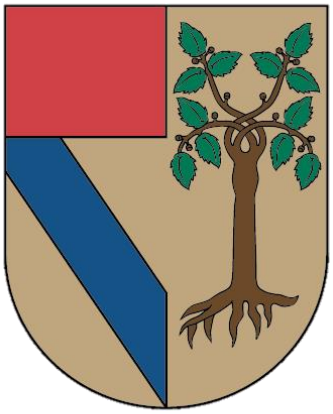
RECOCIDO SIMULADO EN PRODUCCIÓN

Optimización y Metaheurísticas I



25 DE OCTUBRE DE 2021

Alumno:

0227221 – Alan Samuel Aguirre Salazar

ÍNDICE

Introducción

En el presente trabajo se explicará cómo se resolvieron las dos instancias dedicadas a minimizar el máximo tiempo que efectúan ‘n’ máquinas para ‘k’ operaciones. Se dará a conocer la idea de donde surgió dicho algoritmo, la explicación de su funcionamiento (el antes del recocido simulado y la función generadora de perturbaciones), resultados y algunos análisis y conclusiones respecto a estos problemas.

Idea de donde surgió el algoritmo

Queremos adaptar el Algoritmo de Recocido Simulado (SA) para resolver el problema de la línea de producción. Como muchos de los algoritmos de optimización, hay tres elementos del SA que tenemos que definir antes de pasar a la codificación:

1. Estado

El estado es el conjunto de condiciones que satisfacen el problema y de los cuales vamos a sacar aquel que nos de la optimización que buscamos. En el algoritmo SA clásico, los estados son las posiciones para las que la función objetivo está definida. Para este problema, los estados son el conjunto de máquinas y el orden en el que procesan los elementos que les corresponde.

2. Transición

La transición es la forma en la que pasamos de un estado a otro en el proceso de búsqueda. Típicamente, estas transiciones son no deterministas, como cuando tomamos un vector aleatorio en la versión elemental del SA. En este caso, para transitar de un estado a otro lo que haremos será tomar una orden aleatoria asignada a una máquina aleatoria, y cambiarla a otra máquina aleatoria. (Por ejemplo, si la orden 2 la tenemos para ser procesada por la máquina 4, cambiarla a la máquina 1).

3. Evaluar

Una vez que definimos el estado y su cambio, debemos de definir la forma en la que vamos a evaluar el estado para saber si es más o menos apropiado que otros por los que hemos pasado. Cuando tenemos una función basta con obtener el valor de la función para las variables dadas. En este caso, vamos a sumar los tiempos de los órdenes de cada máquina, así como sus respectivos tiempos de ajuste, y vamos a sacar el mayor de cada uno. Recordemos que se nos pide obtener el mínimo

tiempo de la máquina que se tarda más. Aquí también vale la pena recordar que hay estados que no son válidos (hay órdenes que no pueden ser procesadas en todas las máquinas), por ello tenemos que penalizarlos evaluando con un valor muy grande.

Ya que se entendieron estos conceptos, observemos una forma simple en la que pueden ser implementados.

1. Estado

Sea O la cantidad de órdenes y M la cantidad de máquinas. Por simplicidad, consideraremos que las máquinas y órdenes están numeradas de 0 a $M-1$ y de 0 a $O-1$, respectivamente. Se propone usar un arreglo a de tamaño O donde $a[i]=k, 0 \leq k < M$ sea el número de máquina que procesará la orden i. Por ejemplo, el arreglo a=[0,0,1,2,0,2] representa el estado en el que la máquina 0 procesa las órdenes 0, 1 y 4, la máquina 1 procesa la orden 2, y la 2 las órdenes 3 y 5.

2. Transición

Una vez definido el estado, vemos que la transición se puede realizar fácilmente tomando aleatoriamente los enteros $0 \leq j < O$ y $0 \leq p < M$ y realizando la asignación $a[j]=p$. Se sugiere usar una distribución uniforme. Si se desea, se puede validar que $a[j] \neq p$ para evitar tomar el elemento y cambiarlo a la máquina en la que se encuentra, aunque este paso se podría omitir. También omitimos la validación del estado, pues ya dijimos que implementaremos un sistema de penalización. Estos dos mecanismos podrían mejorar la eficacia del método, así que vale la pena experimentar con ellos.

3. Evaluar

Evaluar es la parte más complicada del código. Una forma simple consiste en hacer M iteraciones sobre nuestro arreglo de estado, una por máquina, sacar la suma y los ajustes que corresponden, y quedarnos con el mayor resultado de esas iteraciones. La complejidad de ese proceso es de $O(MO)$, y si tomamos en cuenta

que tenemos que hacer varias iteraciones vemos que podríamos necesitar algo más rápido. Hay una forma de hacerlo en $O(M)$ (una sola iteración sobre el arreglo de estado), pero por ahora omitiremos su explicación. No hay que olvidarnos de implementar la validación en este paso también.

Código

Este código fue hecho en el lenguaje de programación R, el cual está dividido en las siguientes partes:

1. En el código primeramente se introdujo un algoritmo para leer los archivos con los datos de entrada, de tal manera que primero se lee el número de operaciones y el número de máquinas que van a haber en el problema (esos son los primeros parámetros de los archivos). Posteriormente se dará lectura a la tabla de los tiempos de procesamiento y finalmente se leerán la misma cantidad de tablas que número de máquinas, en este caso fueron 3 máquinas para ambos problemas, por lo tanto, se leyeron 3 tablas con un tamaño de: (número de operaciones) x (número de operaciones).
2. Posteriormente tenemos nuestra función objetivo, la cual se divide en:
    a. Entrada de la función: La función objetivo recibe una matriz X, la cual guarda las máquinas con sus respectivas operaciones.
    b. Crear vector resultados: Aquí se creó un vector resultados del tamaño del número de máquinas existentes, en el cual se guardarán las sumas totales de tiempo para cada máquina.
    c. Cálculo de suma de tiempos de procesamiento por máquina: Aquí al vector resultado se le va sumando el tiempo de procesamiento según la máquina que se haya elegido en dicha operación.
    d. Cálculo de suma de tiempos de ajuste por cada máquina: En esta parte es donde al vector resultado se le suma el tiempo de ajuste respectivo según la tabla de la máquina se esté utilizando en esa operación.
    e. Return: Es lo que la función objetivo regresa, en este caso es el vector resultado.
3. Finalmente tenemos el algoritmo del recocido simulado, el cual posee las siguientes divisiones:
    a. Variables
        i. t = 1000.

ii. err (error) = 1e-70.
iii. alfa = 0.99.
iv. Cantidad de ciclos = 2000.

b. Panel de variables para guardar el mejor resultado en general.
c. Función generadora de perturbaciones que va a provocar las modificaciones necesarias al vector X para que el algoritmo funcione adecuadamente.
d. Algoritmo inherente al recocido simulado (criterio de aceptación-rechazo de Metrópolis), en este caso modificado para que sólo tome el máximo de fx (recordar que la función objetivo regresa un vector de resultados del tamaño del número de máquinas a utilizar).
e. Modificación de la temperatura.
f. Guardado del mejor candidato.

Funcionamiento del algoritmo

Antes del recocido simulado

En este caso la variable X es una matriz de 2 x (número de operaciones), de tal manera que en la primera fila se guardan valores de 1 hasta el número de máquinas (aceptando repeticiones) y, en la segunda fila, se guardan valores de 1 hasta el número de operaciones existentes (sin repetir). La matriz X queda de la siguiente manera (ejemplo):

2	1	2	3	2	1	2	3	3	1
9	3	1	5	6	4	8	7	0	2

Por lo tanto, el primer paso es generar una matriz X de tal forma que arriba se generen valores del 1 hasta el número de máquinas aceptando repeticiones, y en la fila de abajo un acomodo aleatorio de las 'k' operaciones existentes sin aceptar repeticiones.

Una vez hecho esto, evaluamos nuestra matriz X en la función objetivo y este debería regresarnos un vector resultado de longitud 'n', donde 'n' es el número de máquinas. Este vector resultado tendrá la sumatoria total de los tiempos de procesamiento y de ajuste por cada máquina (en este caso 3 máquinas para ambos ejercicios). En esta ocasión no resulta necesario usar penalizaciones, ya que el problema no especifica que algo que no se pueda hacer, todos los movimientos son permitidos con la única condición de que el mayor de las 3 sumas se minimice todo lo posible.

Durante el recocido simulado (función generadora de perturbaciones)

Una vez dentro del recocido simulado, debemos generar una matriz xhat, donde xhat debe ser igual a la matriz X. A la matriz xhat se le debe provocar una perturbación que le haga distinta de la matriz X, en este caso se optó por realizar un cambio de máquina (cambio en la fila 1) y un swap de operación (cambio en la fila 2), pero hacer estos dos movimientos al mismo tiempo puede provocar demasiada aleatoriedad, por lo que un solo movimiento se hará a la vez, cuando 'k'

sea par sólo hará el cambio de máquina, y cuando sea impar hará el swap de operación, de esta manera se asegura que se realice una sola operación por cada iteración.

Antes de entrar a la función generadora se utilizó vector ‘dk’, el cual provocará el cambio. Este vector guardará 2 valores que son respectivos a las 2 posiciones a cambiar, por ende, se le generarán valores de 1 hasta número de operaciones. Un ejemplo es que dk = c(3,9), por lo tanto, se planea modificar la posición 3 y la posición 9 de la matriz xhat, ya sea haciéndole un swap o haciendo un cambio de máquina en la primera fila.

Para el cambio de máquina (que es cuando ‘k’ es un número par), nos aseguremos que el cambio de máquina sea total y no acepte el mismo, es decir, si por ejemplo tenía la máquina 2, sólo aceptar un valor que sea diferente de 2, en este caso puede ser el 1 o el 3. De esta forma, en esas dos localizaciones de la matriz se tendrán 2 nuevos valores de máquina para la operación que se encuentra en la fila 2, así que se ha provocado una perturbación exitosa y ahora xhat es un nuevo candidato a evaluar.

Para el swap de operación (que es cuando ‘k’ es un número impar), simplemente es tomar las columnas señaladas en el vector dk y cambiarlas de posición. Por ejemplo, si dk = c(3,9), lo que hay en la columna 3 (número de máquina y número de operación) se cambiará por lo que hay en la columna 9 (número de máquina y número de operación), y viceversa, de esta manera se ha provocado una perturbación exitosa y ahora xhat es un nuevo candidato a evaluar.

Una vez realizadas las perturbaciones a la matriz xhat, se procederá a evaluar en la función objetivo, de tal manera que fxhat albergará un vector con 3 resultados (respectivos a cada una de las 3 máquinas).

Esta parte del código luce de la siguiente manera:

```
# Paso 1 - Generar vector xhat
  dk = sample(1:noOperaciones, 2, replace = FALSE)
  xhat = x

  # Cambio de maquina
    if(k%%2 == 0){
      for(i in 1:length(dk)){
          repeat{
            random = sample(1:noMaquinas,1)
            if(xhat[1,dk[i]] != random){
              xhat[1,dk[i]] = random
              break
            }
          }
      }
    }
  # Swap de operacion
    else{
      dk= matrix(dk, ncol = 2, byrow = TRUE)
      for(i in 1:length(dk[,1])){
        random = dk[i,]
        a = xhat[,random[1]]
        xhat[,random[1]] = xhat[,random[2]]
        xhat[,random[2]] = a
      }
    }

  fxhat = fobjetivo(xhat)
```

Ya con la matriz xhat evaluada en la función objetivo, podemos proceder con el algoritmo del recocido simulado, donde se minimizará el máximo valor de tiempo entre las 3 máquinas. Aquí es igual que el recocido simulado original, con la única diferencia que ahora el 'fx' (la matriz X evaluada en la función objetivo) y el fxhat (la matriz xhat evaluada en la función objetivo) poseen un max(), el cual obtendrá el máximo valor de entre los 3 resultados que posean estos vectores. Esta parte del código queda de la siguiente manera:

```
#Paso 2
  if(max(fxhat) < max(fx)){
    x = xhat
    fx = fxhat
  }
  else{
    if(runif(1,0,1) < exp( (max(fx) - max(fxhat)) / t ) ){
      x = xhat
    }
    else{
      x = x
    }
  }

  if(max(fxhat) < max(fmejor)){
    # print('Entre')
    fmejor = fxhat
    xmejor = xhat
  }
```

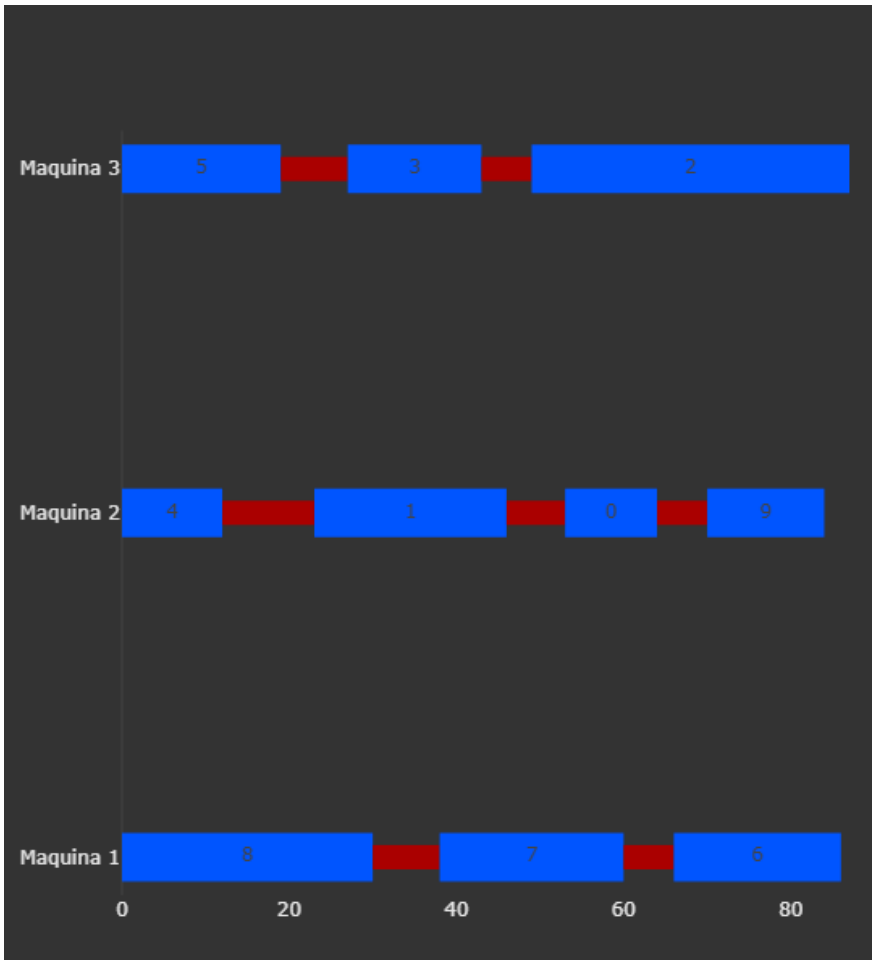
Y finalmente se procederá a modificar la temperatura multiplicando alfa*t (donde ‘t’ es igual a la temperatura).

Con esto el algoritmo puede ser capaz de obtener la respuesta correcta.

Resultados

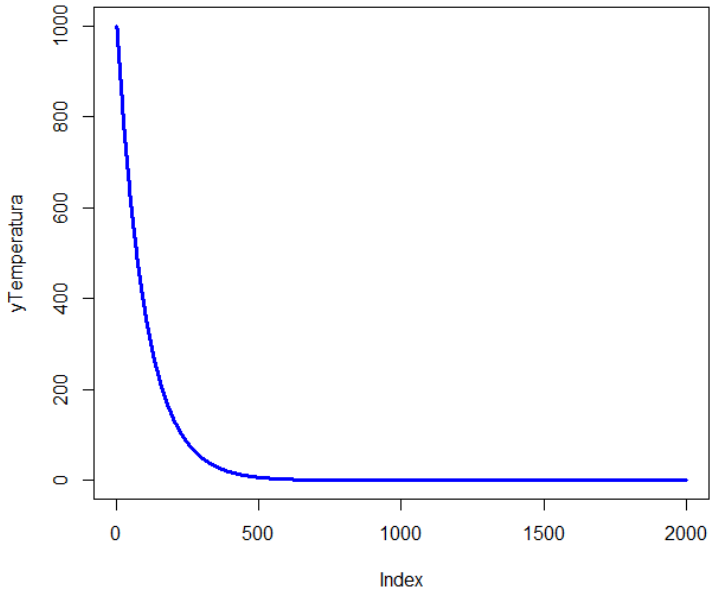
Ejercicio 1 (Instancia1.sjupm)

- Valores usados en el recocido simulado:
  - Temperatura = 1000.
  - Error = 1e-70.
  - Alfa = 0.99.
  - Ciclos dentro del recocido simulado = 2000.
  - Ciclos en general (veces que se repitió el algoritmo) = 100.
- Respuesta:
  - Máximo minimizado en: 87.
  - Mejor vector X:
3	3	2	2	2	3	1	1	2	1
5	3	4	1	0	2	8	7	9	6
  - Mejor vector resultado:
86	84	87
  - Encontrado en la iteración número (dentro del recocido simulado): 1819 de 2000.
  - Encontrado en la iteración número (dentro de las iteraciones generales): 20 de 100.
  - Promedio entre los 100 mejores máximos minimizados: 97.63.
  - Desviación estándar entre los 100 mejores máximos minimizados: 9.018487.
  - Número de veces encontrada el óptimo (87): 9 de 100.
  - Tasa de éxito: 9%
  - Mejores 100 matrices X:
    - Ver Excel llamado: “Resultados1.xlsx
  - Mejores 100 vectores resultado:
    - Ver Excel llamado: “Resultados1.xlsx
  - Diagrama de Gantt:



- Secuencia de operaciones:
  - Máquina 1: 8, 7, 6.
  - Máquina 2: 4, 1, 0, 9.
  - Máquina 3: 5, 3, 2.

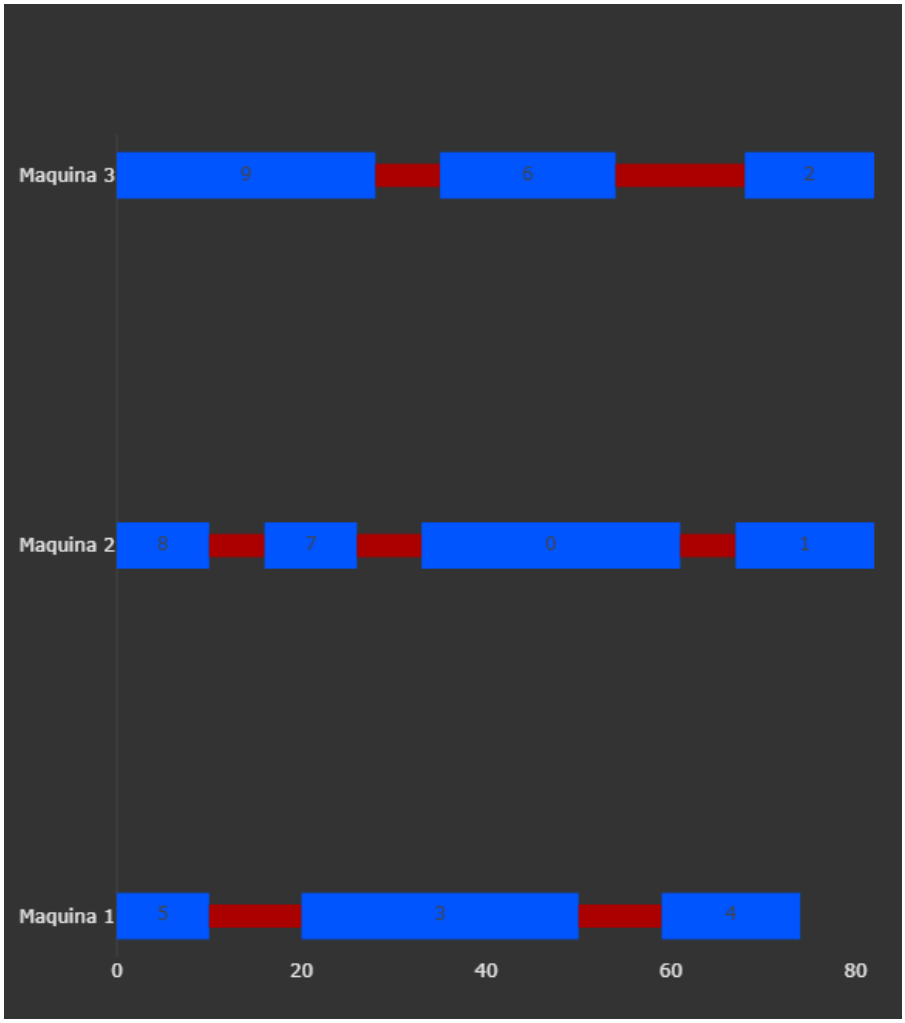
- Gráfica de temperatura:



Ejercicio 2 (Instancia2.sjupm)

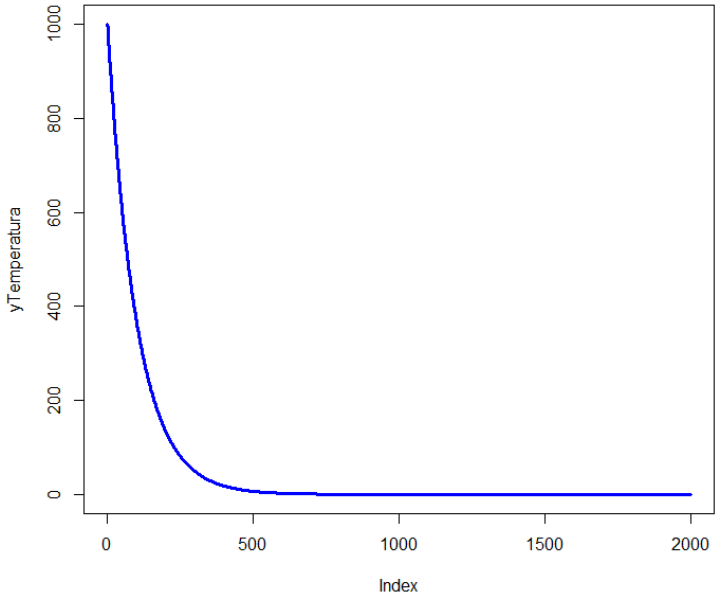
- Valores usados en el recocido simulado:
  - Temperatura = 1000.
  - Error = 1e-70.
  - Alfa = 0.99.
  - Ciclos dentro del recocido simulado = 2000.
  - Ciclos en general (veces que se repitió el algoritmo) = 100.
- Respuesta:
  - Máximo minimizado en: 82.
  - Mejor vector X:
1	3	1	1	2	3	2	3	2	2
5	9	3	4	8	6	7	2	0	1
  - Mejor vector resultado:
74	82	82
  - Encontrado en la iteración número (dentro del recocido simulado): 873 de 2000.

- Encontrado en la iteración número (dentro de las iteraciones generales): 12 de 100.
- Promedio entre los 100 mejores máximos minimizados: 87.83.
- Desviación estándar entre los 100 mejores máximos minimizados: 4.684069283.
- Número de veces encontrada el óptimo (82): 6 de 100.
- Tasa de éxito: 6%
- Mejores 100 matrices X:
  - Ver Excel llamado: “Resultados2.xlsx
- Mejores 100 vectores resultado:
  - Ver Excel llamado: “Resultados2.xlsx
- Diagrama de Gantt:



- Secuencia de operaciones:
  - Máquina 1: 5, 3, 4.
  - Máquina 2: 8, 7, 0, 1.
  - Máquina 3: 9, 6, 2.

- Gráfica de temperatura:



Conclusión de resultados

Como se pudo visualizar, se llegó al resultado usando este algoritmo, por lo que esto quiere decir que la función generadora de perturbaciones hizo su trabajo adecuadamente.

Los valores que se colocaron en el recocido simulado:

- Temperatura = 1000.
- Error = 1e-70.
- Alfa = 0.99.
- Ciclos dentro del recocido simulado = 2000.
- Ciclos en general (veces que se repitió el algoritmo) = 100.

Fueron puestos por ser los más óptimos para este caso. En la gráfica de la temperatura (puesta en la sección anterior dedicada a mostrar los resultados) se puede visualizar que desciende a 0 y se estabiliza en él alrededor de la iteración número 1000, por lo tanto, es una pérdida de recursos y de tiempo hacer más de 2000 iteraciones, ya que en esas instancias del algoritmo la temperatura es demasiado fría y no aceptará más valores, de hecho, será casi imposible aceptar valores por ser una temperatura muy baja, y eso se no lo dejará pasar el criterio de aceptación-rechazo de Metrópolis. Por ello se optó por parar el algoritmo justo a las 2000 iteraciones.

En este caso, el error utilizado fue de 1e-70, simplemente para que se cumplieran las 2000 iteraciones con tranquilidad y esta tasa de error no influyera en dar culmen al algoritmo de manera inoportuna.

El alfa de 0.99 se puso para que la temperatura disminuyera con mayor lentitud y hubiera un gran margen al principio para aceptar más valores.

Y finalmente tenemos al número de ciclos en general, que fueron de 100, esto se puso simplemente porque fue la cantidad estipulada a seguir para correr el algoritmo.

En base a los resultados obtenidos nos podemos percatar que, primeramente, se pudo alcanzar la meta de obtener el verdadero resultado óptimo y, por otra parte, ver la tasa de éxito del algoritmo, la cual fue de 9% y 6% (para el ejercicio 1 y 2 respectivamente) y unas desviaciones estándar de 9.018487 y 4.684069283 respectivamente para cada ejercicio. Esto nos quiere decir que el algoritmo tuvo menos éxito resolviendo el problema 2, ya que obtuvo valores más dispersos entre cada iteración y la tasa de éxito fue más baja, de tan solo el 6%. Por el contrario, el ejercicio 1 tuvo algo más de éxito, llegando casi al 10% y obteniendo valores menos dispersos entre cada iteración.

¿Se podría mejorar aún más la función generadora de perturbaciones para aumentar la tasa de éxito? Sí, pero de momento lo óptimo en la función generadora de perturbaciones es realizar 2 cambios de máquina y 2 swap de operación (haciendo sólo 1 de ellas en cada iteración), ya que al aumentar o disminuir cualquiera de esas dos provoca una considerable disminución en la tasa de éxito, en algunos casos llegando incluso al 0% (dentro de las 100 iteraciones).

Probablemente, esta no sea la mejor solución de algoritmo para resolver este tipo de ejercicios, se le podría dar un enfoque más específico para que el algoritmo siga una ruta determinada, pero por otra parte, esto tiene su desventaja en que no se sabe cuál es el camino claro, en este contexto no se tiene un mapa donde puede verificar si ir hacia un lado o hacia el otro, aquí se manejan una gran cantidad de valores discretos y no se sabe con certeza cuál combinación puede ser la idónea, por eso, este algoritmo puede ser una buena alternativa, ya que sea realizan pequeños cambios en la matriz a evaluar y no tantos como para ser considerado un algoritmo 100% aleatorizado.

Si se observa el problema, nos podemos percatar de 2, cosas, la primera es que se necesita hacer un cambio de operaciones en la matriz X, y la segunda es que se necesita hacer otro cambio en la matriz X, pero en las máquinas; así que al final el algoritmo no se puede centrar solamente en cambiar una de las 2, ya que se estaría limitando al algoritmo en demasía para dar con respuestas válidas. En

cambio, con este algoritmo sí se atacan esas posibilidades; aquí la clave sería en mejorar cómo hacer esos dos cambios sin volver el algoritmo en uno aleatorizado.