

```
Ejercicio 6
Ordenes ficticias:
[35, 36, 38, 37, 9]
[28, 48, 30, 2]
[16, 20, 24, 42, 39]
[25, 8, 39, 17]
[39, 7, 26, 9, 32]
El mejor individuo de la poblacion es:
 [48, 20, 32, 14, 46, 2, 13, 43, 11, 39, 17, 23, 3, 16, 36, 38, 41, 10,
12, 28, 15, 27, 4, 25, 21, 6, 8, 18, 33, 26, 34, 37, 24, 29, 42, 7, 22,
44, 40, 35, 9, 47, 30, 45, 19, 31, 1, 5]
Almacen con el mejor individuo:
 [[ 0  0  0  0  0  0  0  0]
 [ 0 48 20  0  0 21  6  0]
 [ 0 32 14  0  0  8 18  0]
 [ 0 46  2  0  0 33 26  0]
 [ 0 13 43  0  0 34 37  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0 11 39  0  0 24 29  0]
 [ 0 17 23  0  0 42  7  0]
 [ 0  3 16  0  0 22 44  0]
 [ 0 36 38  0  0 40 35  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0 41 10  0  0  9 47  0]
 [ 0 12 28  0  0 30 45  0]
 [ 0 15 27  0  0 19 31  0]
 [ 0  4 25  0  0  1  5  0]
 [ 0  0  0  0  0  0  0  0]]
Tiempo de ejecucion: 1.65 minutos
```

Figura 5: Respuesta de algoritmo genético

5. Algoritmo de satisfacción de Restricciones para un problema de *scheduling*

Se planteó un schedule imaginario del proceso de fabricación de un equipo formado por 3 piezas, las cuales deben pasar por una serie de operaciones para finalmente acoplarlas. Se cuenta con máquinas como ser: 1 torno, 1 equipo para pintar, 1 pulidora, 1 horno y 1 área de ensamblaje. El objetivo del CSR es determinar el período de inicio de cada tarea de manera de no exceder la capacidad de la máquina y respetar el deadline (250 horas).

# de tarea	Máquina	Operación	Tiempo de duración [min]
0	Torno	Mecanizar pieza 1	40
1	Equipo para pintar	Pintado 1 de pieza 1	20
2	Equipo para pintar	Pintado 2 de pieza 1	25
3	Pulidora	Pulir pieza 1	10
4	Torno	Mecanizar Pieza 2	25
5	Equipo para pintar	Pintado 1 de Pieza 2	20
6	Horno	Temple Pieza 3	50
7	Torno	Mecanizar Pieza 3	30
8	Pulidora	Pulir Pieza 3	10
9	Ensamblaje	Ensamblado final	15

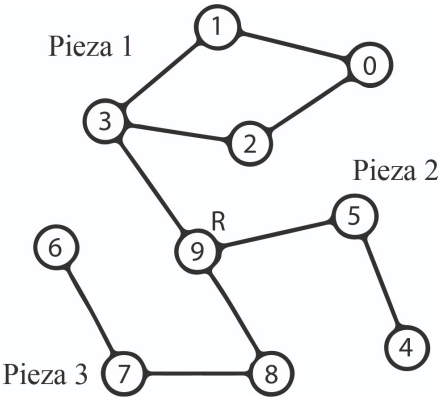
Nota: Se consideró un período de 5 minutos, por eso el tiempo de duración de cada operación se expreso como múltiplo de 5.

Se definieron las restricciones temporales, donde debe realizarse una determinada tarea antes que otras, también esta la restricción del deadline la cual se debe cumplir y las restricciones de las disponibilidad de las máquinas, las cuales solo pueden realizar una tarea a la vez. De la tabla anterior tenemos restricciones globales (más de 2 variables) por ejemplo para el uso del torno, hay restricciones binarias como en el caso del uso de la pulidora y restricciones unarias como el caso del área de ensamblaje y el horno. Restricciones:

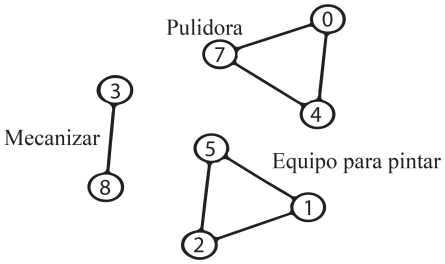
R0	tarea[0] + d[0] <=tarea[1]	R9	tarea[6] + d[6] <= tarea[7]
R1	tarea[0] + d[0] <=tarea[2]	R10	tarea[0] + d[0] <=tarea[4] or tarea[4] + d[4] <=tarea[0]
R2	tarea[1] + d[1] <=tarea[3]	R11	tarea[4] + d[4] <=tarea[7] or tarea[7] + d[7] <=tarea[4]
R3	tarea[2] + d[2] <=tarea[3]	R12	tarea[0] + d[0] <=tarea[7] or tarea[7] + d[7] <=tarea[0]
R4	tarea[4] + d[4] <=tarea[5]	R13	tarea[1] + d[1] <=tarea[2] or tarea[2] + d[2] <=tarea[1]
R5	tarea[7] + d[7] <=tarea[8]	R14	tarea[1] + d[1] <=tarea[5] or tarea[5] + d[5] <=tarea[1]
R6	tarea[3] + d[3] <=tarea[9]	R15	tarea[1] + d[2] <=tarea[5] or tarea[5] + d[5] <=tarea[2]
R7	tarea[5] + d[5] <=tarea[9]	R16	tarea[3] + d[3] <=tarea[8] or tarea[8] + d[8] <=tarea[3]
R8	tarea[8] + d[8] <=tarea[9]	R17	Tarea[9] <= deadline

Nota: Tarea[i] representa el instante de inicio de cada operación i, mientras que d[i] representa la duración respectiva de la misma

El criterio para este algoritmo fue asignar valores primero a la tarea[0], si ese valor se encuentra dentro del dominio prosigue a asignarle un valor a la tarea[1] y así sucesivamente hasta llegar a la tarea[9]. La forma de asignar los períodos para cada intervalo es de a 5 minutos, como se mencionó anteriormente, se empieza



(a) Grafo con la representación de las tareas desde el punto de vista temporal de las piezas []



(b) Grafo con la disponibilidad de la máquinas y la tarea que la requiere

a asignar desde 0 (valor menos restrictivo), luego 5, luego 10, etc. En caso de no verificar con alguna de las condiciones realiza backtracking para probar una nueva combinación y vuelve a realiza una recursión. Para este ejercicio es una búsqueda global, que recorre de forma exhaustiva el grafo, sin ninguna optimización, entonces la solución que se obtiene es la primera que encuentra (no necesariamente es la óptima),de otra forma si no se encuentra ninguna solución luego de recorrer todo el grafo recursivamente, muestra “No se ha encontrado una solución con ese deadline”.

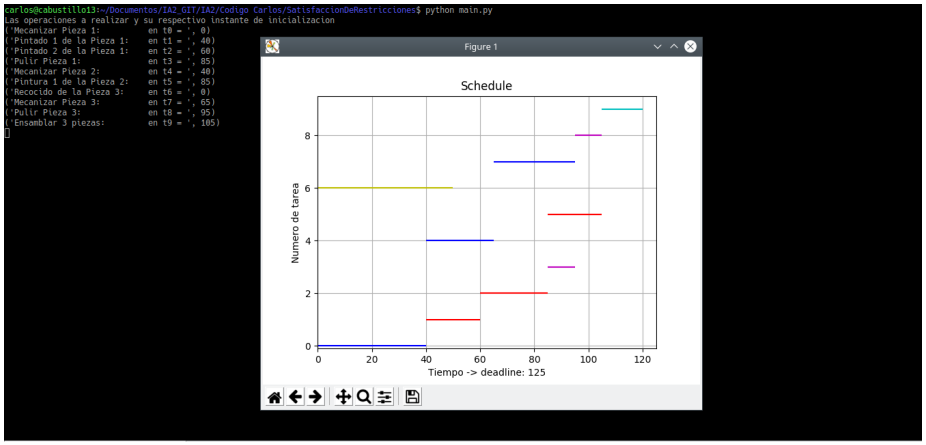


Figura 7: Resolución del algoritmo