Recocido simulado rápido hibridado con enfriamiento para el Job Shop Scheduling Problem

Camilo Oberndorfer Mejía
coberndorm@eafit.edu.co, Universidad EAFIT

30 de octubre de 2023

Descripción

En términos tradicionales, el problema de programación de talleres (JSSP, por sus siglas en inglés) se puede definir como un sistema con n trabajos que deben procesarse en m máquinas, donde cada trabajo consta de m operaciones y cada operación tiene un orden de ejecución predefinido. El objetivo de cualquier algoritmo de programación es programar cada trabajo en las máquinas respectivas teniendo en cuenta un objetivo. El objetivo en este caso fue la minimización del tiempo total de ejecución (makespan) como objetivo de optimización. Para este problema se asumio que: (1) todas las máquinas están disponibles y todos los trabajos pueden cargarse en las máquinas al comienzo de la ejecución del programa, (2) una máquina solo puede procesar un trabajo a la vez y un trabajo no puede procesarse simultáneamente en más de una máquina, (3) una vez que se carga un trabajo en una máquina, no se interrumpirá hasta que se haya procesado por completo, (4) los tiempos de procesamiento y las restricciones de precedencia son estáticos y predefinidos, (5) no hay restricciones de tiempo de espera y almacenamiento, (6) se ignoran factores como los tiempos de preparación, costos, tiempos de lanzamiento, fechas de entrega, averías de máquinas y disponibilidad de operadores, entre otros. Cualquier sistema que cumpla con estas condiciones también se denomina JSSP estático.

En términos matemáticos, para JSSP, cada trabajo consta de m operaciones $\mathrm{O_{j1}}, \mathrm{O_{j2}}, \ldots, \mathrm{O_{jm}}$. La máquina requerida y el tiempo de procesamiento para cada operación Ojk se representan mediante Π_{jk} y Tjk respectivamente. Π y T son dos matrices con dimensiones $n \times m$. Por lo tanto, dos valores n y m, y dos matrices Π y T son capaces de describir completamente un JSSP estático. La salida final de cualquier algoritmo de programación es una matriz S de tamaño $n \times m$, donde un elemento S_{jk} es el tiempo de inicio de la operación Ojk. El JSSP se puede formular como:

$$\mathrm{Min\,Max}\left(S_{jk} + T_{jk} : 1 \leq j \leq n \;\text{ y }\; i \leq k \leq m\right) \tag{1}$$

Sujeto a

$$S_{jk} + T_{jk} \leq S_{j(k+1)} : 1 \leq j \leq n \text{ y } i \leq k \leq m-1 \tag{2}$$

$$S_{jk'} + T_{jk'} \leq S_{j'k} \text{ o } S_{j'k} + T_{j'k} \leq S_{jk'} : \Pi_{jk'} = \Pi_{j'k} \text{ y } 1 \leq j \text{ y} j' \leq n \text{ y } 1 \leq k \text{ y } k' \leq m \tag{3}$$

La primera restricción asegura los requisitos de precedencia y la segunda restricción evita la carga simultánea de trabajos en las máquinas.

La solución se va a representar mediante un vector que contiene exclusivamente el identificador numérico del trabajo. Luego, se examina la operación que debe llevarse a cabo en ese trabajo en el instante actual. Por ejemplo, si el trabajo 2 ya ha sido mencionado previamente y vuelve a mencionarse, esto indica que ha avanzado a la siguiente máquina requerida en su secuencia de operaciones. Como se muestra en el siguiente ejemplo:

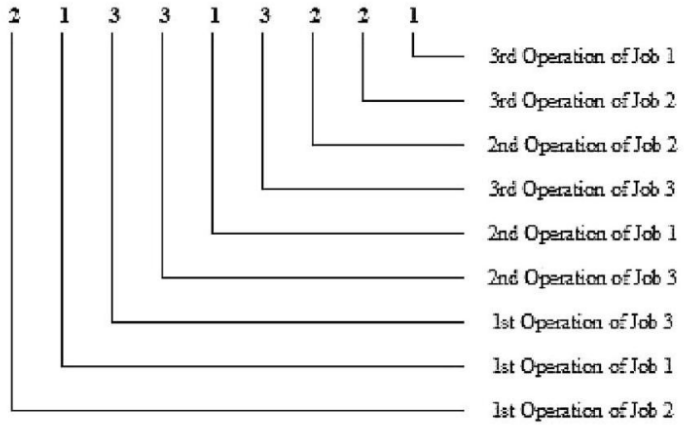

Figura 1: Diagrama de Flujo del

Algoritmo

La metodología de generación de vecindario propuesta por Nowicki & Smutnicki. La generación de vecinos es esencial en los algoritmos de búsqueda en vecindario, ya que permite explorar soluciones cercanas a la solución actual.

El primer paso es estimar el camino crítico en el programa de producción. El camino crítico es la secuencia de operaciones que determina la duración total del programa. Esto se logra mediante un algoritmo propuesto por Cruz-Chávez & Frausto-Solís. Una vez identificado el camino crítico, se divide en bloques, donde cada bloque contiene operaciones sucesivas que deben realizarse en la misma máquina.

La generación de vecinos se realiza cerca de los límites de estos bloques en el camino crítico. En resumen, los intercambios se realizan de la siguiente manera:

1. Se intercambian las dos últimas operaciones del primer bloque.
2. Se intercambian las dos primeras operaciones del último bloque.
3. Se intercambian las dos primeras y las dos últimas operaciones de los bloques restantes, siempre que el bloque tenga suficientes operaciones para realizar los intercambios.

Alg. 1 HFSAQ(JobList, MachineList)

Inicializar parámetros y variables (temperatura inicial, maximas iteraciones (I), maxima iteracion en cualquier temperatura (J))
Inicializar lista tabú como vacía
Inicializar la mejor solución (BS) como una solución aleatoria legal
Inicializar la solución actual como BS
Configurar temperatura (T) e iteraciones a la misma temperatura (J)
Configurar criterios de parada
Configurar la longitud de la lista tabú como $L = 10 + \frac{n}{m}$
while $i < I$ **do**
  while $j < J$ **do**
    Generar una lista de soluciones vecinas (movimientos) de la solución actual (CS)
    Seleccionar aleatoriamente una solución de la lista
    if solución vecina no está en la lista tabú **then**
      Calcular el makespan de la solución vecina candidata
      if solución candidata mejor que solución actual **then**
        Aceptar la solución candidata
        Actualizar la solución actual
        Reiniciar la lista tabú
        if makespan de la candidata es menor que el makespan de BS **then**
          Actualizar BS
        end if
      else
        Calcular la probabilidad de aceptación basada en la PDF de Cauchy
        if NúmeroAleatorio ¡Probabilidad **then**
          Aceptar la solución candidata
          Actualizar la solución actual
        end if
      end if
    else if makespan de la candidata es menor que el makespan de BS **then**
      Aceptar la solución tabú (criterio de aspiración)
      Actualizar la solución actual
      Reiniciar la lista tabú
      Actualizar BS
    end if
    $j = j + 1$
  end while
  Calcular T como $\frac{T_{\text{inicial}}}{i}$
  Devolver j a 1
  Restablecer J a su valor original si se hizo quenching
  if BS no cambia durante un número predefinido de iteraciones **then**
    #nota: esto es el ciclo de quenching
    Reducir T significativamente
    Aumentar J significativamente
  end if
  $i = i + 1$
end while
return BS

El siguiente diagrama de flujo fue sacado de Akram *et al.* (2016).

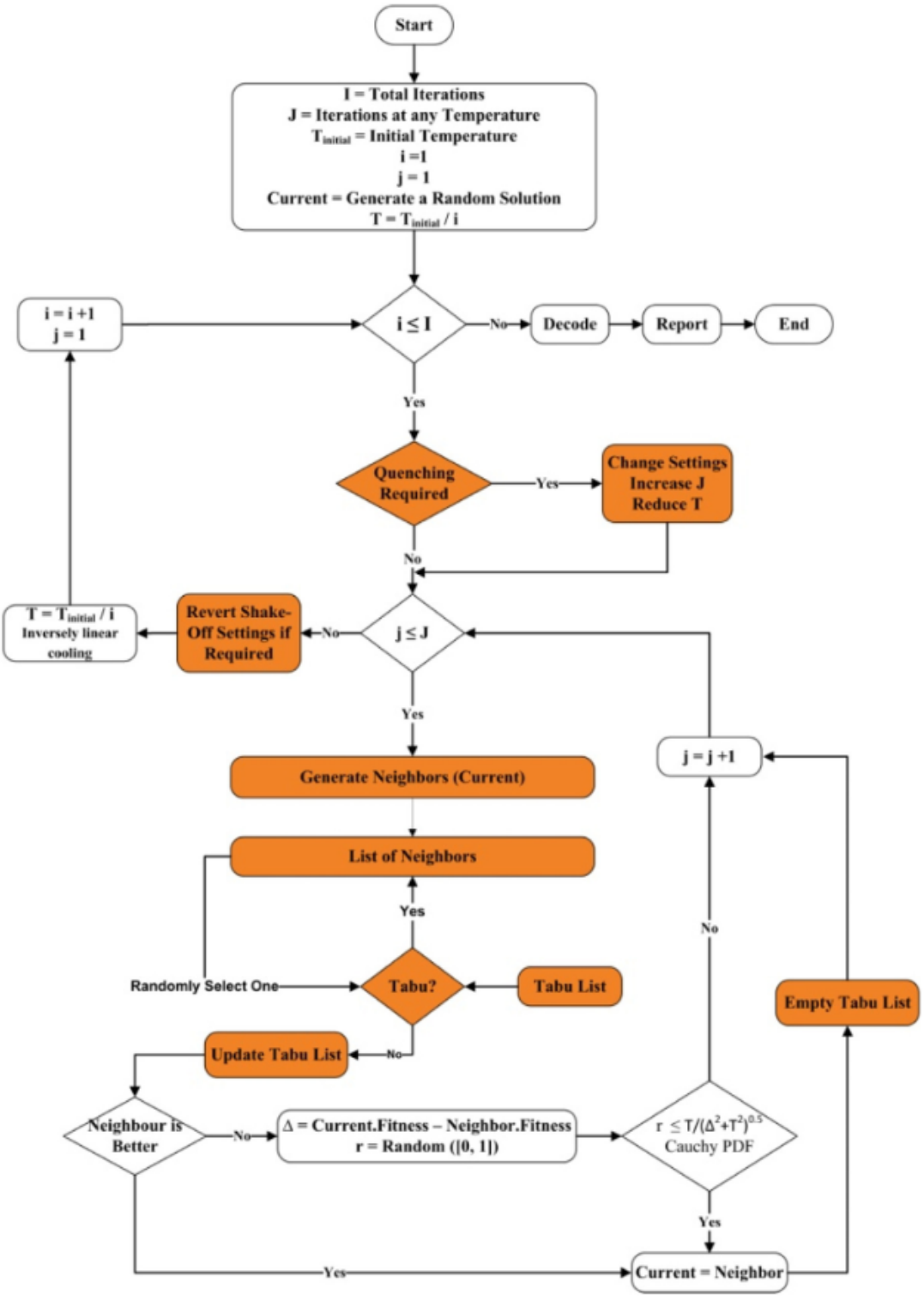

Figura 2: Diagrama de Flujo del

Conclusiones

Basándonos en el método descrito anteriormente para abordar el Problema de Programación de Taller (Job Shop Scheduling Problem), se pueden obtener las siguientes conclusiones:

1. **Eficacia en la generación de vecinos:** La metodología de generación de vecinos propuesta por Nowicki & Smutnicki se utiliza en el método descrito y es efectiva para generar vecinos factibles en el espacio de soluciones del problema de planificación de la tienda de trabajo. Esto es crucial porque la generación de vecinos es esencial en los algoritmos de búsqueda de vecindarios, y una generación de vecinos eficiente puede ayudar a explorar el espacio de soluciones de manera más efectiva.

2. **Enfoque en el camino crítico:** este enfoque se centra en identificar y trabajar con el camino crítico. Esto es una estrategia útil porque las operaciones en el camino crítico tienen el mayor impacto en el makespan total. El método tiene como objetivo mejorar de manera efectiva la programación de las máquinas al concentrarse en estas operaciones y permitir intercambios cerca de los límites de los bloques del camino crítico. Esto puede resultar en una reducción significativa del tiempo de ejecución del proyecto.

Referencias

Akram, Kashif, Kamal, Khurram, & Zeb, Alam. 2016. Fast simulated annealing hybridized with quenching for solving job shop scheduling problem. *Applied Soft Computing*, **49**, 510–523.

Cruz-Chávez, Marco Antonio, & Frausto-Solís, Juan. 2006. A New Algorithm That Obtains an Approximation of the Critical Path in the Job Shop Scheduling Problem. *Pages 450–460 of:* Gelbukh, Alexander, & Reyes-Garcia, Carlos Alberto (eds), *MICAI 2006: Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg.

Nowicki, Eugeniusz, & Smutnicki, Czeslaw. 1996. A fast taboo search algorithm for the job shop problem. *Management Science*, **42**(6), 797–813.