

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

Trabajo para optar al título de:  
INGENIERA INDUSTRIAL

Tatiana Porras Cortés 20092015073  
Correo: tatiporras96@gmail.com

LINDSAY ÁLVAREZ POMAR  
Directora 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. 2021-05-30

### TÍTULO

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

### RESUMEN

El Algoritmo Voraz Iterativo con Multi-Vecindad (MNIG por sus siglas en inglés Multi-Neighborhood Iterated Greedy algorithm) aplicado al Problema de Secuenciación Difuso Multiproducto y Multi-etapas (FMMSp por sus siglas en inglés Fuzzy Multiproduct Multistage Scheduling Problem), no ha sido tratado en la literatura científica internacional. En este trabajo se exploró esta combinación nueva. Para ello, se modificó el algoritmo MNIG y se utilizó para resolver el modelo FMMSp. El modelo FMMSp en notación de scheduling es el modelo  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ .

Se implementó el algoritmo MNIG en Python, y al aplicarlo al modelo FMMSP se encontró que el algoritmo MNIG es al menos tan bueno como el algoritmo DBSA-LS, e inclusive mostró mejora en la solución promedio. Esto se comprobó con la única instancia disponible que publicaron los creadores del modelo FMMSP.

**Palabras Clave:** Algoritmo Voraz Iterativo, Multi-vecindad, Secuenciación, Números Triangulares Difusos, Multiproducto, Multietapas.

## ABSTRACT

The Multi-neighborhood Iterated Greedy algorithm (acronymized as MNIG) applied to the Fuzzy Multiproduct Multistage Scheduling Problem (acronymized as FMMSP), has not been treated in the international scientific literature. This work explored this new combination. For that, the MNIG algorithm was modified and used to solve the FMMSP model. In scheduling notation, the FMMSP model is  $FFc|\tilde{T}_{i,u}, batch(b), pmu, Z_{i,u,s}|C_{max}$ .

The MNIG algorithm was implemented in Python, and by applying it to the FMMSP model, it was found that the MNIG algorithm is at least as good as the DBSA-LS algorithm, and it even showed an improvement in the average solution value. This was proven with the only available instance that was published by the creators of the FMMSP model.

**Keywords:** Iterated Greedy Algorithm, Multi-neighborhood, Scheduling, Triangular Fuzzy Numbers, Multiproduct, Multistage.

## 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 (Yan, Han, and Gu 2020) es llamado FMMSP que significa Fuzzy Multiproduct Multistage Scheduling Problem, que en español se podría traducir como Problema de Secuenciación Difuso Multiproducto y Multietapas, y su representación en notación de scheduling es  $FFc|\tilde{T}_{i,u}, batch(b), pmu, Z_{i,u,s}|C_{max}$ .

Es un modelo interesante de ser estudiado pues incluye números difusos, múltiples productos, y múltiples etapas. El algoritmo encontrado (Shao, Shao, and Pi 2020) 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 (Shao, Shao, and

Pi 2020).

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, estos cuatro algoritmos son llamados: DBSA-LS, BDBSA, MBSA, e IGA (Yan, Han, and Gu 2020).

## 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  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$  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 encontrar 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.

## Elementos del Problema

Según lo anterior, los elementos del problema son:

- Se requieren secuencias con bajos tiempos de terminación en el modelo FMMSP, denotado como  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ . Estas secuencias son llamadas ‘soluciones.’
- El algoritmo MNIG tiene potencial para encontrar dichas soluciones en un tiempo prudente. Los resultados se pueden comparar a otros cuatro algoritmos que ya han sido usados en el modelo FMMSP

## Pregunta del Problema

¿Cómo puede el algoritmo MNIG encontrar nuevas soluciones para el modelo FMMSP?

## Subpreguntas del Problema

¿Cómo puede implementarse el algoritmo MNIG, para obtener resultados del modelo FMMSP, denotado como  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ ?

¿De qué forma puede la implementación del algoritmo aportar resultados útiles, como por ejemplo, una secuencia de trabajos?

¿Cómo se comparan los resultados del algoritmo con los resultados de los otros cuatro algoritmos?

## OBJETIVOS

### Objetivo General

- Encontrar nuevas soluciones a través del algoritmo MNIG para el modelo FMMSP.

### Objetivos Específicos

- Implementar el algoritmo MNIG de modo que se puedan obtener resultados del modelo FMMSP, denotado como  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ .
- Obtener secuencias del modelo FMMSP usando la implementación realizada.
- Comparar el algoritmo MNIG aplicado al modelo FMMSP con los otros cuatro algoritmos que se han aplicado al modelo FMMSP.
- Establecer si tiene sentido o no aplicar el algoritmo MNIG al modelo FMMSP, o si es preferible buscar otros algoritmos incluidos los que ya han sido probados.
- Determinar la efectividad del algoritmo MNIG, no solo respecto al modelo FMMSP sino también respecto al modelo flow shop general.

## 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  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$  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.

**¿Cuál es el problema de investigación?:** El problema es que en toda la historia registrada de la humanidad, nunca se ha aplicado el algoritmo MNIG al modelo de scheduling FMMSP (para ello se revisaron las bases de datos internacionales a que tiene acceso la Universidad Distrital, y hasta la fecha no se ha resuelto este problema). Se podría pensar que cualquier metaheurística que existe se puede aplicar a cualquier modelo de scheduling, sin embargo esto no se puede asumir como cierto hasta que no se demuestre, y nunca se ha demostrado que el algoritmo MNIG sea competente para encontrar soluciones del modelo FMMSP.

Pero, ¿Qué se entiende al decir que un algoritmo es competente o no? En este contexto, se entiende que un algoritmo es competente si produce soluciones de calidad en un tiempo razonable. Para saber si las soluciones son de calidad, se compara aquí el algoritmo MNIG con otros cuatro algoritmos DBSA-LS, BDBSA, MBSA, e IGA (estos se explican en la sección del marco de referencia), los cuales tienen comprobada efectividad para encontrar soluciones en el modelo FMMSP.

**¿Cómo ha sido resuelto este problema de scheduling?:** El modelo FMMSP ha sido resuelto usando cuatro algoritmos, específicamente los algoritmos DBSA-LS, BDBSA, MBSA, e IGA. Hasta donde se sabe (tras hacer una revisión profunda de las bases de datos internacionales a las que tiene acceso la Universidad Distrital), estos son los únicos cuatro algoritmos con que se ha solucionado el modelo FMMSP. Por ello el

algoritmo MNIG se comparará con estos cuatro algoritmos.

**¿Cuál es el aporte a la solución de éste problema de scheduling en particular?:** En este trabajo, el aporte a la solución del modelo FMMSP, es aportar un nuevo algoritmo que tenga comprobada efectividad para solucionar el modelo FMMSP. Este nuevo algoritmo sería el algoritmo MNIG. Debido a que este trabajo aún no se ha aprobado ni realizado, cabe la posibilidad de que el algoritmo MNIG aporte mejores soluciones al modelo FMMSP, es decir que el máximo de los tiempos de terminación triangulares difusos sea menor gracias al algoritmo MNIG, esto debido a que los cuatro algoritmos, DBSA-LS, BDBSA, MBSA, e IGA, NO son algoritmos de optimización sino metaheurísticas, entonces cabe la posibilidad de que el algoritmo MNIG encuentre aún mejores soluciones en el espacio de búsqueda (esto es explicado con más profundidad en la sección del marco de referencia). Esto se probará con la única instancia pública internacional que existe del modelo FMMSP (desgraciadamente los autores del modelo FMMSP no publicaron más instancias. En todo caso se probará el algoritmo MNIG con instancias de Taillard).

**¿Por qué hay que solucionar este problema?:** Porque nunca se ha hecho y porque hay un interés científico por avanzar nuestro conocimiento sobre los límites que unen a las metaheurísticas y los modelos de scheduling. Debido a la naturaleza NP-Hard de los modelos derivados de flow shop (como lo es el modelo FMMSP), aún no existe forma de garantizar la optimalidad de las soluciones encontradas por las metaheurísticas, y por eso existe cierta competencia entre diferentes metaheurísticas que usan diferentes estrategias para buscar soluciones en el espacio de búsqueda. Esta competencia sigue abierta para el modelo FMMSP, y aunque el algoritmo MNIG fue diseñado para problemas muy diferentes, en este trabajo se ha logrado adaptar el algoritmo MNIG para tratar el modelo FMMSP y encontrar soluciones al mismo.

**¿Qué sucede si no solucionamos este problema identificado?:** Si no solucionamos este problema, no avanzará nuestro conocimiento sobre el modelo FMMSP y sus soluciones. El modelo FMMSP es un modelo que se puede encontrar en sistemas de producción reales, nada más el hecho de que el modelo FMMSP incluye números difusos triangulares (F es por Fuzzy), muestra que el modelo FMMSP puede ser aplicado a sistemas de producción colombianos, en donde no hay una fuerte estandarización, porque si las estaciones de trabajo tienen tiempos de procesamiento fuertemente estandarizados, ya no se hacen necesarios los números triangulares difusos, y resulta más fácil y razonable usar simples números para representar los tiempos de procesamiento. En cambio, en sistemas donde no hay una fuerte estandarización, puede tener sentido usar números triangulares difusos para representar los tiempos de producción de las estaciones de trabajo, porque así se representa la incertidumbre sobre el verdadero tiempo de producción, usando las tres partes que forman un número triangular difuso: el tiempo optimista de producción, el tiempo promedio de producción, y el tiempo pesimista de producción (esto se puede ver también en la sección del marco de referencia).

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.

## ALCANCES Y LIMITACIONES DEL PROYECTO

**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 (DBSA-LS, BDBSA, MBSA, e IGA) aplicados al modelo FMMSP  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ . 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 preestablecido, entonces si las fechas de la graduación no cuadran con las fechas del paper, se publica el paper después de la graduación.

## MARCO DE REFERENCIA

El marco de referencia se enfoca en el scheduling de los sistemas de producción tipo FMSP, en el algoritmo MNIG que es un tipo de metaheurística, y en los otros cuatro algoritmos que son DBSA-LS, BDBSA, MBSA, e IGA.

## Antecedentes y Marco Teórico

En los antecedentes encontramos dos temas pilares: el scheduling, y las metaheurísticas. Del libro de scheduling de Pinedo (Pinedo 2016) 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. (Pinedo 2016)

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 común 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. (Pinedo 2016)

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. (Talbi 2009)

La notación del modelo FMSP:  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ , fue creada con base en la notación tradicional de scheduling de Pinedo. (Pinedo 2016)

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 (Talbi 2009). 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 FMSP  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$  se puede adaptar fácilmente sin cambiar la esencia de la metaheurística.

## 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, la 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. (Pinedo 2016)

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. (Pinedo 2016)

**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. (Talbi 2009)

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. (Talbi 2009)

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  $n$ . (Talbi 2009)

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. Lo anterior requiere que para estos modelos de scheduling se usen metaheurísticas para su resolución. (Garey, Jhonson, and Sethi 1976)

**Modelo FMSP:** El modelo FMSP, denotado como  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ , 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 FMSP 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. (Yan, Han, and Gu 2020)

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. (Yan, Han, and Gu 2020)

Los números difusos triangulares se definen de la siguiente forma: (Anand and Bharatraj 2017)

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:

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

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  $c$  descende de 1 a 0, lo que hace que efectivamente la función forme un triángulo que inicia en  $a$ , tiene el máximo en  $b$  y termina en  $c$ . (Anand and Bharatraj 2017)

Haciendo uso de los números difusos triangulares, a continuación se define el modelo FMMSP  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ : (Yan, Han, and Gu 2020)

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

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



El tiempo de procesamiento del trabajo  $i$  en la unidad  $u$  es un número difuso triangular, por lo que se puede representar como:  $\tilde{T}_{i,u} = (T_{i,u}^1, T_{i,u}^2, T_{i,u}^3)$ . Notar que  $u$  es cualquier unidad de todas las etapas, y no de una etapa en particular, es decir si se numeran 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  $\tilde{T}s_{i,s}$ , 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 es el siguiente:

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

$$\sum_{u \in U_s} Z_{i,u,s} = 1 \quad \forall i \in I, s \in S \quad (3)$$

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

$$\widetilde{Tf}_{i,s} + M * (2 - Z_{i,u,s} - Z_{j,u,s}) \leq \tilde{T}s_{j,s} \quad \forall i \in I, j \in I, i < j, s \in S, u \in U_s \quad (5)$$

$$\widetilde{Tf}_{i,s} \leq \tilde{T}s_{i,s+1} \quad \forall i \in I, s \in S \quad (6)$$

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  $s$ . (Yan, Han, and Gu 2020)

Teniendo definido el modelo  $FFc[\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s} | C_{max}$ , 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: (Yan, Han, and Gu 2020)

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 casos siguientes

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

**Algoritmo MNIG:** El algoritmo MNIG se entiende dividido a su vez en cuatro algoritmos, cada uno realizando un rol diferente y al ser aplicados como se indica logran una solución. (Shao, Shao, and Pi 2020)

El primer algoritmo es llamado DNEH\_SMR, sirve para obtener una secuencia de trabajos inicial promisorio 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. (Shao, Shao, and Pi 2020)

El algoritmo MNIG usa su propia notación, que en ocasiones usa los mismos símbolos del modelo FMMSF  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ . En este trabajo se le dará prelación a los símbolos del modelo FMMSF, 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 FMMSF.

Para entender el algoritmo DNEH\_SMR se requieren algunos símbolos extra. Una secuencia de trabajos factible se denota como  $\pi$ , entonces  $\pi$  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  $k$ . (Shao, Shao, and Pi 2020)

En el modelo FMMSF  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$  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  $\tilde{Ta}_{i,s} = \frac{1}{n_s} * \sum_{u \in U_s} \tilde{T}_{i,u}$ .  $Ta$  es por *Taverage*. Esta modificación es válida porque en cada etapa las máquinas trabajan en paralelo, es decir, cuando un trabajo pasa por una etapa, solamente es procesado por una máquina, no por todas las máquinas, entonces el tiempo promedio por etapa representa el tiempo que en promedio pasan los trabajos en cada etapa.

Teniendo  $\tilde{Ta}$  se definen los tiempos de procesamiento promedio por grupo de etapas, para el primer grupo antes de  $k$ ,  $\tilde{T}'_{i,1} = \frac{1}{k} * \sum_{l=1}^k \tilde{Ta}_{i,l}$ , y para el segundo grupo después de  $k$ ,  $\tilde{T}'_{i,2} = \frac{1}{L-k} * \sum_{l=k+1}^L \tilde{Ta}_{i,l}$ . Nótese el símbolo prima para indicar que son los tiempos promedio de los grupos de etapas. (Shao, Shao, and Pi 2020)

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  $\tilde{T}'_{i,2}$ , sea dicha secuencia  $\pi_{re1}$  donde **re1** indica que es el primer reordenamiento de la secuencia. Para el segundo reordenamiento,  $\pi_{re2}$  se consigue al tomar el primer trabajo y luego el trabajo de en medio de  $\pi_{re1}$ , eliminando los trabajos tomados de  $\pi_{re1}$ . 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  $\pi_{re2}$ . (Shao, Shao, and Pi 2020)

Dado que el modelo FMMSp es un modelo para sistemas del tipo flow shop flexible, cada trabajo sigue una secuencia de máquinas, es decir, en este modelo cada solución es una secuencia de trabajos en donde cada trabajo tiene una secuencia de máquinas. Para elegir la secuencia de máquinas de un trabajo, en este documento se usará la regla de asignar un trabajo a la máquina que lo procese más pronto, incluyendo la posibilidad de esperar a que un trabajo anterior sea procesado en una máquina y que se asigne el siguiente trabajo a esa máquina, según lo que resulte en el menor tiempo final de procesamiento. Esto se representa en la notación del modelo con:  $Z_{i,u,s}$ , porque el uso de la variable  $Z_{i,u,s}$  en el modelo FMMSp muestra que es posible procesar dos trabajos diferentes al mismo tiempo en una misma etapa, usando dos máquinas diferentes.

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  $\pi_{re3}$  la secuencia resultante de estas reinserciones. El algoritmo DNEH\_SMR retorna la secuencia  $\pi_{re3}$  como su resultado. (Shao, Shao, and Pi 2020)

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  $\tilde{T}_{i,u}$ ):

1: Computar  $k = \lfloor \frac{L}{2} \rfloor$

2: Computar  $\widetilde{T}a_{i,s} = \frac{1}{n_s} * \sum_{u \in U_s} \tilde{T}_{i,u} \forall i \in I, s \in S$

3: Computar  $\tilde{T}'_{i,1} = \frac{1}{k} * \sum_{l=1}^k \widetilde{T}a_{i,l}$ , y  $\tilde{T}'_{i,2} = \frac{1}{L-k} * \sum_{l=k+1}^L \widetilde{T}a_{i,l} \forall i \in I$

4: Ordenar los trabajos según el orden ascendente de  $\tilde{T}'_{i,2}$ , guardando la secuencia en  $\pi_{re1}$

5: Crear la secuencia  $\pi_{re2}$ , quitando el primer trabajo y luego el de en medio de  $\pi_{re1}$ , y ponerlos en  $\pi_{re2}$ , repetir con los trabajos restantes hasta que ya no queden trabajos en  $\pi_{re1}$

6: Copiar  $\pi_{re2}$  en  $\pi_{re3}$

```

7: for j in  $\pi_{re3}$ :

8: Reinsertar j en cada posición de  $\pi_{re3}$  y medir el makespan

9: Dejar j en la posición con el mínimo makespan sin mover los trabajos de iteraciones anteriores

10: return  $\pi_{re3}$ 

```

---

Por claridad en el paso 4 del algoritmo DNEH\_SMR, si por ejemplo la secuencia  $\pi_{re1}$  es la secuencia  $[1, 6, 2, 5, 4, 3]$ , la secuencia  $\pi_{re2}$  correspondiente será entonces  $[1, 5, 6, 4, 2, 3]$ . (Shao, Shao, and Pi 2020)

Con el algoritmo DNEH\_SMR se obtiene una solución inicial  $\pi_{re3}$ . 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  $\pi_{re3}$  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. (Shao, Shao, and Pi 2020)

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  $\pi_d$ . Con los trabajos restantes se forma la secuencia  $\pi_r$ . Luego se toma cada trabajo en  $\pi_d$  y se reinserta en cada posición de  $\pi_r$ , 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. (Shao, Shao, and Pi 2020)

Luego cada trabajo en  $\pi_r$  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  $\pi_r$  sin reemplazar los que ya hayan sido insertados. Por último se repite este proceso con los demás trabajos en  $\pi_d$ . La secuencia resultante es retornada por el algoritmo. (Shao, Shao, and Pi 2020)

Llámesele  $\pi_{input}$  a la secuencia a ser destruida y reconstruida, y sea  $\pi_{desrec}$  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( $\pi_{input}, d$ ):

```

- 1: Remover aleatoriamente  $d$  trabajos de la secuencia  $\pi_{input}$  y ponerlos en la secuencia  $\pi_d$ .  
Poner el resto de trabajos en la secuencia  $\pi_r$
- 2: for j in  $\pi_d$ :
- 3: Reinsertar j en cada posición de  $\pi_r$  y medir el makespan como  $fitness1$
- 4: Insertar j en la posición que tenga el mínimo  $fitness1$
- 5: for j2 in  $\pi_r$  excepto los trabajos insertados de  $\pi_d$ :
- 6: Reinsertar j2 en cada posición de  $\pi_r$  y medir el makespan como  $fitness2$
- 7: if ( $fitness2 < fitness1$ ):
- 8: Insertar j2 en la posición con el menor  $fitness2$
- 9:  $\pi_{desrec} = \pi_r$
- 10: return  $\pi_{desrec}$

---

El algoritmo 3 es el algoritmo de búsqueda local con multi-vecindad. En este algoritmo se toma una secuencia a ser procesada  $\pi_{input}$ , 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 implica tomar dos trabajos y cambiarlos de posición, el uno en la posición del otro. Sea  $\pi_{modif}$  la secuencia tras ser modificada con alguna de estas operaciones, y  $\pi_{temp}$  una secuencia para guardar un resultado temporal. Para este algoritmo, sea  $C_{max}(\pi) = F$  para dejar a  $F$  en función de la secuencia  $\pi$ .  $C_{max}$  es un símbolo común para el makespan. (Shao, Shao, and Pi 2020)

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  $\pi$  y se inserta en cada posición, midiendo el makespan y al final, si el makespan mejora, se deja 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  $\pi$ , 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. (Shao, Shao, and Pi 2020)

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( $\pi_{input}$ ):

1:   $|\pi_{modif} = \pi_{input}, l_{max} = 2, l = 1$ 

2:  while ( $l \leq l_{max}$ ):

3:      if ( $l == 1$ ):

4:           $|\pi_{temp} = Insertion_{inner}(\pi_{modif})$ 

5:      else if ( $l == 2$ ):

6:           $|\pi_{temp} = Swap_{inner}(\pi_{modif})$ 

7:      if ( $C_{max}(\pi_{temp}) < C_{max}(\pi_{modif})$ ):

8:           $|\pi_{modif} = \pi_{temp}, l = 1$ 

9:      else:

10:          $l = l + 1$ 

11:  $|\pi_{multivec} = \pi_{modif}$ 

12: return  $\pi_{multivec}$ 

```

---

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  $\pi_{result}$  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 FMMSF  $FFc|\tilde{T}_{i,u}, batch(b), pmu, Z_{i,u,s}|C_{max}$ ) es  $N^2 * L * 0.1ms$  (Shao, Shao, and Pi 2020).

Este es un buen criterio, pero tiene dos problemas, primero no tiene en cuenta todas las partes del modelo FMMSF y segundo está en unidades de milisegundo. Para el primer problema, se puede incluir lo que falta del modelo FMMSF, 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 un nuevo parámetro derivado  $T$ , cuya ecuación es  $T = T_0 * \frac{\sum_i \sum_s \tilde{T}a_{i,s}}{10 * N * L}$ . Donde  $T_0$  es un parámetro diferente de cero y no muy grande. (Shao, Shao, and Pi 2020)

Pseudocódigo de Python del algoritmo MNIG:

---

**Algoritmo 4** MNIG

---

```
def MNIG(todas las  $\tilde{T}_{i,u}$ ,  $d$ ,  $T_0$ ):  
  
1:   $|\pi_{re3} = \text{DNEH\_SMR}(\text{todas las } \tilde{T}_{i,u})$   
  
2:   $|\pi_{result} = \pi_{re3}, \pi_{temp} = \pi_{re3}, iter = 1$   
  
3:  while ( $iter \leq N^2 * L * UT$ ):  
  
4:       $|\pi_{temp} = \text{local\_search}(\pi_{temp})$   
  
5:      if ( $C_{max}(\pi_{temp}) < C_{max}(\pi_{re3})$ ):  
  
6:           $|\pi_{re3} = \pi_{temp}$   
  
7:          if ( $C_{max}(\pi_{temp}) < C_{max}(\pi_{result})$ ):  
  
8:               $|\pi_{result} = \pi_{temp}$   
  
9:      else:  
  
10:         if ( $\text{random}(0,1) < e^{\frac{-(C_{max}(\pi_{temp}) - C_{max}(\pi_{re3}))}{T}}$ ):  
  
11:              $|\pi_{re3} = \pi_{temp}$   
  
12:          $|\pi_{temp} = \text{destruction\_reconstruction}(\pi_{temp}, d)$   
  
13: return  $\pi_{result}$ 
```

---

**Algoritmos BSA en general:** Los algoritmos BSA, por sus siglas en inglés que significan Backtracking Search Algorithm, son algoritmos que al ser aplicados a problemas de Scheduling, constan de 5 pasos:

inicialización, selección-I, mutación, cruzamiento, y selección-II. (Yan, Han, and Gu 2020)

La inicialización genera la población de secuencias. El paso selección-I actualiza la población de secuencias si es necesario, usando la memoria de la información histórica que guarda un algoritmo BSA. Dicha memoria también se usa para construir o cambiar la dirección de búsqueda de soluciones. El paso mutación genera una nueva población de secuencias mutando o alterando la población recibida del paso anterior. El paso cruzamiento cruza las secuencias unas con otras para obtener nuevas secuencias, descartando las secuencias no factibles. Por último, el paso de selección-II compara el makespan de las nuevas secuencias respecto a la población de secuencias del paso selección-I, y las mejores secuencias son elegidas para la iteración siguiente. (Yan, Han, and Gu 2020)

Tres de los cuatro algoritmos de comparación en este trabajo, son algoritmos de tipo BSA.

**Algoritmo DBSA-LS:** El algoritmo DBSA-LS, por sus siglas en inglés que significan Discrete Backtracking Search Algorithm with Local Search, es un algoritmo que busca aumentar la velocidad de convergencia del algoritmo BSA. Para ello, el algoritmo DBSA-LS utiliza la mejor secuencia de cada iteración, conocida como la secuencia Gbest. Aparte de esto, el algoritmo DBSA-LS utiliza búsqueda local para mejorar la de exploración de soluciones. (Yan, Han, and Gu 2020)

**Algoritmo BDBSA:** El algoritmo BDBSA, por sus siglas en inglés que significan Basic Discrete Backtracking Search Algorithm, es un algoritmo tipo BSA que aplica los mismos cinco pasos del algoritmo BSA a problemas de tiempo discreto. Este algoritmo puede ser aplicado a problemas de scheduling en los que los trabajos inician y terminan en unidades de tiempo discretos. (Yan, Han, and Gu 2020)

**Algoritmo MBSA:** El algoritmo MBSA, por sus siglas en inglés que significan Modified Backtracking Search Algorithm, es una forma modificada del algoritmo BSA. Esta modificación no cambia los cinco pasos esenciales del algoritmo BSA. (Yan, Han, and Gu 2020)

**Algoritmo IGA:** El algoritmo IGA, por sus siglas en inglés que significan Interactive Genetic Algorithm, es un algoritmo genético que tiene algunas similitudes con el algoritmo BSA. Por ejemplo, en el algoritmo IGA las secuencias son modificadas mediante mutaciones aleatorias, y las mejores secuencias según el makespan pasan a la siguiente iteración. Esto también ocurre en el algoritmo BSA. (Yan, Han, and Gu 2020)

Esto termina la presentación del modelo FMSP denotado como  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ , el algoritmo MNIG, y los cuatro algoritmos DBSA-LS, BDBSA, MBSA, e IGA. Con este marco de referencia se puede llevar a cabo el proyecto de grado.

## HIPÓTESIS

A través del algoritmo MNIG se pueden encontrar nuevas soluciones para el modelo FMSP  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ . Tras su aplicación la efectividad del algoritmo MNIG es medible y es comparable a la efectividad de los otros cuatro algoritmos: DBSA-LS, BDBSA, MBSA, e IGA.

## 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 FMSP, y comparar los resultados del algoritmo MNIG con los resultados de los otros cuatro algoritmos que ya han sido aplicados al modelo



FMMSP (DBSA-LS, BDBSA, MBSA, e IGA).

## 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  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ , 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.

## Ejecución del algoritmo MNIG en instancia del modelo FMMSP

Aquí se explica por qué sólo se trabajará una instancia y no varias del modelo FMMSP  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ . 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. (Yan, Han, and Gu 2020)

Por lo anterior se ejecutará el algoritmo MNIG en la única instancia conocida del modelo FMMSP  $FFc|\tilde{T}_{i,u}, batch(b), prmu, Z_{i,u,s}|C_{max}$ . 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: (Yan, Han, and Gu 2020)

Tiempos de procesamiento de la instancia o10s2u5

Trabajo	Unidad 1 Etapa 1	Unidad 2 Etapa 1	Unidad 3 Etapa 2	Unidad 4 Etapa 2	Unidad 5 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.

## Comparación del algoritmo MNIG con otros algoritmos

Tras aplicar el algoritmo MNIG a la instancia del modelo FMSP 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 FMSP. En la siguiente tabla se presentan los resultados de esos cuatro algoritmos:

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

## Instancias de Taillard

A nivel internacional, se reconocen algunas instancias con las que se puede medir la efectividad de las metaheurísticas. Para el caso flow shop, las instancias de Taillard son instancias que se han venido usando desde 1993 cuando fueron creadas, son en total 12 instancias. La más pequeña cuenta con 20 trabajos y 5 máquinas. La instancia más grande es de 500 trabajos y 20 máquinas. Se aplicará el algoritmo MNIG a estas instancias aunque no esté diseñado para resolver el modelo flow shop básico, pues esto permitirá comprobar que el algoritmo MNIG puede ser aplicado a sistemas tipo flow shop.

## RESULTADOS

## CONCLUSIONES Y RECOMENDACIONES

## GLOSARIO DE ANGLICISMOS

## REFERENCIAS

Anand, M. Clement, and Janani Bharatraj. 2017. "Theory of Triangular Fuzzy Number." In.  
Garey, M., D. Jhonson, and R. Sethi. 1976. "The Complexity of Flowshop and Jobshop Scheduling."

- Mathematics of Operations Research* 1 (2): 117, 129.
- Pinedo, Michael. 2016. *Scheduling, Theory, Algorithms, and Systems*. 5th ed. New York, USA: Springer.
- Shao, Weishi, Zhongshi Shao, and Dechang Pi. 2020. "Modeling and Multi Neighborhood Iterated Greedy Algorithm for Distributed Hybrid Flow Shop Scheduling Problem." *Knowledge-Based Systems*. <https://doi.org/https://doi.org/10.1016/j.knosys.2020.105527>.
- Taillard, E. 1993. "Benchmarks for Basic Scheduling Problems." *EJOR* 64 (2): 278, 285.
- Talbi, El-Ghazali. 2009. *Metaheuristics, From Design to Implementation*. 1st ed. USA: Wiley.
- Yan, Xueli, Yuxin Han, and Xingsheng Gu. 2020. "An Improved Discrete Backtracking Searching Algorithm for Fuzzy Multiproduct Multistage Scheduling Problem." *Neurocomputing*. <https://doi.org/https://doi.org/10.1016/j.neucom.2020.02.066>.