ALGORITMO VORAZ ITERATIVO CON MULTI-VECINDAD APLICADO AL PROBLEMA DE SECUENCIACIÓN DIFUSO MULTIPRODUCTO Y MULTIETAPAS

Anteproyecto de Grado

Tatiana Porras Cortes 20092015073

Correo tatiporras96@gmail.com

LINDSAY ÁLVAREZ POMAR
Director del trabajo de grado

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍA
PROYECTO CURRICULAR DE INGENIERÍA INDUSTRIAL

Bogotá D.C., Colombia. 2020-03-13

Tabla de Contenido

0 TÍTULO

"ALGORITMO VORAZ ITERATIVO CON MULTI-VECINDAD APLICADO AL PROBLEMA DE SECUENCIACIÓN DIFUSO MULTIPRODUCTO Y MULTIETAPAS"

1 RESUMEN

El Algoritmo Voraz Iterativo con Multi-Vecindad aplicado al Problema de Secuenciación Difuso Multiproducto y Multietapas (MNIG_to_FMMSP por sus siglas en inglés) no ha sido tratado en la literatura científica internacional. En este trabajo se explora esta combinación nueva. Para ello, se modificará el algoritmo MNIG de modo que pueda ser utilizado para resolver el modelo FMMSP.

Al final se reportarán los resultados del algoritmo aplicado a una instancia del problema, y se compara dichos resultados con los de otros cuatro algoritmos que han sido aplicados al modelo FMMSP en la literatura.

Palabras Clave: Algoritmo Voraz Iterativo, Multi-vecindad, Difuso, Secuenciación, Multiproducto, Multietapas

1.1 ABSTRACT

The Multi-neighborhood Iterated Greedy algorithm applied to the Fuzzy Multiproduct Multistage Scheduling Problem (from now on MNIG_to_FMMSP) has not been treated in the international scientific literature. This work explores this new combination. For that, the MNIG algorithm will be modified so that it can be used to solve the FMMSP model.

At the end, the results of the algorithm will be reported, after applying it to several instances of the problem, and the results will be compared to those of another four algorithms that have been applied to the FMMSP model in the literature.

Keywords: Iterated Greedy, Multi-neighborhood, Scheduling, Fuzzy, Multiproduct, Multistage

2 INTRODUCCIÓN

Mediante el presente trabajo de tesis de pregrado se pretende hacer un aporte aunque pequeño al conocimiento. En la ingeniería industrial existen diversas áreas en las que se puede hacer un aporte de este tipo. Existe el área de investigación de operaciones, el área de gestión, mercadeo, higiene industrial, seguridad y salud en el trabajo, ergonomía, etcétera.

Un aporte de nuevo conocimiento en pregrado es un aporte pequeño y muy específico, detallado. Por ello se selecciona una de las áreas de conocimiento de la ingeniería industrial y dentro de esa área se elige un tema en particular. De ese tema elegido se trabaja un detalle que no haya sido estudiado con anterioridad.

En el presente trabajo se eligió el tema de secuenciación, mejor conocido por

su nombre en inglés como "scheduling". Dentro de este tema se revisó el estado del arte, y se encontró un modelo de scheduling que ha sido poco estudiado, y por aparte se encontró un algoritmo que nunca ha sido aplicado al modelo, pero que podría ser aplicado. El modelo encontrado [1] es llamado FMMSP que significa Fuzzy Multiproduct Multistage Scheduling Problem, que en español se podría traducir como Problema de Secuenciación Difuso Multiproducto y Multietapas. Es un modelo interesante de ser estudiado pues incluye números difusos, múltiples productos, y múltiples etapas. El algoritmo encontrado [2] es llamado MNIG que significa Multi-Neighborhood Iterated Greedy algorithm. En español se traduce como algoritmo Voraz Iterativo con Multi-Vecindad. Se ha encontrado que este algoritmo sirve para resolver problemas de scheduling [2].

Lo novedoso resulta en que dicho algoritmo jamás ha sido aplicado al modelo, a tal punto que para realizar este trabajo se hará necesario modificar el algoritmo, pues el algoritmo original no se puede aplicar directamente al modelo. No existen pruebas de que este algoritmo sea apropiado para este modelo. Con el presente trabajo se propone comprobar que tan bueno es el algoritmo MNIG aplicado al modelo FMMSP, al compararlo con otros cuatro algoritmos que ya han sido aplicados al modelo FMMSP [1].

3 PLANTEAMIENTO DEL PROBLEMA

El problema que ataca esta tesis es explorar nuevas posibilidades de encontrar secuencias en las que se puedan realizar los trabajos en un sistema de producción del tipo FMMSP, de modo tal que dichas secuencias de trabajos tengan un tiempo de terminación máximo lo más bajo posible.

Siendo que el espacio de búsqueda del modelo FMMSP para problemas de tamaño realista es muy grande como para buscar de manera exhaustiva la secuencia con el menor de los tiempos de terminación máximo (es decir probando cada secuencia posible por separado), se diseñan algoritmos que permitan buscar en regiones promisorias del espacio de búsqueda. De esta manera, se hace innecesario revisar todo el espacio de búsqueda, y los algoritmos dan buenos resultados, o sea, con bajos tiempos de terminación máximo.

El algoritmo MNIG aplicado al modelo FMMSP tiene la posibilidad de competir en el encuentro de buenas soluciones. Aplicando el algoritmo MNIG un sistema productivo tipo FMMSP podría encontrar secuencias de trabajos con bajos tiempos de terminación máximo. Sin embargo esta posibilidad aún no está comprobada, de eso se trata esta tesis.

3.1 Elementos del Problema

Según lo anterior, los elementos del problema son:

- ✠ Se requieren secuencias con bajos tiempos de terminación en el modelo FMMSP.
- ✠ El algoritmo MNIG tiene potencial para encontrar dichas secuencias en un tiempo prudente. Los resultados se pueden comparar a otros cuatro algoritmos que ya han sido usados en el modelo FMMSP

3.2 Pregunta del Problema

¿Puede aplicarse el algoritmo MNIG al modelo FMMSP para obtener resultados similares a los obtenidos mediante los otros cuatro algoritmos ya probados?

3.3 Subpreguntas del Problema

¿Puede implementarse el algoritmo MNIG aplicado al modelo FMMSP en un lenguaje de programación como Python?

¿Puede correr el algoritmo MNIG aplicado al modelo FMMSP e imprimir o devolver una secuencia de trabajos factible en ese modelo?

¿Es comparablemente bueno el algoritmo MNIG aplicado al modelo FMMSP respecto a los otros cuatro algoritmos que ya han sido aplicados a ese modelo?

4 OBJETIVOS

4.1 Objetivo General

- Aplicar el algoritmo MNIG al modelo FMMSP

4.2 Objetivos Específicos

- Implementar el algoritmo MNIG aplicado al modelo FMMSP en el lenguaje de programación Python
- Correr el algoritmo MNIG en una instancia del modelo FMMSP y obtener secuencias factibles
- Comparar el algoritmo MNIG aplicado al modelo FMMSP con los otros cuatro algoritmos que se han aplicado al modelo FMMSP. La comparación se hará con el tiempo de terminación máximo de la secuencia resultante

5 JUSTIFICACIÓN

Se propone este trabajo porque en el mundo académico existe una necesidad de comprobar que los algoritmos den resultados esperados al ser aplicados a los modelos. En particular en el mundo del scheduling ocurre que un algoritmo que se usa para un tipo de modelos pueda ser adaptado para tratar otro tipo de modelos diferente. Este trabajo se justifica como un aporte de prueba sobre

si el algoritmo MNIG sirve para resolver el modelo FMMSP en comparación a otros algoritmos ya establecidos.

En el futuro, podría ser preferible para un sistema de producción tipo FMMSP contar con un algoritmo como el MNIG. En cualquier caso, no sobra contar con un algoritmo extra para obtener las secuencias de trabajos en sistema de producción de este tipo.

De este proyecto puede llegar a crearse un paper que podría ser publicado en inglés en una revista indexada internacional. Dicho paper podría publicarse en el Workshop on Engineering Applications que se organiza en la Universidad Distrital. Este aporte sirve también como justificación del proyecto.

6 ALCANCES

Alcances: Los alcances de este proyecto están bien definidos. Parte del alcance consiste en escribir un algoritmo en Python, específicamente implementar el algoritmo MNIG en un programa de consola de Python. En el proyecto también se incluye la tesis de grado en la que se compararán los resultados del algoritmo implementado en este proyecto con los de otros cuatro algoritmos aplicados al modelo FMMSP. Dicha tesis será entregada a la Universidad Distrital.

De manera opcional se incluye la publicación de un paper en el Workshop on Engineering Applications. Este paper es solamente opcional porque no se saben las condiciones futuras, por ejemplo podría pasar que se acabaran las fechas de publicación, en ese caso este proyecto terminaría (pues el paper era opcional) y se publicaría el paper después, en las siguientes fechas pero no como parte de este proyecto para no atrasarlo teniendo en cuenta que el plazo para graduarse en la Universidad Distrital es prestablecido, entonces si las fechas de la graduación no cuadran con las fechas del paper, se publica el paper después de la graduación.

7 MARCO DE REFERENCIA

El marco de referencia se enfoca tanto en el scheduling de los sistemas de producción tipo FMMSP, como en el algoritmo MNIG que es un tipo de metaheurística.

7.1 Antecedentes y Marco Teórico

En los antecedentes encontramos dos temas pilares: el scheduling, y las metaheurísticas. Del libro de scheduling de Pinedo [3] se tiene una interrelación entre los temas de los antecedentes. Se han creado multitud de heurísticas y metaheurísticas para resolver los problemas de scheduling. También ha ocurrido que muchas metaheurísticas pueden ser modificadas para tratar otros problemas diferentes de scheduling.

El scheduling es un proceso de toma de decisiones en el que se asignan recursos a los trabajos (u ordenes) a realizar. En esta asignación los trabajos se van realizando en virtud de los recursos que le han sido asignados. En otras palabras, no se puede completar un trabajo que no ha sido asignado a los recursos que lo llevan a cabo. En la decisión de cómo realizar dichas asignaciones surge la importancia del scheduling, por el hecho de que distintas asignaciones conllevan a diferentes tiempos de realización de los trabajos. [3]

La medida de desempeño más ubicua del scheduling es el tiempo de terminación máximo, también conocido como makespan, que es el tiempo en que se termina el último trabajo de la secuencia. En este sentido, el makespan representa el tiempo que toma realizar todos los trabajos. En el scheduling, lo más comúm es que la función objetivo sea minimizar el makespan, es decir, minimizar el máximo de los tiempos de terminación de los trabajos de la secuencia [3]

Respecto a las heurísticas y metaheurísticas, en ambas se renuncia a la optimalidad, se renuncia a encontrar la mejor solución posible según determinada medida de desempeño, a cambio de encontrar soluciones aceptables en un tiempo prudente. [4]

En cuanto al algoritmo MNIG y la diferencia entre heurísticas y metaheurísticas, las metaheurísticas pueden abarcar un amplio abanico de problemas, mientras que las heurísticas se diseñan para un problema muy específico [4]. Acorde a lo anterior, el algoritmo MNIG es una metaheurística pues es un algoritmo que se puede usar para diversos tipos de problemas, y aunque originalmente no haya sido diseñado pensando en el modelo FMMSP se puede adaptar fácilmente sin cambiar la esencia de la metaheurística.

7.2 Marco Conceptual

Scheduling: problemas de optimización en los que se establece una secuencia de trabajos en un sistema de producción, y se busca optimizar la medida de desempeño de interés en el sistema de producción.

Existe variedad de tipos de sistemas de producción, los primera división se hace en sistemas determinísticos y estocásticos. En los sistemas determinísticos se conoce con precisión los parámetros del sistema, mientras que en los sistemas estocásticos los parámetros tienen variabilidad [3]

Luego están los sistemas de una sola máquina, que son sistemas teóricos que sirven para formular modelos más complejos, puesto que en los sistemas de producción reales rara vez se presenta el caso de una sola máquina. Después vienen los sistemas de máquinas paralelas en los que varias máquinas pueden procesar los mismos trabajos. En el siguiente nivel de complejidad se encuentran los sistemas tipo Flow Shop en los que los trabajos van fluyendo de una máquina

a la siguiente en un mismo orden de máquinas. Sigue los sistemas tipo Job Shop en los que el orden de las máquinas puede ser diferente para cada trabajo. Por último los sistemas Open Shop en los que el orden de las máquinas por las que debe pasar cada trabajo puede ser parte de las decisiones de scheduling. [3]

Metaheurística: Las metaheurísticas surgen como una alternativa a la situación en que la optimización de problemas toma demasiado tiempo. Cuando los problemas a optimizar son grandes y según aumentan de tamaño, el tiempo de optimización puede pasar de horas a días, a meses y años, e incluso mucho más tiempo. Los problemas realistas tienden a ser de gran tamaño, por lo que su optimización puede tomar años o más, incluso con los computadores más rápidos, en estos casos se usan metaheurísticas, que aunque no optimizan el problema, sí proveen soluciones aceptables en un tiempo razonable. [4]

La complejidad de una metaheurística, y en general la complejidad de un algoritmo se mide con el tiempo que toma su ejecución en función del tamaño del problema. Dicho tiempo se denota usando la llamada notación O-grande, por ejemplo, si n es el tamaño del problema, y $O(n^2)$ es la notación O-grande de un algoritmo dado, significa que el tiempo de resolución de dicho algoritmo aumenta al cuadrado del tamaño del problema n. Esto no significa que el tiempo de resolución sea igual al cuadrado de n, sino que el tiempo de resolución va aumentando con el tamaño del problema, y se acota por una cantidad proporcional al cuadrado del tamaño del problema. [4]

Hay varios tipos de complejidad en la optimización de problemas y en los algoritmos. Se considera que un tiempo de resolución razonable es el llamado tiempo polinomial, así, un algoritmo de tiempo polinomial, es un algoritmo cuyo tiempo de resolución puede ser acotado con una función polinomial, y su notación O-grande respectiva será $O(p(n))$ donde $p(n)$ es una función polinomial del tamaño del problema n. [4]

Ocurre en los casos reales que muchas veces el tiempo de resolución de la optimización en un problema de scheduling, no puede ser acotado por una función polinomial. En estos casos la complejidad del problema es no polinomial, o NP por sus siglas en inglés. Una demostración importante, es que la optimización de modelos tipo Flow Shop y la optimización de modelos más complejos que se basen en el Flow Shop es de complejidad NP-Hard, que es una variante de los problemas no polinomiales. [5]

Lo anterior requiere que para estos modelos de scheduling se usen metaheurísticas para su resolución.

Modelo FMMSP: El modelo FMMSP representa un sistema de producción en el que los trabajos van pasando por etapas, y en cada etapa hay un número de máquinas (o unidades como les llaman en la literatura). Cada trabajo puede ser procesado solamente por una unidad en cada etapa. Cuando un trabajo

pasa por todas las etapas se dice que está terminado. En el modelo FMMSP se busca optimizar el tiempo de terminación máximo de los trabajos, es decir, el tiempo en el que el último trabajo de la secuencia es terminado, medido desde el inicio de la producción. [1]

Para modelar el tiempo de producción en el modelo FMMSP se utilizan números difusos, en concreto número difusos triangulares. El hecho de que el tiempo de producción se represente mediante números difusos triangulares conlleva a que también el tiempo de terminación y el tiempo de inicio de cada trabajo sean también números difusos triangulares. Los números difusos triangulares son usados en situaciones donde se quiere representar la incertidumbre. Para el caso del modelo FMMSP se usan los números difusos triangulares para representar la incertidumbre en los tiempos de terminación, en otras palabras, un trabajo puede terminar o bien antes o bien después del momento en que se espera su terminación. En el modelo FMMSP cada número difuso triangular está compuesto a su vez por tres números. Esto es, cada tiempo de procesamiento, de terminación, o de inicio es una terna ordenada, en la que el primer número representa el tiempo optimista (bien sea de procesamiento, terminación, o inicio según el caso), el segundo número representa el tiempo esperado, y el tercer número representa el tiempo pesimista. [1]

Los números difusos triangulares se definen de la siguiente forma: [6]

Sea $\tilde{A}$ un número difuso triangular, el símbolo encima de la A indica que se trata de un número difuso. Entonces $\tilde{A} = (a, b, c)$ donde a, b, y c son tres números reales que conforman al número difuso.

Por otra parte sea x cualquier número real, y sea $\mu_{\tilde{A}}(x)$ la función del grado de pertenencia de x al número difuso $\tilde{A}$, entonces:

$$\mu_{\tilde{A}}(x) = \begin{cases} 0, & x < a; \\ \frac{x-a}{b-a}, & a \leq x \leq b; \\ \frac{c-x}{c-b}, & b < x \leq c; \\ 0, & x > c, \end{cases} \tag{1}$$

Como se observa en la ecuación (1) la función del grado de pertenencia está entre 0 y 1. Antes de a y después de c su valor es 0, es decir que sólo tiene valor distinto de 0 entre a y c. Por otra parte, esta función del grado de pertenencia llega a un valor de 1 cuando $x = b$, entre a y b el valor de la función va ascendiendo de 0 a 1, y entre b y c desciende de 1 a 0, lo que hace que efectivamente la función forme un triangulo que inicia en a, tiene el máximo en b y termina en c. [6]

Haciendo uso de los números difusos triangulares, a continuación se define el modelo FMMSP: [1]

Sea N el número de órdenes o trabajos a procesar, sea I el conjunto de los trabajos, tal que $I = \{1, 2, \ldots, i, \ldots, N\}$ donde i es cualquier trabajo del conjunto I.

Sea L el número de etapas por las que deben pasar los trabajos, sea S el conjunto de las etapas, tal que $S = \{1, 2, \ldots, s, \ldots, L\}$ donde s es cualquier etapa del conjunto S.

En cada etapa s se presentan n unidades de procesamiento paralelas, sea U_s el conjunto de unidades de procesamiento de la etapa s, tal que $U_s = \{1, 2, \ldots, n\}$.

El tiempo de procesamiento del trabajo i en la unidad u es un número difuso triangular, por lo que se puede representar como: $\tilde{T}_{i,u} = (T^1_{i,u}, T^2_{i,u}, T^3_{i,u})$. Notar que u es cualquier unidad de todas las etapas, y no de una etapa en particular, es decir si se númeran todas las unidades o "máquinas" sin distinguir la etapa, entonces u es uno de estos números, o sea u es cualquier máquina de todo el sistema de producción. Las unidades que pertenecen a una etapa en particular se denotan con U_s.

El tiempo de inicio del trabajo i en la etapa s también es un número difuso triangular y se representa como $\widetilde{Ts}_{i,s}$, y el tiempo de terminación del trabajo i en la etapa s siendo un número difuso triangular se representa como $\widetilde{Tf}_{i,s}$

Para definir completamente el modelo hacen falta pocas variables, entre ellas una variable binaria, llamada $Z_{i,u,s}$. Si la variable binaria es igual a 1 indica que el trabajo i se procesará en la unidad u de la etapa s, en caso contrario si la variable es igual a 0 significa que el trabajo i no será procesado en la unidad u de la etapa s. Sea F el tiempo de terminación del último trabajo de la secuencia, por tanto F es la función objetivo a minimizar, F también es llamado el makespan. Y por último sea M un número muy grande, normalmente conocido como gran M.

Teniendo las definiciones anteriores, el modelo FMMSP:

$$minF = max(\widetilde{Tf}_{i,L}) \quad \forall i \in I \tag{2}$$

$$\sum_{u \in U_s} Z_{i,u,s} = 1 \qquad \forall i \in I, s \in S \tag{3}$$

$$\widetilde{Tf}_{i,s} = \widetilde{Ts}_{i,s} + \sum_{u \in U_s} (Z_{i,u,s} * \tilde{T}_{i,u}) \quad \forall i \in I, s \in S \tag{4}$$

$$\begin{gathered} \widetilde{Tf}_{i,s} + M * (2 - Z_{i,u,s} - Z_{j,u,s}) \leq \widetilde{Ts}_{j,s} \\ \forall i \in I, j \in I, i < j, s \in S, u \in U_s \end{gathered} \tag{5}$$

$$\widetilde{Tf}_{i,s} \leq \widetilde{Ts}_{i,s+1} \quad \forall i \in I, s \in S \tag{6}$$

La ecuación (2) es la función objetivo de minimizar el máximo de los tiempos de terminación de los trabajos en la etapa L, que es la última etapa. Luego la ecuación (3) establece que el trabajo i en la etapa s solamente se pueda realizar en una sola unidad u. La ecuación (4) muestra el tiempo de terminación siendo igual al tiempo de inicio más el tiempo de procesamiento. La inecuación (5) es una restricción para el tiempo de inicio del trabajo j si este se realiza en la misma unidad que un trabajo anterior i. Por último la inecuación (6) es una restricción al tiempo de inicio del trabajo i en la etapa siguiente $s+1$ que lo hace ser mayor o igual que el tiempo de terminación del trabajo i en la etapa s. [1]

Teniendo definido el modelo, para resolverlo hace falta saber exactamente qué se entiende por el máximo entre dos números difusos triangulares. Como en el modelo se calcula el máximo de los tiempos de terminación, y dichos tiempos son números difusos triangulares, entonces se requiere definir los operadores de comparación $<$, y $>$ para números difusos triangulares. Solo es necesario definir un operador pues el otro es su negación. El operador $>$ se define de la siguiente manera: [1]

Sean $\tilde{A} = (a^1, a^2, a^3)$ y $\tilde{B} = (b^1, b^2, b^3)$. Para establecer que $\tilde{A} > \tilde{B}$ se dan los tres siguientes casos

$$\begin{aligned} Si\left(\frac{a^1+2*a^2+a^3}{4}\right) > \left(\frac{b^1+2*b^2+b^3}{4}\right), entonces \tilde{A} > \tilde{B} \\ Si\left(\frac{a^1+2*a^2+a^3}{4}\right) = \left(\frac{b^1+2*b^2+b^3}{4}\right), ya^2 > b^2, \\ entonces \tilde{A} > \tilde{B} \\ Si\left(\frac{a^1+2*a^2+a^3}{4}\right) = \left(\frac{b^1+2*b^2+b^3}{4}\right), ya^2 = b^2, \\ a^3 - a^1 > b^3 - b^1, entonces \tilde{A} > \tilde{B} \end{aligned} \tag{7}$$

Algoritmo MNIG: El algoritmo MNIG se entiende dividido en cuatro algoritmos, cada uno realizando un rol diferente y al ser aplicados como se indica logran una solución. [2]

El primer algoritmo es llamado DNEH_SMR, sirve para obtener una secuencia de trabajos inicial promisoria en la región de búsqueda. DNEH_SMR significa "Distributed NEH with Small-Medium Rule" a su vez NEH proviene de "Nawaz, Enscore, Ham" que es un reconocido algoritmo para minimizar el makespan. [2]

El algoritmo MNIG usa su propia notación, que en ocasiones usa los mismos símbolos del modelo FMMSP. En este trabajo se le dará prelación a los símbolos del modelo FMMSP, puesto que el algoritmo MNIG se está adaptando al modelo,

así también se adaptarán los símbolos de modo que sean congruentes con los del modelo FMMSP.

Para entender el algoritmo DNEH_SMR se requieren algunos símbolos extra. Una secuencia de trabajos factible se denota como π, entonces π es una solución factible al problema. En el algoritmo DNEH_SMR se divide en dos la secuencia, el punto en que se divide es llamado k, y es definido como $k = \lfloor \frac{L}{2} \rfloor$. Luego en el algoritmo DNEH_SMR se promedian los tiempos de cada grupo de etapas: antes de k y después de k. [2]

En el modelo FMMSP no se consideran los tiempos de procesamiento por etapa sino por unidad (o máquina) por lo que se hace necesario modificar ligeramente el algoritmo DNEH_SMR. Para hacer un tiempo de procesamiento equivalente por etapa, partiendo de los tiempos por unidad, se promedian los tiempos de las unidades en la etapa, es decir $\forall u \in U_s$, de este modo defínase $\widetilde{Ta}_{i,s} = \frac{1}{n} * \sum_{u \in U_s} \widetilde{T}_{i,u}$, recordando que n es el número de unidades de la etapa s. Ta es por $Taverage$.

Teniendo $\widetilde{Ta}$ se definen los tiempos de procesamiento promedio por grupo de etapas, para el primer grupo antes de k, $\widetilde{T}'_{i,1} = \frac{1}{k} * \sum_{l=1}^{k} \widetilde{Ta}_{i,l}$, y para el segundo grupo después de k, $\widetilde{T}'_{i,2} = \frac{1}{L-k} * \sum_{l=k+1}^{L} \widetilde{Ta}_{i,l}$. Nótese el símbolo prima para indicar que son los tiempos promedio de los grupos de etapas. [2]

Del algoritmo DNEH_SMR resulta una secuencia factible y además no aleatoria, pues intenta crear una solución inicial de calidad. Como parte de ello se crea una secuencia ordenando los trabajos, en orden ascendente del tiempo promedio en el grupo 2 $\widetilde{T}'_{i,2}$, sea dicha secuencia π_{re1} donde $re1$ indica que es el primer reordenamiento de la secuencia. Para el segundo reordenamiento, π_{re2} se consigue al tomar el primer trabajo y luego el trabajo de en medio de π_{re1}, eliminando los trabajos tomados de π_{re1}. Si el número de trabajos en el que se busca el trabajo de en medio es par, se puede tomar el trabajo que está antes de la mitad (pues cuando el número de trabajos que quedan es par existen dos trabajos en el medio, tomar el primero). El hecho de que se van eliminando los trabajos significa que solo hay que repetir la instrucción: tomar el primero, tomar el de en medio, tomar el primero, tomar el de en medio, hasta agotar los trabajos y así dar forma a π_{re2}. [2]

Por último en el algoritmo, se toma cada trabajo, y se reinserta en cada posición y se deja en la posición en la que resulte el mínimo makespan. Sea π_{re3} la secuencia resultante de estas reinserciones. El algoritmo DNEH_SMR retorna la secuencia π_{re3} como su resultado. [2]

Teniendo los elementos anteriores, a continuación se presenta el algoritmo DNEH_SMR en su versión de pseudocódigo de Python:

Algoritmo 1 DNEH_SMR en pseudocódigo Python

```
def DNEH_SMR(todas las T̃_{i,u}):
1:  Computar k = ⌊L/2⌋
2:  Computar T̃a_{i,s} = (1/n) * Σ_{u∈U_s} T̃_{i,u} ∀i ∈ I, s ∈ S
3:  Computar T̃'_{i,1} = (1/k) * Σ_{l=1}^{k} T̃a_{i,l} ,
        y T̃'_{i,2} = 1/(L−k) * Σ_{l=k+1}^{L} T̃a_{i,l}  ∀i ∈ I
4:  Ordenar los trabajos según el orden ascendente
    de T̃'_{i,2}, guardando la secuencia en π_{re1}
5:  Crear la secuencia π_{re2}, quitar el primer
    trabajo y luego el de en medio de π_{re1}, y ponerlos
    en π_{re2}, repetir con los trabajos restantes hasta
    que ya no queden trabajos en π_{re1}
6:  Copiar π_{re2} en π_{re3}
7:  for j in π_{re3}:
8:      Reinsertar j en cada posición de π_{re3} y medir
        el makespan
9:      Dejar j en la posición con el mínimo makespan
        sin mover los trabajos de iteraciones
        anteriores
10:
    return π_{re3}
```

Con el algoritmo DNEH_SMR se obtiene una solución inicial π_{re3}. Este primer algoritmo, y los dos siguientes, algoritmos 1, 2, y 3 se integran en el algoritmo 4. Desde ya decir que el algoritmo 4 es el algoritmo MNIG. Para llevar a cabo el algoritmo MNIG hace falta mostrar otros dos algoritmos, los algoritmos 2 y 3 que serán trabajados a continuación.

Para hacer el algoritmo MNIG se necesita poder destruir y reconstruir la secuencia π_{re3} reconstruyendo de modo que se vaya mejorando la función objetivo. La destrucción y reconstrucción de la secuencia se logran con el algoritmo 2. También se necesita hacer la búsqueda local con multi-vecindad, en la que se hacen inserciones e intercambios de trabajos en la secuencia, buscando la mejora en la función objetivo. Esta búsqueda local con multi-vecindad se realiza con el algoritmo 3. [2]

En el algoritmo 2, la destrucción y reconstrucción de una secuencia de trabajos tiene por objeto la mejora en la función objetivo. Para hacer este algoritmo de destrucción y reconstrucción, se comienza por remover d trabajos de la secuencia de manera aleatoria. Con estos trabajos removidos se forma la secuencia π_d. Con los trabajos restantes se forma la secuencia π_r. Luego se toma cada

trabajo en π_d y se reinserta en cada posición de π_r, se mide el makespan a pesar de que esta secuencia tenga menos trabajos que la original y se deja el trabajo en la posición que tenga el mínimo makespan. [2]

Luego cada trabajo en π_r se reinserta en la secuencia bajo construcción, se mide el makespan, y si éste es menor que el makespan inicial entonces se inserta el el trabajo en la posición que haya generado el nuevo y mejor makespan. Se hace lo mismo con los restantes trabajos en π_r sin reemplazar los que ya hayan sido insertados. Por último se repite este proceso con los demás trabajos en π_d. La secuencia resultante es retornada por el algoritmo. [2]

Llámesele π_{input} a la secuencia a ser destruida y reconstruida, y sea π_{desrec} la secuencia retorna por el algoritmo 2 tras ser destruida y reconstruida. En consideración de lo dicho se presenta el algoritmo de destrucción y reconstrucción en su versión de pseudocódigo de Python:

Algoritmo 2 Destrucción y reconstrucción de las secuencias

```
def destruction_reconstruction(π_input, d):
1:  Remover aleatoriamente d trabajos de la
    secuencia π_input y ponerlos en la secuencia π_d
    Poner el resto de trabajos en la secuencia π_r
2:  for j in π_d:
3:      Reinsertar j en cada posición de π_r y medir
        el makespan como fitness1
4:      Insertar j en la posición que tenga el mínimo
        fitness1
5:      for j2 in π_r excepto los trabajos
        insertados de π_d:
6:          Reinsertar j2 en cada posición de π_r y
            medir el makespan como fitness2
7:          if (fitness2 < fitness1):
8:              Insertar j2 en la posición con el
                menor fitness2
9:      π_desrec = π_r
10:
    return π_desrec
```

El algoritmo 3 es el algoritmo de búsqueda local con multi-vecindad. En este algoritmo se toma una secuencia a ser procesada π_{input}, y el algoritmo retorna una secuencia factible mejorada $\pi_{multivec}$. En el algoritmo se le hacen dos tipos de operaciones a la secuencia: inserciones e intercambios de trabajos. Insertar un trabajo significa tomar un trabajo y ponerlo en otra posición, desplazando los demás trabajos hacia la derecha. Intercambiar dos trabajos im-

plica tomar dos trabajos y cambiarlos de posición, el uno en la posición del otro. Sea π_{modif} la secuencia tras ser modificada con alguna de estas operaciones, y π_{temp} una secuencia para guardar un resultado temporal. Para este algoritmo, sea $C_{max}(\pi) = F$ para dejar a F en función de la secuencia π. C_{max} es un símbolo común para el makespan. [2]

En el algoritmo de búsqueda local con multi-vecindad hay dos funciones clave, que realizan las operaciones de inserción y de intercambio de trabajos. La operación de inserción es realizada con la función $Insertion_inner(\pi)$, en esta función se toma cada trabajo de π y se inserta en cada posición, midiendo el makespan y al final dejando insertado el trabajo en la posición que tenga el menor makespan. La otra función es $Swap_inner(\pi)$ en la que se intercambian dos trabajos de π, se mide el makespan, y al final dejando intercambiados los trabajos en donde se presente el menor makespan. Si después de aplicar estas funciones se encuentra mejora, la función vuelve a ser aplicada, hasta que no se presente mejora en el makespan. [2]

Con lo anterior, aquí el algoritmo de búsqueda local con multi-vecindad en su versión de pseudocódigo de Python:

Algoritmo 3 Búsqueda local con multi-vecindad

```
def local_search(π_input):
1:  π_modif = π_input, l_max = 2, l = 1
2:  while (l <= l_max):
3:      if (l == 1):
4:          π_temp = Insertion_inner(π_modif)
5:      else if (l == 2):
6:          π_temp = Swap_inner(π_modif)
7:      if (C_max(π_temp) < C_max(π_modif)):
8:          π_modif = π_temp, l = 1
9:      else:
10:         l = l + 1
11: π_multivec = π_modif
12:
    return π_multivec
```

Teniendo los tres primeros algoritmos, el cuarto y último es la síntesis de los tres más algunas adiciones, por ello es llamado el algoritmo MNIG. Sea π_{result} la secuencia final resultado de la aplicación del algoritmo MNIG. Para aplicar el algoritmo MNIG es necesario definir un criterio de terminación del algoritmo, pues en caso contrario el algoritmo nunca pararía de ejecutarse. En la literatura el criterio de terminación propuesto (ajustado para el modelo FMMSP) es $N^2 * L * 0.1ms$ [2].

Este es un buen criterio, pero tiene dos problemas, primero no tiene en cuenta todas las partes del modelo FMMSP y segundo está en unidades de milisegundo. Para el primer problema, se puede incluir lo que falta del modelo FMMSP, las unidades (o máquinas). Sea $UT = \sum_{s=1}^{L} |U_s|$, UT es el total de unidades. Lo segundo puede ser un problema porque si el algoritmo para según una cantidad de tiempo arbitraria en milisegundos, ello fuerza a escribir el código de modo que cada iteración dure el menor tiempo posible. Pero eso es una desventaja porque no se puede hacer que el algoritmo dé resultados parciales, pues imprimir resultados parciales toma tiempo, a pesar de que sea poco tiempo. Otra forma de hacerlo sería teniendo un criterio de terminación en número de iteraciones, o sea que el algoritmo pare luego de un número de iteraciones. Según lo anterior, un criterio de terminación para el algoritmo MNIG aplicado al modelo FMMSP podría ser $Iterations = N^2 * L * UT$.

Antes de proceder con el pseudocódigo del algoritmo MNIG falta decir que este algoritmo explora en la región de búsqueda, intentando no quedarse en óptimos locales. A este efecto, existe un criterio de aceptación de nuevas soluciones que incluye la posibilidad de aceptar soluciones peores con una baja probabilidad. Este criterio define una nueva variable T, cuya ecuación es $T = T_0 * \frac{\sum_i \sum_s \widetilde{Ta}_{i,s}}{10*N*L}$ Donde T_0 es un parámetro diferente de zero y no muy grande. [2]

Pseudocódigo de Python del algoritmo MNIG:

Algoritmo 4 MNIG

```
def MNIG(todas las T̃_{i,u}, d, T_0):
1:  π_re3 = DNEH_SMR(todas las T̃_{i,u})
2:  π_result = π_re3, π_temp = π_re3, iter = 1
3:  while (iter <= N^2 * L * UT):
4:      π_temp = local_search(π_temp)
5:      if (C_max(π_temp) < C_max(π_re3)):
6:          π_re3 = π_temp
7:          if (C_max(π_temp) < C_max(π_result)):
8:              π_result = π_temp
9:      else:
10:         if (random(0,1) < e^{-(C_max(π_temp)-C_max(π_re3))/T}):
11:             π_re3 = π_temp
12:     π_temp = destruction_reconstruction(π_temp, d)
13:
    return π_result
```

Esto termina la presentación del modelo FMMSP y del algoritmo MNIG.

Con este marco de referencia se puede llevar a cabo el proyecto de grado.

8 HIPÓTESIS

El algoritmo MNIG puede ser aplicado al modelo FMMSP.

9 DISEÑO METODOLÓGICO

El diseño metodológico de este proyecto abarca tres partes principales: escribir el algoritmo MNIG en Python, ejecutar el algoritmo MNIG en una instancia del modelo FMMSP, y comparar los resultados del algoritmo MNIG con los resultados de los otros cuatro algoritmos que ya han sido aplicados al modelo FMMSP.

9.1 Escritura del algoritmo MNIG en Python

El algoritmo MNIG se puede escribir en el lenguaje Python como una aplicación de consola. El programa empieza por leer un archivo de texto plano en que se encuentren los parámetros del modelo FMMSP, en particular los tiempos de procesamiento, el número de etapas, el número de unidades por etapa, el número de trabajos a remover en el algoritmo de destrucción y reconstrucción d, y el parámetro T_0. Se prefiere tener estos datos en un archivo de texto plano para que se pueda repetir el algoritmo rápidamente, y si hay que hacer cambios a los datos se hacen independientemente del programa.

Una división lógica del programa en Python resultaría de poner los archivos en una sola carpeta, y crear un archivo de Python por cada algoritmo separado. Una posible jerarquía de archivos resultante sería:

```
MNIG_to_FMMSP/
├── __main__.py
└── algorithms/
    ├── __init__.py
    ├── DNEH_SMR.py
    ├── destruction_reconstruction.py
    └── local_search.py
```

En esta estructura, cada algoritmo tiene su propio archivo, y el archivo que tiene el algoritmo MNIG como síntesis de los demás es el archivo ___main___.py desde este archivo se llama a los demás algoritmos.

Durante la ejecución, el programa puede ir imprimiendo en la terminal resultados parciales de cada algoritmo, por ejemplo las secuencias que van resultando de cada algoritmo. Al finalizar la ejecución del programa, debe quedar impreso en pantalla la secuencia final así como su makespan. El hecho de imprimir por

pantalla no representa ninguna limitación, pues este resultado puede ser redireccionado a un archivo, de modo que los resultados queden en un archivo en disco duro, siendo el archivo elección del usuario.

Como se observa en la estructura de archivos, la carpeta algorithms/ es tratada como un paquete mediante el archivo ___init___.py se podría decir que es el paquete con los algoritmos necesarios para ejecutar el algoritmo principal MNIG.

9.2 Ejecución del algoritmo MNIG en instancia del modelo FMMSP

Aquí se explica por fin por qué sólo se trabajará una instancia y no varias del modelo FMMSP. La razón principalmente es que sólo existe una instancia públicamente a nivel internacional. Ello porque aunque en la literatura se habla de varias instancias (de unas 15 instancias), sólo en 1 se dan los datos, en el resto se dice que fueron generadas con números aleatorios uniformes con ciertas condiciones, pero instancias creadas aleatoriamente no pueden ser replicadas con exactitud. [1]

Por lo anterior se ejecutará el algoritmo MNIG en la única instancia conocida del modelo FMMSP. En la literatura las instancias son nombradas con 3 letras y un número por letra, por ejemplo la instancia que se trabajará es "o10s2u5". En esta cadena de caracteres, "o" significa "ordenes" (o trabajos), "s" significa etapas (stages en inglés), "u" significa "unidades" (o máquinas). Por ende la instancia sobre la que se probará el algoritmo es de 10 trabajos, 2 etapas, y 5 unidades. La siguiente tabla contiene los datos de la instancia: [1]

Table 1: Tiempos de procesamiento de la instancia o10s2u5

Trabajo	Unidad 1	Unidad 2	Unidad 3	Unidad 4	Unidad 5
	Etapa 1		Etapa 2		
1	(10,12,13)	(11,12,14)	(9,10,12)	(6,7,9)	(8,9,10)
2	(7,8,10)	(8,9,10)	(5,6,8)	(4,5,6)	(4,5,6)
3	(10,11,12)	(9,10,12)	(2,3,4)	(5,6,8)	(5,6,7)
4	(8,9,10)	(6,7,8)	(7,8,9)	(4,5,6)	(5,6,8)
5	(6,7,8)	(8,9,10)	(4,5,6)	(6,7,8)	(6,7,9)
6	(4,5,6)	(2,3,4)	(15,16,19)	(13,14,15)	(15,16,20)
7	(11,13,15)	(1,2,3)	(11,13,14)	(10,11,13)	(9,10,12)
8	(10,11,12)	(18,19,23)	(5,6,7)	(6,7,9)	(7,8,10)
9	(5,6,8)	(4,5,6)	(14,15,16)	(19,21,25)	(10,12,13)
10	(15,17,20)	(12,14,15)	(16,17,20)	(17,18,21)	(18,19,21)

Recordar que los tiempos de procesamiento vienen en ternas en la tabla

pues son números difusos triangulares. El algoritmo será ejecutado sobre esta instancia o10s2u5 y se guardará el resultado en un archivo para compararlo con los de los otros cuatro algoritmos.

9.3 Comparación del algoritmo MNIG con otros algoritmos

Tras aplicar el algoritmo MNIG a la instancia del modelo FMMSP y obtener resultados, se podrá comparar a otros cuatro algoritmos. Estos son: DBSA-LS, BDBSA, MBSA, e IGA. Estos cuatro algoritmos ya fueron aplicados a la instancia del modelo FMMSP. En la siguiente tabla se presentan los resultados de esos cuatro algoritmos: [1]

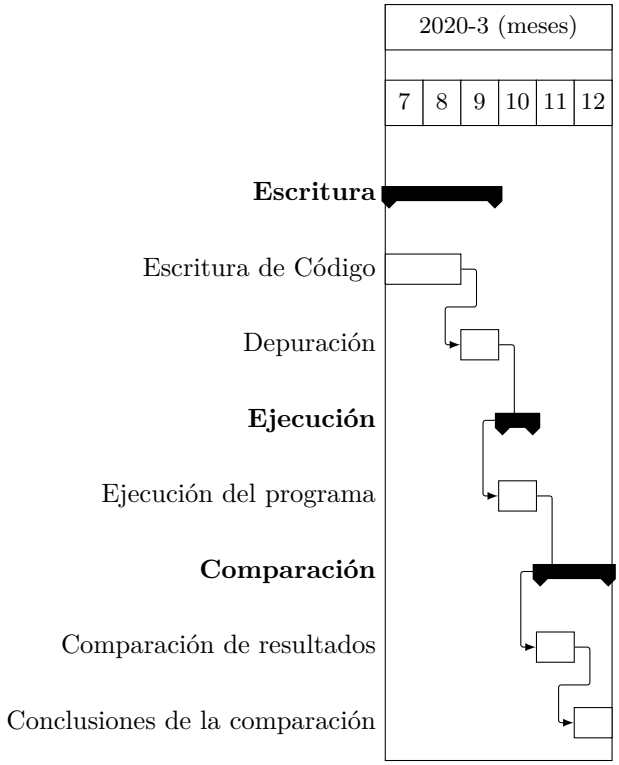
Table 2: Resultados del makespan de los cuatro algoritmos

Instancia	Algoritmo	Mejor valor	Valor medio	Peor valor
o10s2u5	DBSA-LS	(36,44,52)	(38,45,52)	(42,48,53)
	BDBSA	(36,44,52)	(38.6,45.5,52)	(40,46,54)
	MBSA	(40,47,55)	(44,50,57)	(42.2,49.3,54.6)
	IGA	(39,45,52)	(40.1,46.9,54.5)	(43,49,57)

Con esta tabla se compararán directamente los resultados del algoritmo MNIG al ser aplicado al modelo FMMSP.

10 CRONOGRAMA

El cronograma a continuación se basa en el diseño metodológico expuesto.



Del cronograma se observan los meses numerados en la parte superior, 7 es Julio, 8 es Agosto, etcétera. Se estima que el proyecto dure 6 meses o menos. Cada grupo de actividades está basado en una parte del diseño metodológico: escritura, ejecución, y comparación.

11 PRODUCTOS DEL PROYECTO

El producto principal del proyecto será la tesis de pregrado en ingeniería industrial, junto con su respectiva sustentación.

Aparte de lo anterior, opcionalmente se podrá escribir un paper en una revista indexada internacional, por ejemplo a través del evento Workshop on Engineering Applications. Sin embargo teniendo en cuenta los costos y fechas del paper puede que esto no se lleve a cabo durante el proceso de grado, y en ese caso se podría hacer después del grado cuando se presenten las condiciones; por ello se insiste en que este producto es opcional como parte del proyecto de grado.

12 BIBLIOGRAFÍA

[1] Xueli Yan, Yuxin Han, and Xingsheng Gu. "An improved discrete backtracking searching algorithm for fuzzy multiproduct multistage scheduling problem". In: *Neurocomputing* (2020). DOI: https://doi.org/10.1016/j.neucom.2020.02.066.

[2] Weishi Shao, Zhongshi Shao, and Dechang Pi. "Modeling and multi neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem". In: *Knowledge-Based Systems* (2020). DOI: https://doi.org/10.1016/j.knosys.2020.105527.

[3] Michael Pinedo. *Scheduling, Theory, Algorithms, and Systems*. 5th ed. New York, USA: Springer, 2016. ISBN: 978-3-319-26578-0.

[4] El-Ghazali Talbi. *Metaheuristics, From Design to Implementation*. 1st ed. USA: Wiley, 2009. ISBN: 978-0-470-27858-1.

[5] M. Garey, D. Jhonson, and R. Sethi. "The complexity of flowshop and jobshop scheduling". In: *Mathematics of Operations Research* 1.2 (1976), pp. 117,129.

[6] M. Clement Anand and Janani Bharatraj. "Theory of Triangular Fuzzy Number". In: Mar. 2017.