Revista Ingeniería e Innovación
ISSN: 2346-0474 (En línea)

Estudio comparativo entre el algoritmo búsqueda de lobos y otras metaheurísticas en el problema de una máquina con tiempos de alistamiento dependientes de la secuencia.

Comparative study between the wolf search algorithm and other metaheuristics for the scheduling single machine problem with sequence- dependent setup times

Carlos A. Mendoza A [1]*
Universidad Pontificia Bolivariana

German E. Perez[2].[2]
Universidad Pontificia Bolivariana

Cesar A. López[3]
Universidad Pontificia Bolivariana

Fecha recepción: 15 de enero de 2020
Fecha aceptación: 16 de mayo de 2020

RESUMEN

En este artículo se adaptó la metaheurística basada en el algoritmo de Búsqueda de Lobos para resolver el problema de programación en una sola máquina con tiempos de alistamientos dependientes de la secuencia. Para representar la solución del problema, se propuso un esquema de codificación discreta permutada. El desempeño

ABSTRACT

In this article, the Wolf Search algorithm was adapted to solve the problem a single machine with sequence-dependent setup times. To represent the solution of the problem, a permuted discrete coding scheme was proposed. The performance of the algorithm was compared with the Metaheuristic Viral System and Simulated Annealing in 35

[1] *Estudiante de Ingenieria Industrial, Facultad de Ingenieria Industrial, Universidad Pontificia Bolivariana, Montería – Colombia. E-mail: carlos.mendozaa@upb.edu.co.*
[2] *Estudiante de Ingenieria Industrial, Facultad de Ingenieria Industrial, Universidad Pontificia Bolivariana, Montería – Colombia. E-mail: german.perezh@upb.edu.co*
[3] *Ingeniero Industrial, Ms(c) Ingeniera De Producción, Docente Asistente en la facultad de Ingenieria Industrial, Universidad Pontificia Bolivariana, Cra. 6 #97A-99, of. 310, teléfono: (4) 7860146, Montería, Colombia. E-mail: cesar.lopezma@upb.edu.co.*

del algoritmo fue comparado con las metaheurísticas Sistema Viral y Recocido Simulado en 35 instancias con tamaños entre 15 hasta 150 trabajos. A partir de un Análisis De Variancia, se encontró que los algoritmos de Búsqueda de Lobos y Sistema Viral son similares en su desempeño en cuanto a calidad de respuesta, sin embargo, el algoritmo Búsqueda De Lobos presenta un bajo tiempo computacional frente al algoritmo Sistema Viral. Se concluye que el algoritmo de Búsqueda de Lobos puede ser usado con gran certeza para abordar el problema de programación de producción en una sola máquina con tiempos de alistamientos dependientes de la secuencia, para lo cual se propone que sea usado en otros problemas de programación de producción o de optimización combinatoria.

PALABRAS CLAVE: Secuenciamiento, Optimización Combinatoria, Algoritmo Búsqueda De Lobos, Recocido Simulado, Sistema Viral.

instances with sizes between 15 and 150 jobs. From a Variance Analysis, it was found that the Wolf Search and Viral System algorithms are similar in their performance in terms of response quality, however, the Wolf Search algorithm has a low computational time compared to the Viral System algorithm. It is concluded that the Wolf Search algorithm can be used with great certainty to address the problem of production programming on a single machine with sequence-dependent enlistment times, for which it is proposed that it be used in other programming problems of combinatorial production or optimization.

KEYWORDS: : Scheduling, Combinatory Optimization, Wolf Search Algorithm, Simulated Annealing, Viral System.

INTRODUCCION

En muchos contextos industriales, los tiempos de alistamientos entre uno y otro trabajo/lote dependen del orden en el que se vaya a realizar cada uno de ellos. Por ejemplo: el tiempo para alistar una maquina a partir de un producto A para fabricar luego fabricar un producto B, no es el mismo que tomaría primero producir el B y luego el A. Esta característica es denominada tiempos de alistamiento dependientes de la secuencia. Es muy común encontrar esta situación en contexto reales, por ejemplo, embotellamiento de bebidas gaseosas en una fábrica, atención de personal por parte de un módulo de servicio, ejecución de trabajos en un torno, entre otros.

En estas situaciones y para cuando se tengan muchas órdenes de trabajo, no es fácil determinar la secuencia en la que se deberían realizar las órdenes en el recurso disponible. Esto debido a la naturaleza de los tiempos de alistamiento, los cuales impactan directamente en el tiempo de finalización total de todos los trabajos disminuyéndolo o en el peor de los casos aumentándolo.

La solución de este tipo de problemas es compleja debido a su carácter combinatorio (Pinedo, 2012). Por ejemplo, si se parte del supuesto, donde se quiere programar una máquina que atiende la demanda de 20 trabajos u órdenes, las diferentes maneras de combinar dichos trabajos para ser procesados en el recurso ascienden a 20!, es decir 2,43x1018 formas de combinar la elaboración de los productos; a medida que aumente el número de órdenes, aumenta exponencialmente las posibles soluciones, lo que genera que al momento de encontrar una solución óptima, el tiempo computacional de búsqueda sea extensamente largo. Es de allí la importancia de esta investigación, donde el objeto de estudio que persigue es comparar diferentes métodos aproximados con el fin de reducir el tiempo de respuesta garantizando respuestas eficientes en cuanto la calidad de respuesta.

Para la solución de los problemas de este tipo se dispone de dos tipos de métodos: los métodos de solución exactos y los métodos de solución aproximados. Los métodos exactos se muestran ineficientes para problemas de gran tamaño, debido a que el tiempo computacional gastado para encontrar una solución óptima crece conforme el tamaño del problema aumenta. Sin embargo, dada la alta complejidad del problema; existen los métodos aproximados que se conciben como una alternativa para obtener buenas soluciones en menor tiempo computacional.

Dentro de estos métodos de solución aproximados, se encuentran las metaheurísticas, estas se consideran como estructuras algorítmicas que generalmente se aplican a una variedad de problemas de optimización con solo unas pocas modificaciones para adaptarse al problema dado (Abdel-Basset, Abdel-Fatah and Sangaiah, 2018). Por tal razón, se propone en este artículo, adaptar una metaheurística basada en el comportamiento de caza de los lobos a sus presas conocido como el algoritmo Búsqueda De Los Lobos o Wolf Search Algorithm (WSA) la cual fue propuesta por (Tang et al., 2012) para solucionar el problema de programación en una sola maquina con tiempos de alistamiento dependientes de la secuencia.

La adaptación de diferentes metaheurísticas a ciertos problemas de optimización es una actividad que ha venido desarrollándose con mucha fuerza en los últimos años. Debido a la alta complejidad de estos problemas, estas adaptaciones son un campo de investigación activo y de gran interés para la comunidad académica.

Es así, que en el trabajo presentado por (Osaba et al., 2018) utiliza un Recocido Simulado Evolutivo o Evolutionary Simulated Anneling (ESA) para comparar un nuevo algoritmo basado en el ciclo del agua. Esta comparación se realizó resolviendo 14 problemas para el TSP asimétrico, los cuales presentan un rango entre 17 y 124 nodos o ciudades. Para el método ESA, se fijó el parámetro de enfriamiento (α) a un nivel de 0.95. Se utilizó un operador de cruce 2-opt para generar las soluciones vecinas. Los experimentos computacionales arrojaron que se alcanzaron soluciones optimas en 5 problemas (tamaño 17, 33, 35, 38, 43 respectivamente) con un tiempo computacional promedio de 13.1 segundos.

Otra aplicación para el recocido simulado es la que propone (Jin, Song and Wu, 2009) en su trabajo; aquí se desarrolla un algoritmo de recocido simulado aplicado al problema de minimizar la demora máxima en una sola máquina en donde los tiempos de alistamiento se caracterizan por agruparse en familias de trabajo. Para este algoritmo, se utilizó una temperatura inicial de 100, una temperatura final de 1 y un parámetro de enfriamiento de (α) de 0.75. Las instancias fueron generadas aleatoriamente con trabajos desde 60, 70, 80, 90, 100. En total se evaluaron 2400 instancias frente a un algoritmo genético clásico. Los resultados obtenidos demuestran que el desempeño del algoritmo genético supera al del algoritmo genético utilizado.

Por su parte, en la literatura se evidencia la aplicación del algoritmo Viral System en problemas de una sola máquina. El trabajo presentado por (Santosa and Affandi, 2013), se adapta VS a un problema de una sola máquina para minimizar la tardanza ponderada total. Para verificar el desempeño del algoritmo, se resolvieron 125 instancias de 40 trabajos, 125 instancias de 50 trabajos y 25 instancias de 100 trabajos las cuales están disponibles en la OR-Library. Estos resultados fueron comparados con un recocido simulado multi etapa (SAmulti-s), un algoritmo de búsqueda tabu o Tabu Search (TB), un algoritmo búsqueda tabú multi etapa (TBmulti-s) y una búsqueda local iterada (ILS). Los resultados muestran que VS alcanza 125 óptimos para las 125 instancias de 40 trabajos, 101 óptimos para las 125 instancias de 50 trabajos y solo 9 óptimos para las 25 instancias de 100 trabajos. Para los problemas de tamaño 100, VS presentó un alto tiempo computacional.

El algoritmo Viral System también se ha aplicado a otros problemas de optimización. Es el caso del trabajo propuesto por (Suryadi and

Kartika, 2011), aplica este algoritmo al problema de la mochila o Knapsack Problem. Aquí se formulan tres casos con el fin de encontrar una calibración adecuada para los niveles de los parámetros del método. Los parámetros a analizar son: Probabilidad del Proceso Lítico (PLi), Probabilidad de Replica (Pr), Probabilidad de Generar Antígenos (Pa), Número máximo de núcleos replicados (LNR), Número máximo de núcleos mutados (LIT), Cada parámetro se fijaron dos niveles de la siguiente manera: Pli: 0.2-0.8, Pa: 0.2-0.8, Pr: 0.2-0.8, LNR: 5-25 y LIT: 5-25. Se realizaron 5 réplicas para el caso 1 y 2, para el caso 3 se realizaron 15 réplicas. Además se fijó el tamaño de la población inicial en 100 y el número de iteraciones en 10000. Para el caso 1, se tiene un problema de 4 ítems con una mochila de 14 kg de capacidad. En este caso, se encontró que los parámetros no presentan influencia en la variable de respuesta. En el caso 2, se analizó un problema un problema de 20 ítems y una mochila con capacidad de 1000 kg. Aquí la interacción entre los parámetros LIT, Pr y PLi muestran influencia en la variable de respuesta y para el caso 3, se tienen 50 ítems y una mochila con capacidad de 230 kg. Se define que los parámetros LNR, PLi, la interacción entre Pr y PLi y la interacción entre LIT y LNR afectan a la variable de respuesta. Además, los resultados obtenidos para la variable de respuesta en los casos 1 y 2 son óptimos, mientras que el para el caso 3 no lo son, resaltando un alto consumo computacional. En el artículo no se muestran cuáles son los niveles escogidos para los parámetros, solamente muestra la influencia de los parámetros en la variable de respuesta.

El trabajo presentado por (Suryadi and Kandi, 2012), aplican el algoritmo Viral System al problema del agente viajero simétrico. Se realiza una comparación contra un algoritmo genético clásico en un problema con 48 ciudades, los resultados computacionales demuestran que para el problema muestra un rendimiento inferior al obtenido por algoritmos genéticos.

De lo anterior, se puede concluir que Viral System es un algoritmo para el cual la calibración de los parámetros es factor fundamental para su adecuado desempeño. Además, para los problemas estudiados, siempre muestra dificultad de encontrar soluciones óptimas al momento de enfrentarse a problemas de tamaños mayor, sumado al tiempo que gasta computacionalmente. Recocido Simulado, ha sido un método ampliamente utilizado en la literatura, que brinda soluciones cercanas a las óptimas y su tiempo computacional es relativamente bajo comparado con otros métodos.

Por su parte, para el algoritmo Wolf Search Algorithm no se encuentran adaptaciones recientes a problemas de Scheduling o similares. Solamente se reportan dos adaptaciones a los problemas del ámbito informáticos en los artículos (Li et al., 2017) y (Agbehadji, Fong and Millham, 2016) mostrando un adecuado desempeño en esos problemas. En estos artículos solamente se presenta a la adaptación de este método al problema, no se muestra información de los niveles de los parámetros utilizados, esto motiva la adaptación e implementación de esta metaheurística para solucionar el problema propuesto.

El resto del artículo se desglosa de la siguiente manera: la sección 2 se explican los materiales y métodos utilizados, en la sección 3 se presentan los resultados de los experimentos computacionales, en la sección 4 se presentan las conclusiones y recomendaciones.

1. MATERIALES Y METODOS

Para este trabajo, se definieron las etapas que se consolidan en la Fig. 1 donde se inicia con la identificación tanto del problema a estudiar como de las metaheurística a utilizar. Posteriormente, se realizó la adaptación de las metaheurísticas al problema. Luego se realizaron los experimentos computacionales y finalmente se analizaron los resultados obtenidos para verificar la pertinencia de adaptar las metaheurísticas al problema.

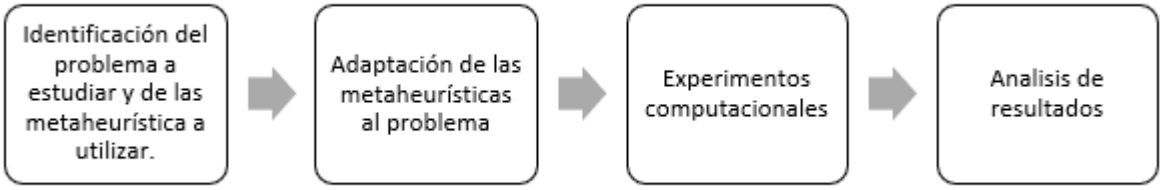


Figura 1 Fases metodológicas. Fuente: Autores.

2.1. problema de programación en una sola maquina con tiempos de alistamiento dependientes de la secuencia.

Considere que se debe programar n trabajos en una sola maquina con tiempos de alistamiento dependientes de la secuencia de tal forma que el objetivo sea minimizar el tiempo de finalización total de todos los trabajos o su equivalente en ingles Makespan. Los tiempos de proceso de cada trabajo son fijos y definidos; además, asociado a cada trabajo, se incurre también en los tiempos de preparación de la máquina que dependen del orden en el que se procesan (secuencia) los trabajos. A partir de lo anterior se consideran los siguientes supuestos.

El tiempo de proceso del trabajo i está dado por P_i.

Los tiempos de preparación (setup) para procesar el trabajo j después de procesar el trabajo i está dado por S_{ij}, donde S_{0j}, representa la preparación inicial cuando el trabajo i es el primer trabajo procesado en la máquina.

Además de ser de estar disponibles en la etapa inicial, los trabajos son independientes entre sí.

La máquina opera sin fallas e interrupciones en el ciclo de programación de programación.

El objetivo es minimizar el tiempo total de finalización de todos los trabajos.

Una variante del problema del agente viajero asimétrico (ATSP) tiene una equivalencia al problema descrito anteriormente, en donde el agento viajero no retorna a la ciudad de origen, los trabajos se asocian a las ciudades y los tiempos de alistamiento se asocian a las distancias; de esta forma el criterio de optimización (Minimizar la distancia total recorrida) es equivalente a minimizar el tiempo de alistamiento (Hornig and Pinto, 2011)

El modelo matemático asociado al problema se muestra a continuación (Ord and Taha, 2006):

Índices:

$$i: Índice\ para\ los\ trabajos$$

$$j: Índice\ para\ los\ trabajos$$

Conjuntos

$$N: Conjuntos\ de\ trabajos.\ \ i, j \in N$$

Parámetros:

$$S_{ij}: \textit{Tiempo de alistamiento de el trabajo i al trabajo j.}$$

$$M: \textit{Es número muy grande.}$$

Variables de Decisión:

$$X_{ij}: \begin{Bmatrix} 1, \textit{Si pasa del trabajo i al trabajo j} \\ 0, \textit{en caso de lo contrario.} \end{Bmatrix}$$

$$U_i: \textit{Variable continua}$$

Función objetivo:

$$Min\ Z = \sum_{i=1}^{N}\sum_{j=1}^{N} S_{ij}X_{ij} \qquad (1)$$

$$S_{ij} = M \ \ \forall\, i, j.$$

Restricciones

$$\sum_{j=1}^{N} X_{ij} = 1;\ \forall\, i\ \in N \qquad (2)$$

$$\sum_{i=1}^{N} X_{ij} = 1;\ \forall\, j\ \in N \qquad (3)$$

$$U_i - U_j + NX_{ij} \leq N - 1;\ \forall\, i, j = 2,3,4 \ldots\ldots N\ \wedge\ i\ \neq j \qquad (4)$$

$$X_{ij} \in \{0,1\} \qquad (5)$$

$$U_i \geq 0 \qquad (6)$$

La ecuación (1) calcula en valor de la función objetivo que es minimizar el tiempo total de finalización de todos los trabajos. La ecuación (2) asegura que desde un trabajo i se pueda ir a un trabajo j. La ecuación (3) asegura que se pueda llegar a un trabajo j desde un trabajo i. La ecuación (4) asegura que no existan sub-tours o ciclos en la solución. Las ecuaciones (5) y (6) definen el dominio de las variables.

2.2 Metaheuristica Wolf Search Algortihm

Wolf Search Algorithm (WSA) es una metaheurística que imita la forma en que los lobos buscan comida y sobreviven evitando a sus enemigos (Tang et al., 2012). Cada lobo tiene un rango visual denominado RV, una velocidad denominada $\propto$ y un tamaño de paso denominado S, los cuales son parámetros que se predeterminan antes de ejecutar el algoritmo. Además, cada lobo está representado por un vector que determina la posición de éste. Cada lobo está representado por una matriz de dos filas por un número de columnas, en este caso, el número de trabajos. La primera fila corresponde a las dimensiones de la posición del lobo. La segunda fila corresponde a la discretización de la posición, esta representará la secuencia de producción a analizar. Esta discretización se realiza utilizando la metodología Short Position Value (SPV) Kumar & Vidyarthi (2016). En la figura 1 muestra cómo se aplica para obtener la discretización de la solución. En primer lugar se generan n números aleatorios en el rango [-5,5] (fila uno de la matriz); se generan los índices respectivamente para cada número generado (fila dos de la matriz). SPV consiste en organizar de menor a mayor los números aleatorios generados, lo cual generará que los índices cambien de posición según sea la ubicación del número aleatorio en la solución. Esta nueva ubicación de los índices describirá

el plan de producción a seguir (secuencia de producción). Esta metodología se ha utilizado para discretizar soluciones como se evidencia en el trabajo propuesto por (López Martínez, Hernández Riaño and Soto de la Vega, 2019), en el cual, se discretiza las soluciones para un algoritmo basado en la búsqueda de alimentos por parte de ardillas, logrando que el algoritmo alcance excelentes resultados.

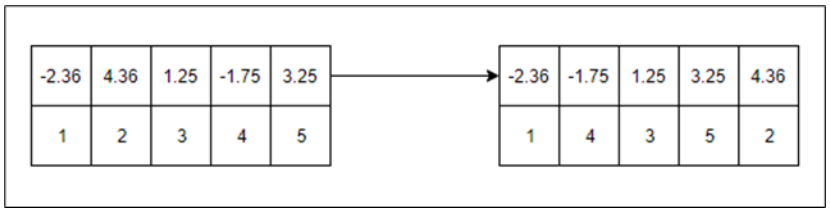


Figura 2 Representación de un lobo. Fuente: Autores.

El algoritmo inicia con una población inicial de lobos dispersos en un bosque abierto. Luego para cada lobo se genera una nueva posición conforme a la ecuación 7:

$$X_i^{t+1} = X_i^t + (\alpha * RV * U[-5,5]) \quad (7)$$

Donde X_i^{t+1} y X_i^t representan la coordenada de la posición en el momento t+1 y la coordenada actual del lobo respectivamente. En este punto, se verifica que lobo de la manada está dentro del rango visual de la ubicación generada, para así, moverse hacia esa posición. Si la posición hacia la que se movió el lobo es mejor que la que tenía anteriormente, el lobo escoge seguir ese camino. En caso contrario, genera una nueva posición conforme a la ecuación número 7. Ahora bien, en los dos casos anteriores; puede existir la presencia de un predador. Cuando esto sucede, el lobo debe escapar rápidamente del rango visual del predador para no ser cazado. Esto movimiento de escape está definido por la ecuación 8.

$$X_i^{t+1} = X_i^t + (\alpha * S * U[-5,5]) \quad (8)$$

Donde X_i^{t+1} y X_i^t representan la coordenada de la posición en el momento t+1 y la coordenada actual del lobo respectivamente y S es el tamaño de paso del lobo. Este comportamiento, le da al algoritmo la característica de explorar el espacio de soluciones.

2.3 Descripcion De Instancias Y Solución

En este apartado, se generaron las instancias (problemas) con las cuales se va a colocar a prueba el método propuesto en esta investigación. Las instancias se generaron aleatoriamente utilizando el software R versión 3.5.3 "Great (R Core Team, 2019) mediante una distribución uniforme con rango [1, 200] para los tiempos de alistamiento, para los tiempos de proceso, se generaron números bajo una distribución uniforme con rango [1,35]. Se generaron 35 instancias distribuidas en 7 grupos de 5 instancias cada uno, con tamaños entre 15, 20, 35, 50, 75, 100, 150 trabajos respectivamente.

En este punto, se encontraron las mejores soluciones para las instancias generadas. Esto con el fin de tener un patrón de comparación para medir el desempeño del algoritmo adaptado al problema. Se implementó en el lenguaje AMPL, el modelo matemático y se utilizó la plataforma Neos Server para encontrar las soluciones correspondientes. El solver escogido para esta etapa fue Gurobi. Para las instancias de tamaño CGC150B, CGC150C y CGC150E para el tiempo de corrida fijado, no se obtuvieron soluciones optimas; el gap obtenido fue de 3.55%, 2.79% y 0.62% respectivamente. Estos resultados se tomaron como mejor solución para hacer las respectivas comparaciones.

2.4. Metaheuristica Viral System Y Recocido Simulado

Sistema Viral o Viral System (VS) es una metaheurística inspirada en el comportamiento de los virus al infectar células

del cuerpo humano, esta fue por propuesta por (Cortés et al., 2008). Es una metaheurística Bio-Inspirada. El algoritmo inicia con la generación de la población inicial aleatoria. Una célula está representada por una codificación permutada, formada por un vector de tamaño igual al número de trabajos. La población inicial se conforma de un número finito de células. Se establece un número de iteraciones fijo como criterio de parada, es decir, el algoritmo terminará cuando se alcance este número de iteraciones. Cada célula podrá infectarse por medio de un tipo de infección, esta puede ser Lítica o Lisogénica. Para esto, se debe definir que células se van a infectar por medio del proceso lítico y cuales por el proceso lisogénico. Se define una probabilidad en que la célula sea infectada por el proceso lítico Plit, esta probabilidad es comparada con la generación de un numero aleatorio distribuido uniformemente entre 0 y 1, si el número aleatorio generado es menor que esta probabilidad, la célula seguirá un proceso lítico, sino seguirá un proceso lisogénico. Esto se debe realizar para cada célula de la población inicial. En el proceso lítico, la célula es infectada por el virus y este comienza a multiplicarse y copar por completo la célula hasta su ruptura (Muerte de la célula). En este punto, el virus sale e infecta a las células vecinas. Un vecino se puede obtener intercambio una posición del vector célula. En total, si se considera un intercambio, el número de vecinos serán calculados por la ecuación 9:

$$N°Vecinos = \frac{n(n-1)}{2} \qquad (9)$$

Después de evaluar todos los vecinos de la célula muerta; el vecino con mejor valor de Makespan la reemplazará en el resto del desarrollo del algoritmo.

Por otra parte, en el proceso lisogénico, la célula sufre una mutación. Esta mutación ocurre de la siguiente manera; se generan dos números aleatorios enteros a y b entre 1 y n, posteriormente se hace el intercambio en la célula de estas posiciones. Después de repetir este proceso para cada célula y por cada iteración, el algoritmo se detiene y localiza la célula con mejor valor de Makespan, la cual será la solución al problema.

Por su parte, el algoritmo de Recocido Simulado (RS) es una metaheurística basada en el proceso de recocido de metales, el cual consiste en someter a altas temperaturas el material y luego hacer un enfriamiento controlado hasta llegar a una temperatura de equilibrio. Este enfriamiento controlado garantiza que el material tenga unas propiedades específicas tales como dureza, ductilidad etc. El algoritmo fue propuesto por (Kirkpatrick, 2014) y ha sido adaptado a una diversidad de problemas de optimización.

El algoritmo inicia con la generación de la solución inicial. Se fija una temperatura de inicio muy alta, la cual va a ir disminuyendo conforme a la ecuación 9 hasta llegar a una temperatura de control mínima T_{min} preestablecida al inicio de la ejecución del algoritmo.

$$T_{actual} = \alpha T \qquad (9)$$

Donde α es el parámetro de enfriamiento y se fija como un parámetro al inicio de la ejecución. Luego, se genera una solución vecina. Se calculan los valores del Makespan para cada una de las soluciones, las cuales van a representar la energía de cada solución. Posteriormente se calcula ΔE y se evalúa la condición si es mayor a cero o no. Si ΔE es negativo, quiere decir que la solución generada como vecina es mejor que la solución actual, por lo que será escogida para seguir el proceso de optimización. Si ΔE es positivo, quiere decir que la solución vecina generada no es mejor que la solución actual, por lo que se debe definir si se acepta o no. Para esto, el algoritmo evalúa una probabilidad de aceptación. La probabilidad de aceptación es calculada conforme a la ecuación 10.

$$Probabilidad\ de\ aceptación = e^{-\frac{\Delta E}{T}} \qquad (10)$$

Se genera un número aleatorio entre 0 y 1 distribuido uniformemente y se compara con el valor de la probabilidad de aceptación; si este número aleatorio es menor, se acepta la solución vecina para seguir el proceso de optimización. Por su parte, si el número aleatorio es mayor, la solución vecina no es tenida en cuenta y la solución actual es

escogida para seguir en el proceso de optimización. A este proceso se le conoce como algoritmo Metrópolis (Kirkpatrick, 2014). Después de cumplir el criterio de parada, el cual sucede cuando la temperatura actual es igual o menor a la temperatura mínima preestablecida; el algoritmo para y devuelve la mejor solución obtenida hasta el momento.

2.5 Obtencion De Los Niveles Para Los Parámetros De Las Metaheuristicas

A continuación, se detalla la calibración de los algoritmos utilizados. En primer lugar, se fijaron los parámetros para el algoritmo Viral System (VS). Esto, se basó en el análisis de un algoritmo VS aplicado a un problema de una sola máquina para optimizar la tardanza ponderada total desarrollado en (Santosa and Affandi, 2013), en este trabajo se determinaron 6 parámetros con los siguientes niveles respectivos: Tamaño clínico: 20, Probabilidad Proceso Lítico: 0.6, Probabilidad Generación de antígenos: 0.3, Número máximo de núcleos replicados: 5, Número máximo de núcleos mutados: 5 y Probabilidad de réplica: 0.7. El número de iteraciones se fijó en 1000.

Para el algoritmo Recocido Simulado (RS), se tomó como punto de partida los niveles de los parámetros utilizados en el trabajo propuesto por (Jin, Song and Wu, 2009). Desde aquí se configuró un extenso análisis computacional para determinar cuál combinación de los niveles brindaba mejor solución. Se trabajó para cada parámetro tres niveles, teniendo como nivel central, el propuesto en el trabajo mencionado. Se fijó un número de iteraciones de 500, se replicó cinco veces el experimento durante un tiempo de 24 horas en una instancia escogida al azar. Los niveles escogidos fueron: Temperatura Inicial: 250, Temperatura Final: 2 y Alfa: 0.95.

Por último, para el algoritmo Wolf Search Algorithm (WSA) se realizaron experimentaciones computacionales previas con diferentes combinaciones de parámetros, las cuales ayudaron a detectar que nivel de los parámetros podrían tenerse en cuenta para obtener un adecuado desempeño del algoritmo. Se fijó un tamaño de 150 lobos, con un número de iteraciones de 500, con cinco replicas con un tiempo de duración de 24 horas en una instancia escogida al azar. Los niveles escogidos fueron: Rango Visual: 35, Tamaño de paso: 5, Velocidad Lobo: 0.5 y Probabilidad Depredador: 0.45.

Todos los experimentos se ejecutaron en un computador Dell Intel® Core ™ i5-6500 CPU @ 3.20 GHz 3.19 GHz con memoria RAM de 8 GB.

2. ANALISIS DE RESULTADOS

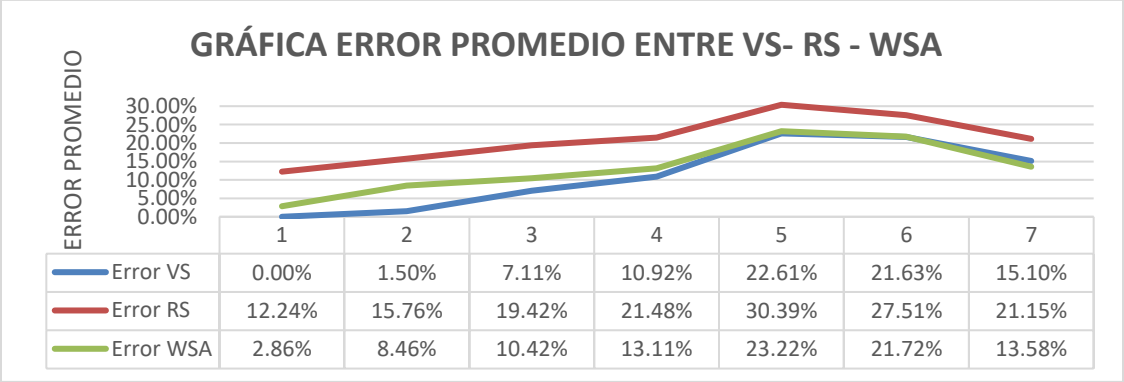
Los resultados obtenidos se resumen en la Tabla 1, donde son discriminados por grupos de instancias (trabajos), el cual serán analizados los respectivos resultados obtenidos por los tres métodos propuestos WSA, VS y RS, con respecto a soluciones obtenidas por Gurobi en cuanto al tiempo de terminación total de los trabajos, la cual fue la variable dependiente del estudio.

Tabla 1 Resultados obtenidos. Fuente: Autores

ID	INSTANCIAS	VS	RS	WSA	GUROBI	DESVIACION VS	DESVIACION RS	DESVIACION WSA
1	CGC15A	672	758	734	672	0,000	0,113	0,084
	CGC15B	693	749	722	693	0,000	0,103	0,040
	CGC15C	712	795	712	712	0,000	0,155	0,000
	CGC15D	668	828	668	668	0,000	0,188	0,000
	CGC15E	797	912	812	797	0,000	0,263	0,018
2	CGC25A	1029	1242	1079	986	0,042	0,459	0,086
	CGC25B	889	991	931	873	0,018	0,322	0,062
	CGC25C	949	1102	1035	944	0,005	0,390	0,088
	CGC25D	989	1125	1074	981	0,008	0,403	0,087
	CGC25E	1057	1305	1172	1055	0,002	0,485	0,100
3	CGC35A	1174	1356	1204	1143	0,026	0,504	0,051
	CGC35B	1108	1219	1133	1035	0,066	0,449	0,086
	CGC35C	1375	1583	1457	1286	0,065	0,575	0,117
	CGC35D	1083	1314	1105	975	0,100	0,489	0,118
	CGC35E	1116	1285	1182	1006	0,099	0,477	0,149
4	CGC50A	1443	1705	1507	1287	0,108	0,606	0,146
	CGC50B	1334	1454	1369	1193	0,106	0,538	0,129
	CGC50C	1731	1960	1772	1547	0,106	0,657	0,127
	CGC50D	1517	1683	1497	1369	0,098	0,601	0,086
	CGC50E	1527	1780	1601	1331	0,128	0,622	0,169
5	CGC75A	1844	2067	1856	1628	0,117	0,675	0,123
	CGC75B	1687	1916	1737	1258	0,254	0,649	0,276
	CGC75C	1735	1886	1751	1270	0,268	0,644	0,275
	CGC75D	1763	1898	1702	1303	0,261	0,646	0,234
	CGC75E	1672	1902	1723	1287	0,230	0,647	0,253
6	CGC100A	2226	2442	2276	1911	0,142	0,725	0,160
	CGC100B	2161	2296	2129	1750	0,190	0,707	0,178
	CGC100C	2426	2664	2411	2085	0,141	0,748	0,135
	CGC100D	2351	2506	2351	1198	0,490	0,732	0,490
	CGC100E	2244	2414	2251	1977	0,119	0,722	0,122
7	CGC150A	2889	3169	2819	2457	0,150	0,788	0,128
	CGC150B	2998	3284	2855	2505	0,164	0,795	0,123
	CGC150C	2805	2914	2764	2471	0,119	0,769	0,106
	CGC150D	2820	3102	2820	2387	0,154	0,783	0,154
	CGC150E	2900	3064	2900	2411	0,169	0,781	0,169

Como primer hallazgo, se puede observar que VS y WSA encontraron soluciones óptimas en instancias de tamaño 15. VS alcanzó soluciones óptimas en dichas instancias. Para el método WSA se encontraron en el mismo tamaño de instancias dos soluciones óptimas, y por último el RS no encontró ningún resultado óptimo en comparación con los dos anteriores algoritmos. Para instancias mayores ningún método encontró resultados óptimos, sin embargo, cabe resaltar que el error promedio de los algoritmos arroja buenos resultados, destacando que el VS tiene menor error promedio para instancia de tamaño 20. A medida que incrementa el tamaño de instancias el error promedio entre WSA y VS se reduce. En instancias de tamaño 100 se puede apreciar que el error promedio de VS y WSA arroja el mismo valor lo que muestra un comportamiento similar y se empieza a observar que en instancias de tamaño 150 el error promedio de WSA es mejor que el VS.

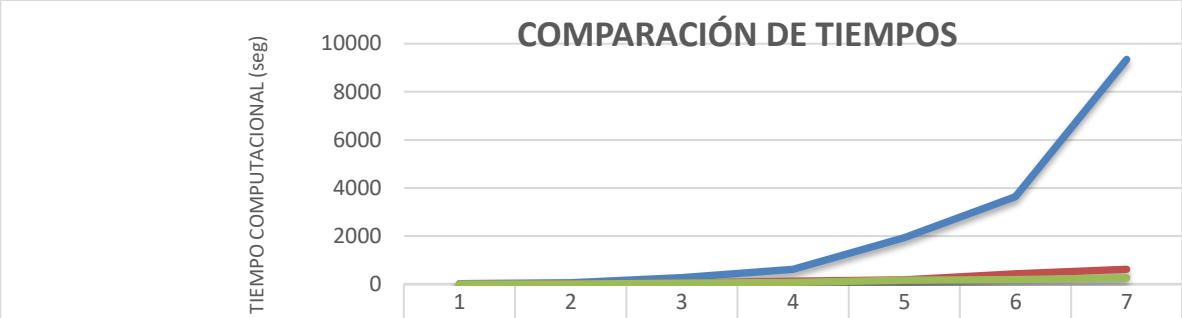
Ahora bien, la Figura 3 muestra el error promedio entre los tres algoritmos en mención, al principio se puede apreciar la diferencia entre los tres algoritmos pero a medida que aumenta el tamaño de la instancia se puede apreciar que VS y el WSA tiene un margen de error muy similar, además cabe resaltar que el error promedio de los tres métodos de solución tiene un comportamiento ascendente hasta el bloque 5, y a partir de ese punto el error promedio tiene una pendiente descendente, teniendo en cuenta que la brecha entre los dos algoritmos VS y WSA es prácticamente nula, por último se puede decir que hasta este momento el algoritmo WSA y el VS no tiene una gran diferencia a la hora de encontrar los resultados para cada problema analizado.



	1	2	3	4	5	6	7
Error VS	0.00%	1.50%	7.11%	10.92%	22.61%	21.63%	15.10%
Error RS	12.24%	15.76%	19.42%	21.48%	30.39%	27.51%	21.15%
Error WSA	2.86%	8.46%	10.42%	13.11%	23.22%	21.72%	13.58%

Figura 3 Error promedio entre los métodos utilizados. Fuente: Autores

La Figura 4 ilustra el tiempo promedio de respuesta entre las tres herramientas utilizadas, de primera mano se puede observar tanto que RS y WSA tienen un comportamiento similar a medida que incrementa el tamaño entre instancias. También se destaca que a partir del bloque 4, se evidencia un incremento en el tiempo computacional promedio de VS a diferencia de los otros métodos. Este comportamiento se puede explicar debido a la naturaleza de búsqueda del algoritmo al momento de evaluar las células que se infectan por el proceso lítico. En este proceso, se deben evaluar uno y cada uno de los vecinos de la célula infectada, esto, para problemas de gran tamaño, obliga a un mayor consumo computacional con respecto a los otros métodos utilizados.



	1	2	3	4	5	6	7
TIEMPO PROMEDIO VS (seg)	20.2	57.8	270	622.6	1933	3642.6	9348.6
TIEMPO PROMEDIO RS (seg)	3.4	12.8	51	123.6	174	435.6	620.4
TIEMPO PROMEDIO WSA (seg)	6.8	11.4	45.6	69.8	163.2	190.4	264.4

Figura 4 Tiempo computacional de respuestas. Fuente: Autores.

Para determinar cuál de los métodos obtuvo un mejor resultado frente a los problemas estudiados, se procedió a realizar un análisis de varianza (ANOVA) de un solo factor, donde los niveles serán los algoritmos VS, RS, WSA. Este ANOVA se realizó en bloques, debido a que se tienen grupos de instancias diferentes. Para iniciar, se deben comprobar los supuestos para que el estudio sea considerado válido. Se realizaron las pruebas de normalidad, homocedasticidad e independencia.

Normalidad:

El primer supuesto que se comprobó fue el de la normalidad. Aquí se comprueba que los residuos estándares para el tiempo de terminación total de los trabajos se ajusten a una distribución normal. Para este fin, se realizó la prueba Kolmogorov Smirnov. Con la ayuda del software R se logró obtener un P-valor de 0.2883, por lo cual, con un nivel de confianza del 95% es posible afirmar que los residuos estándares provienen de una distribución normal.

Igualdad de Varianzas:

Otro supuesto importante al momento de comprobar una ANOVA es considerar que los grupos de datos tienen varianzas iguales, para este fin, se realizó la prueba de Levene para medir la igualdad de varianzas entre grupos. Con la ayuda del software R se logró obtener un P-valor de 0.9103, por lo cual, se puede afirmar que con un nivel de confianza del 95% los grupos de datos tienen varianzas iguales.

Independencia

Al momento de realizar los experimentos, se garantizó que los mismos fueran independientes, de esta forma se puede asegurar que las mediciones obtenidas no interfieren entre si garantizando independencia.

Con la evidencia sólida reportada hasta el momento, es posible realizar el análisis de varianza para encontrar cuál de los métodos tienen mejor desempeño. En la tabla 2, se puede apreciar el resultado de este análisis:

Tabla 2 Análisis de varianza. Fuente: Software R.

	Grados De Libertad	Suma Cuadrados	Cuadrados Medios	Valor F	P- Valor
Método	2	675665	337832	30.35	6.1 x 10-11
Instancia (Bloque)	6	522997894	8715831	783.05	2 x 10-16
Residuales	96	1068536	11131		

Se puede argumentar con un nivel de confianza del 95%, que existe diferencias significativas entre los métodos, dado que el valor P obtenido es menor a 5%.

Ahora bien, este análisis nos indica solamente si hay o no diferencia estadísticamente significativa entre los niveles del factor, para encontrar que nivel permite optimizar la variable de respuesta, se debe acudir a la prueba Tukey. Esta prueba clasifica en pares los niveles y muestra si existe o no diferencia entre estos. Al aplicar la prueba Tukey, se obtiene el siguiente resultado resumido en la tabla 3:

Tabla 3 Prueba Tukey. Fuente: Software R.

Prueba Tukey (Comparación De Medias – Método)				
	Diferencia	Bajo	Alto	P- Valor Ajustado
VS-RS	-179.60	-239.63	-119.56	0.000000
WSA-RS	-158.82	-218.86	-98.79	0.000000
WSA-VS	20.77	-39.26	80.80	0.6893

Al analizar el valor P ajustado obtenido por la prueba, con un nivel de confianza del 95%, se puede inferir que no existe una diferencia estadísticamente significativa entre el método WSA y el método VS, puesto que su p ajustado es mayor al 5%. Mientras que, en los otros dos grupos, si existe evidencia que hay diferencia estadísticamente significativa entre ellos, puesto que sus valores P están por debajo del nivel de significancia de 5%.

Bajo estos estudios analíticos se pueden afirmar, en efecto como se demostró en el análisis descriptivo, los algoritmos VS y WSA presentan comportamientos similares y a la vez son los que mejor respuesta generan cuando se implementan al problema estudiado.

3. CONLUSIONES Y RECOMENDACIONES

Para determinar el desempeño del algoritmo Wolf Search Algorithm en el problema estudiado, se propuso un esquema de codificación discreto permutado que representara las características naturales de un programa de producción en estos tipos de configuraciones. Sumado a esto, las soluciones encontradas a las instancias generadas, además de la adaptación de dos algoritmos más al problema, permitieron crear un contexto completo para lograr responder la pregunta problema planteada en este proyecto.

Al analizar los datos suministrados por los estudios computacionales, se puede definir qué; en efecto, se logró encontrar soluciones al problema por medio de este algoritmo. Aunque las soluciones encontradas en algunos grupos de instancias no fueron óptimas, no es distante en relación con el comportamiento frente a los algoritmos enfrentados, ubicándose junto a Viral System dentro del grupo que presentó mejor desempeño en el proyecto. Dentro de este grupo, se evidenció que para instancias de menor tamaño, Viral System obtiene soluciones óptimas o en su defecto, soluciones muy cercanas al óptimo.

A medida que los grupos de instancias cambian a problemas de mayor tamaño, este algoritmo presenta un gran inconveniente; el tiempo computacional gastado. Viral System mostró un elevado tiempo computacional (aproximadamente 2 horas) para encontrar la mejor solución a partir de los grupos de instancias de tamaño 50. En contra parte WSA, se mostró rápido en estos grupos de instancias (aproximadamente 70 segundos) y como se evidenció en los resultados computacionales, mostró un mejor comportamiento en los últimos grupos de instancias analizados.

La comparación estadística realizada por medio de un análisis de varianza demostró que existe una diferencia estadística significativa entre los métodos utilizados y por medio de una prueba Tukey se confirmó que los métodos VS y WSA presentan desempeños similares diferenciándose de RS en calidad de respuesta (minimizar Makespan).

En conclusión, WSA es un algoritmo alternativo para abordar el problema de programación de producción en una sola maquina con tiempos de alistamientos dependientes de la secuencia. Su fácil adaptación e implementación al problema, ligado a los resultados obtenidos permiten promover el uso en este o entro problema de programación de producción o de optimización combinatoria.

Para lograr que el algoritmo WSA pueda ubicarse en soluciones óptimas o encontrar soluciones muy cercanas a este; se propone comprobar el desempeño del algoritmo, utilizando otros métodos de discretización, tales como, Sigmoid Function (SF) o el método Random Key (RK).Sumado a esto, se recomienda resolver instancias de mayor tamaño, puesto que, se logró detectar que para este grupo el desempeño del método WSA podría ser más efectivo. Además, se propone realizar una experimentación más robusta para la calibración de los parámetros del algoritmo. Esto, podría afectar la convergencia hacia una solución óptima. Además, se recomienda realizar una validación de funcionamiento del algoritmo, abordando una instancia con datos reales.

4. AGRADECIMIENTOS

Este proyecto se realizó gracias al apoyo de la Facultad De Ingeniería Industrial perteneciente a la escuela de Arquitectura e Ingeniería de la Universidad Pontifica Bolivariana Seccional Montería, además al Centro De investigación, desarrollo e innovación – CIDI de la Universidad Pontifica Bolivariana Seccional Montería por el apoyo brindado a lo largo de este proyecto.

1. REFERENCIAS BIBLOGRAFICAS

[1]. Abdel-Basset, M., Abdel-Fatah, L. and Sangaiah, A. K. (2018) Metaheuristic Algorithms: A Comprehensive Review, Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications. Elsevier Inc. doi: 10.1016/b978-0-12-813314-9.00010-4.

[2]. Agbehadji, I. E., Fong, S. and Millham, R. (2016) 'Wolf search algorithm for numeric association rule mining', Proceedings of 2016 IEEE International Conference on Cloud Computing and Big Data Analysis, ICCCBDA 2016, pp. 146–151. doi: 10.1109/ICCCBDA.2016.7529549.

[3]. Cortés, P. et al. (2008) 'Viral systems: A new bio-inspired optimisation approach', Computers and Operations Research, 35(9), pp. 2840–2860. doi: 10.1016/j.cor.2006.12.018.

[4]. Hornig, E. S. and Pinto, O. S. (2011) 'Estrategias mmas para minimización del makespan en la programación de una máquina con setup', Revista Ingeniería Industrial, 10(2), pp. 17–29.

[5]. Jin, F., Song, S. and Wu, C. (2009) 'A simulated annealing algorithm for single machine scheduling problems with family setups', Computers and Operations Research, 36(7), pp. 2133–2138. doi: 10.1016/j.cor.2008.08.001.

[6]. Kirkpatrick, S. (2014) 'Optimization by Simulated Annealing Optimization by Simulated Annealing', 220(January 1983). doi: 10.1142/9789812799371.

[7]. Li, J. et al. (2017) 'Elitist Binary Wolf Search Algorithm for Heuristic Feature Selection in High-Dimensional Bioinformatics Datasets', Scientific Reports. Springer US, 7(1), pp. 1–14. doi: 10.1038/s41598-017-04037-5.

[8]. López Martínez, C. A., Hernández Riaño, H. E. and Soto de la Vega, M. J. (2019) 'Un Squirrel Search Algorithm discreto aplicado al problema Job Shop con operadores calificados', Inge Cuc, 15(2), pp. 1–13. doi: 10.17981/ingecuc.15.2.2019.14.

[9]. Ord, K. and Taha, H. A. (2006) Operations Research: An Introduction., Operational Research Quarterly (1970-1977). doi: 10.2307/3008276.

[10]. Osaba, E. et al. (2018) 'A discrete water cycle algorithm for

solving the symmetric and asymmetric traveling salesman problem', Applied Soft Computing Journal, 71, pp. 277–290. doi: 10.1016/j.asoc.2018.06.047.

[11]. Pinedo, M. L. (2012) Scheduling. Fourth Edi. Edited by Springer. Boston, MA: Springer US. doi: 10.1007/978-1-4614-2361-4.

[12]. R Core Team (2019) 'R: A language and environment for statistical computing'. R Foundation for Statistical Computing. Available at: https://www.r-project.org/.

[13]. Santosa, B. and Affandi, U. (2013) 'Application of viral systems for single-machine total weighted tardiness problem', IOP Conference Series: Materials Science and Engineering, 46(1). doi: 10.1088/1757-899X/46/1/012010.

[14]. Suryadi, D. and Kandi, Y. (2012) 'A Viral Systems Algorithm for the Traveling Salesman Problem', Proceedings of the 2012 International Conference on Industrial Engineering and Operations Management, pp. 1989–1994.

[15]. Suryadi, D. and Kartika, E. K. (2011) 'Viral systems application for Knapsack problem', Proceedings - 3rd International Conference on Computational Intelligence, Communication Systems and Networks, CICSyN 2011, pp. 11–16. doi: 10.1109/CICSyN.2011.16.

[16]. Tang, R. et al. (2012) 'Wolf search algorithm with ephemeral memory', 7th International Conference on Digital Information Management, ICDIM 2012, (August), pp. 165–172. doi: 10.1109/ICDIM.2012.6360147.