Job Shop Scheduling Algoritmos Evolutivos Grupo 12

Sergio Bonilla, 4.430.955-3
Leonardo Clavijo, 5.054.830-5
{serginhobonilla, joleocl}@gmail.com

***Abstract*—Este artículo presenta un algoritmo evolutivo eficiente aplicado al problema Hard-NP conocido como Job Shop Scheduling. El algoritmo planteado está especialmente diseñado para lograr soluciones óptimas al problema, utilizando operadores evolutivos específicos, y una codificación que impacta de forma positiva generando soluciones factibles al problema. En la sección evaluación experimental se plantea las acciones a tomar para demostrar la eficiencia del algoritmo ante versiones deterministas que resuelven el problema utilizando heurísticas simples. Se plantea la comparación entre operadores de cruzamiento y el impacto que puede tener éste sobre la calidad de la solución al problema. El objetivo es brindar una solución eficiente a JSSP utilizando algoritmos evolutivos, mejorando resultados obtenidos a partir de heurísticas.**

I. Introducción

En el siguiente trabajo se presentará un problema de optimización combinatoria como es la planificación de procesos en un ambiente multiprocesador. Se explicará acerca de las técnicas utilizadas resolver el problema. Se hará una presentación formal acerca de éste y se compararán resultados con respecto a otras modalidades de resolución.

II. Algoritmos Evolutivos

La computación evolutiva es una rama de la computación; y en particular de la inteligencia artificial. Se la puede definir como un conjunto de metaheurísticas que emulan la evolución biológica. Interpretando a la naturaleza e inspirados en la teoría de la evolución de Charles Darwin, se crea un mecanismo análogo a los procesos evolutivos de ésta, con el objetivo primordial de resolver problemas de optimización combinatoria. Este mecanismo trabaja con una población de individuos, que sigue la idea de supervivencia de los individuos más aptos (mediante una función de fitness), la idea de reproducción (mutación y recombinación) y la idea de diversidad genética.

III. Job Shop Scheduling

Uno de los problemas clásicos dentro del área de ciencias en computación y la optimización combinatoria, es el problema conocido como JSP (Job Shop Scheduling). Los problemas de optimización combinatoria, básicamente poseen un espacio de soluciones discretas, pero que en general son muy grandes debido a que se representan como permutaciones o combinaciones. Por lo que encontrar una solución óptima, o cercana a él, es bastante complejo con técnicas comunes. Este tipo de problemas se los caracteriza como difíciles o NP-difíciles. Es por eso que se decide utilizar técnicas de algoritmos evolutivos para plantear una posible solución al JSP.

El problema de Job Shop Scheduling es un tipo de problema de planificación de tareas, y en particular de optimización; donde se busca optimizar el makespan o tiempo de ejecución de todas las tareas. Básicamente el problema consiste en planificar un conjunto de tareas (Jobs) N sobre un conjunto de recursos o máquinas heterogéneas M. Por otra parte se tiene que cada Job está compuesto por un conjunto de operaciones, las cuales son dependientes, es decir, se deben ejecutar en forma secuencial. Además cada tarea tiene un tiempo de término más tardío, no hay restricciones de precedencia entre operaciones de distintas tareas y el planificador es no expropiativo.

El problema queda definido de la siguiente forma:

- Se considera un conjunto de N tareas, independientes entre ellas.
- Sea M la cantidad de máquinas involucradas en el problema, presentan la característica de ser heterogéneas, capaces de ejecutar únicamente una tarea por vez.
- Un trabajo $j, 1 \leq j \leq J$ está compuesto por K_j operaciones.
- Las operaciones mantienen un orden de precedencia preestablecido, esto quiere decir $o_{j,1} \leq o_{j,2} \leq ... \leq o_{j,K_j}$.
- No existe dependencias entre operaciones de distintas tareas.
- Existen tiempos de ejecución predefinidos dado un trabajo j, una operación k y una máquina m, se define en una matriz costos $C_{j,k,m}$, con $1 \leq j \leq J, 1 \leq k \leq K_j, 1 \leq m \leq M$.
- Planificador no expropiativo, esto se debe a que las tareas poseen tiempo de término máximo, por lo cual no puede expropiarse el recurso a una operación una vez le es asignado el recurso.
- Se desea obtener el tiempo de comienzo de ejecución para cada operación, y en qué máquina son asignadas las mismas. Para ello se define $t_{j,k,m}$ como el tiempo de inicio de ejecución de la operación $o_{j,k}$ en la máquina m. Por lo cual para cada operación se obtendrá el par (t_{inicio}, m).

Una vez definido el problema en cuestión, se procede a la formulación matemática del mismo, esto implica la función objetivo a optimizar, en conjunto con las restricciones del problema planteadas en el punto anterior.

El objetivo es minimizar el makespan. Estas son las variables utilizadas para la formulación:

- $t_{j,k,m}$: tiempo de inicio de ejecución de la operación k correspondiente al trabajo j en la máquina m.
- $C_{j,k,m}$: tiempo de proceso de la operación k del trabajo j en la máquina m.
- $p_{j',k',j,k,m}$: variable binaria que define si una operación precede a otra

$$p_{j',k',j,k,m} \begin{cases} 1, & (j,k) \text{ anterior a } (j',k') \text{ en } m \\ 0, & \text{otro caso} \end{cases}$$

- M: Número positivo.

La formulación matemática queda definida:

Minimizar makespan

$$f : min\{max\{t_{j,K_j,m} + C_{j,K_j,m}\}\} \tag{1}$$

s.a

$$t_{j,k,m} + C_{j,k,m} \leq t_{j,k+1,m} \forall j,k,m \tag{2}$$

$$t_{j,k,m} + C_{j,k,m} \leq t_{j',k',m} + M \cdot (1 - p_{j',k',j,k,m}) \forall j,j',k,m \tag{3}$$

$$t_{j',k',m} + C_{j',k',m} \leq t_{j,k,m} + M \cdot p_{j',k',j,k,m} \forall j,j',k,m \tag{4}$$

$$t_{j,k,m} \geq 0 \forall j,k,m \tag{5}$$

$$C_{j,k,m} \geq 0 \forall j,k,m \tag{6}$$

Donde $1 \leq j, j' \leq J$, J cantidad de trabajos del problema; $1 \leq m \leq M$, siendo M la cantidad de máquinas del problema; K_j la cantidad de operaciones de la tarea j; $1 \leq k, k' \leq K_j$. La ecuación (1) es la función objetivo del problema. En la ecuación (2) se impone restricción sobre la precedencia de las operaciones.

En las ecuaciones (3) y (4) se impone restricción sobre la utilización de la máquina, una máquina procesa una operación a la vez. En las ecuaciones (5) y (6) define que las variables en cuestión deben ser positivas.

IV. Estrategia de Resolución

En esta subsección se describirán los operadores evolutivos a utilizar, la codificación del problema y la función fitness.

A. Codificación

Existen dos categorías de codificar un problema, la representación directa [8][2] refiere a codificar el cromosoma como el problema en sí mismo, mientras que la representación indirecta recurre a una estructura distinta, sea por motivos de eficiencia o complejidad de la solución. La codificación seleccionada es indirecta, consiste en una estructura M con un arreglo de tamaño $\sum_{i=1}^{J} cantidad\_operaciones(J)$ representando la cantidad total de operaciones involucradas en el problema, para cada celda de la estructura M corresponde a un valor $j, 0 \leq j \leq J$, donde J es la cantidad del trabajos. De esta forma se obtiene el orden de las operaciones a ser ejecutadas. Observar que se cumple las restricciones del problema, ya que las operaciones dentro de una tarea quedan ordenadas según las ocurrencias en el vector. Seleccionar correctamente la codificación del problema repercute de forma directa la eficiencia de la solución al problema. La selección de esta estructura parece agregar simplicidad en el momento de resolver el problema, ya que el mismo presenta características deseables en cuanto a las restricciones del problema. Para lograr este objetivo es necesario una estructura que almacene la cantidad de operaciones para cada trabajo, se utiliza un arreglo de tamaño J con la cantidad máxima de operaciones para cada trabajo. Observar que la codificación seleccionada simplifica el proceso de aplicar operadores evolutivos, tanto primario como ser el cruzamiento como la mutación. Para el caso del cruzamiento basta seleccionar un operador de recombinación de dos puntos, se ha seleccionado éste operador con la idea de mantener el orden preestablecido dada una solución al problema. Si bien este operador genera una solución que no satisface las restricciones del problema, se crea una operación extra [IV-G] para lograr que la solución generada sea satisfactible. De esta forma se fuerza que todas las soluciones al problema sean satisfactibles; este aspecto de forzar a soluciones satisfactibles presenta una característica deseable; la solución al problema converge de forma más rápida.[1] También dada la codificación de la solución, se simplifica el operador evolutivo de mutación, seleccionando una celda de forma aleatoria del arreglo y cambiando su valor por una tarea del problema. Posteriormente se requiere un proceso de transformación a una solución satisfactible.

B. Función Fitness

El objetivo de la función fitness es minimizar el tiempo de finalización de todos los trabajos, se define el tiempo de finalización de cada trabajo como $t_{final}(j, K_j) = t_{inicial}(j, K_j) + C_{j,K_j,m}$, donde m es el identificador de la máquina que ejecuta la operación K_j del trabajo j; $C_{j,K_j,m}$ es el costo asociado dado un trabajo, operación y máquina, de aquí en adelante se representa a una operación de un trabajo j como $o_{j,k}, 1 \leq k \leq K_j$ donde K_j es la cantidad de operaciones del trabajo. La función fitness será el mayor valor del tiempo de finalización del conjunto de trabajos.

Sea S una solución al problema, la función fitness de S queda expresada (7).

$$f(S) = max\{t_{inicial}(j, K_j) + C(j, K_j, m))\} \quad (7)$$

JSSP se trata de un problema combinatorio, dada la codificación planteada se debe implementar operadores evolutivos específicos para generar soluciones factibles a las restricciones del problema.

C. Inicialización

En este caso se debe tener en cuenta que inicializar poblaciones de forma aleatoria puede generar soluciones no factibles al problema, para ello se debe implementar un operador de construcción aleatorio, teniendo en cuenta varias heurísticas para obtener una población inicial satisfactible y de ésta forma lograr la convergencia en una solución óptima en menor cantidad de generaciones. Para generar una población inicial se puede utilizar alguno de los métodos descriptos a continuación:

1) Generación aleatoria de individuos, para ello se asigna un valor inicial y luego se le asigna a cada operación una máquina de forma aleatoria, en el transcurso de la operación se debe tener en cuenta las restricciones de dominio sobre los individuos, descriptos en la descripción del problema. Se brindan mecanismos de reparación y checkeo de las soluciones generadas.
2) Utilizar heurísticas conocidas para la generar una solución al problema, y posteriormente aplicar operadores evolutivos como ser cruzamiento y mutación para generar nuevos individuos.
3) Priorizar políticas y heurísticas que pueden impactar de forma positiva en la generación de individuos, como ser:
   - SPT Shortest Processing Time: Aplicado a las operaciones con menor tiempo de procesamiento y respetando las restricciones de ordenamiento entre operaciones.
   - LPT Longest Processing Time: Asigna operaciones por mayor tiempo de procesamiento, respetando el orden de ejecución de cada operación.
   - LMB Load Machine Balance: Brindar mayor prioridad a las operaciones que permitan un mejor balance de la utilización de las máquinas. Esto es útil para no adjudicar operaciones únicamente a las máquinas con mejor performance, de esta forma se logra reducir el tiempo final de ejecución.

Para lograr mayor diversidad en la población, los valores serán creados de forma aleatoria; posteriormente se procede a la reparación del individuo en lo que refiere el tiempo de inicio de ejecución para cada operación.

D. Selección

Método de selección por torneo, de esta forma se eliminan las soluciones con valores de fitness no deseables, la diversidad se logra aplicando operadores evolutivos, de recombinación como de mutación.

E. Explotación: Recombinación

Para realizar el proceso de recombinación, se seleccionan dos padres $A = (A^1, A^2, .., A^K)$ y $B = (B^1, B^2, .., B^K)$, donde $K = \sum_{j=0}^{J} cantidad\_operaciones(j)$ según el método de selección propuesto en el ítem anterior. El operador de recombinación consiste en seleccionar de forma aleatoria dos trabajos i y j, con $i \leq j$, y recombinamos los hijos de forma tal que $C = (A^1, ..A^{i-1}, B^i, .., B^j, A^j, .., A^K)$ y de forma análoga procedemos para el segundo hijo $C = (B^1, ..B^{i-1}, A^i, .., A^j, B^j, .., B^K)$. Recordar que cada trabajo h posee K_h operaciones, dada la codificación se obtiene que M_{h,x_h} es de la forma $(M_i, t_{inicial})$ por lo tanto cuando aplicamos el operador de recombinación debemos tener en cuenta el primer término y recalcular el segundo luego de finalizada la operación, se obtienen individuos que cumplen con las restricciones del problema propuesto. Se presenta un pseudocódigo para la solución del operador recombinación.

$S_1 = select(S)$
$S_2 = select(S)$
$i \leftarrow random(1, K)$
$j \leftarrow random(1, K)$
for $c = i$ to j **do**
  $swap \leftarrow S_1[c]$
  $S_1[c] \leftarrow S_2[c]$
  $S_2[c] \leftarrow swap$
end for
$repair(S_1)$
$repair(S_2)$

F. Exploración: Mutación

Si bien en la propuesta inicial se mencionó la implementación de dos operadores de mutación, se realizó la implementación de un operador específico, puesto que el algoritmo a comparar no presenta operador secundario, se considera oportuno que la implementación de un operador es suficiente para brindar diversidad al problema. Explorar es un proceso importante para generar diversidad en el conjunto de soluciones, para este caso se implementa un operador de mutación aleatorio que brinda mejoras tanto a nivel de calidad de la solución como eficiencia computacional, dado que el proceso es sencillo de implementar y eficiente computacionalmente hablando, además de agregar diversidad al problema. El operador de mutación en cuestión se representa de la siguiente forma:

1) Seleccionar dos operaciones $i, j, 0 \leq j, i \leq K, i \neq j$, con $K = \sum_{j=1}^{J} cantidad\_operaciones(j)$.
2) Intercambiar el valor de trabajo de las celdas anteriormente seleccionadas.
3) Reparar la solución, de forma de que sea factible para el problema en cuestión.

Observar que al aplicar el operador de mutación el nuevo individuo cumple con las restricciones de ordenamiento, se aplica mecanismos de reparación y verificación para recalcular los tiempos de inicio de ejecución de cada operación de modo tal que se evita que en una máquina se ejecutan dos procesos al mismo tiempo.

El planteo para la mutación se puede apreciar con el pseudocódigo a continuación:

$i \leftarrow random(1, K)$
$j \leftarrow random(1, K)$
$swap \leftarrow S_1[i]$
$S_1[i] \leftarrow S_1[j]$
$S_1[j] \leftarrow swap$
$repair(Mutation)$

Observar la simplicidad del operador de mutación, computacionalmente es eficiente, se obtienen resultados que satisfacen los requerimientos en cuanto a performance. Brinda diversidad al problema, con lo cual en conjunto con el método de solución y el operador de recombinación la convergencia a una solución óptima ocurre de forma inmediata.

G. Verificación factibilidad y mecanismo de reparación

Si bien los operadores de recombinación y mutación no generan soluciones no factibles desde el punto de vista del orden de ejecución de las operaciones, se debe reparar los tiempos de inicio de ejecución para cada operación una vez se ha aplicado mutación/recombinación.

A continuación se presenta el 'pseudocódigo' relativo al mecanismo de reparación, $Dline_{Jobs}$ y $Dline_{Mach}$ son vectores auxiliares que contienen el tiempo de deadline de los trabajos y máquinas respectivamente.

$Dline_{Jobs} \leftarrow \vec{0}$
$Dline_{Mach} \leftarrow \vec{0}$
$i \leftarrow 1$
for $i \leq I$ **do**
  $j \leftarrow 1$
  for $j \leq J$ **do**
    $Mach \leftarrow machine(M_{j,i})$
    if $Dline_{Jobs}[j] > Dline_{Mach}[Mach]$ **then**
      $M_{j,i} \leftarrow (Mach, Dline_{Jobs}[j])$
    else
      $M_{j,i} \leftarrow (Mach, Dline_{Mach}[M]$
    end if
    $Dline_{Jobs}[j] += C_{j,i,Mach}$
    $Dline_{Mach}[Mach] += C_{j,i,Mach}$
  end for
end for

Observar el orden de ejecución, se debe recorrer todas las operaciones para garantizar la reparación de la solución, si bien resulta un mecanismo costoso también se debe tener el cuenta que la mutación es un operador secundario en la implementación del problema, con lo cual la probabilidad de utilizar este operador es menor, lo cual implica que el costo al aplicar la reparación es despreciable para ese caso.

H. Características CHC

Brevemente se describe las características del algoritmo CHC implementado, se describe con detalle el mecanismo de reinicialización de la población.

1) Codificación: Se trata de una codificación binaria, precisamente un arreglo de tamaño $\sum_{i=1}^{O} i$ donde $O = cantidad\_operaciones$, el tamaño se deduce de las relaciones entre las operaciones, es decir la precedencias entre las mismas, dada una operación j y una operación k, si $j < k$ y j tiene precedecia sobre k entonces la precedencia entre las mismas se ve denotada por $(j, k) = 1$, indicando con un 1 en caso que el item de la izquierda tenga la precedencia sobre la derecha.

2) Función Fitness: Análogo al caso del algoritmo propuesto anteriormente, la diferencia significativa que posee es que debe transformar a partir de la codificación binaria un vector con el orden de procesamiento de los trabajos, de este modo se realiza la operación de fitness de forma idéntica al caso de propuesto para el algoritmo 'GA'.

3) Inicialización: Consiste en una inicialización aleatoria, posteriormente aplicando mecanismo de reparación que se describe en subsección posterior. Básicamente para el tamaño del problema se sortea aleatoriamente un número binario y posteriormente se verifica la factibilidad de la solución, en caso de no ser factible a las restricciones del problema, se procede a la reparación de la misma.

4) Selección: Se trata de una selección elitista.

5) Explotación: Recombinación: Se utiliza operador de recombinación HUX.

6) Verificación factibilidad y mecanismo de reparación: Se debe enunciar este proceso ya que el mismo es costoso, y provoca que los tiempos de ejecución puedan llegar a ser extensos. La descripción del mismo se encuentra en el artículo [1]. El proceso se llama 'Local Harmonization' y consiste en generar una matriz $M[cantidad\_operaciones, cantidad\_operaciones]$ y contabilizar las prioridades de las operaciones y de esta forma generar un orden total entre las mismas. La idea básica consiste en que alguna operación tendrá prioridad sobre las demás, otra operación obtendrá prioridad sobre todas excepto la anterior descrita, así sucesivamente hasta obtener la última operación que no tiene prioridad sobre ninguna. Una vez terminado el conteo de las prioridades de las operaciones, puede ocurrir que se generen ciclos entre las mismas, y en ese caso se le debe brindar prioridad a una de ellas. Una vez contabilizado la prioridad para cada operación, se toma la operación que posee la mayor prioridad y se asigna prioridad 1 ante todas las demás operaciones, entonces esta operación es la primera en ser ejecutada sobre las demás. Se procede de forma análoga con la segunda operación indicando prioridad sobre las demás operaciones de forma tal que no tenga prioridad sobre la máxima, de esta forma se obtiene la segunda operación con mayor prioridad. Así sucesivamente hasta llegar a la última operación la cual no tendrá prioridad sobre ninguna. Al tratarse de una matriz antisimétrica una vez se van reparando los prioridades sobre las operaciones partiendo de la matriz triangular superior derecha, las demás filas también irán gradualmente corrigiendo las prioridades, por lo que el proceso se irá ajustando de forma gradual hasta generar el orden total de las operaciones. Una vez culminado el proceso se obtiene el orden total de las operaciones, que será de gran utilidad para el cálculo del fitness y para obtener

una solución que sea satisfactible al problema.

7) Reinicialización: En este mecanismo se describe como se procede a reinicializar la población una vez el conjunto de soluciones es convegente. Primero se presenta el criterio para determinar la convergencia dado un espacio de soluciones. Para esto se toman dos variables que determinarán si existe o no convergencia en el espacio de soluciones planteado, una variable llamada d la cual actúa como contador para determinar si existe o no convergencia. La segunda variable se llama s e indica si la población actual diverge o no, en caso de hacerlo se denotará el valor de $s = -2$ y se procederá al procedimiento de reinicialización de la población. Para el caso en que $s = 0$, se dispone de un vector ordenado de forma decreciente, de modo que para $s = 0$ se comparan todos los individuos con el último para determinar si alguno de ellos es mayor que el último, el único caso posible para que esto no suceda es que todos sean iguales, y entonces se decrementa el valor de s en una unidad, caso contrario(existe un individuo mayor que el último) se incrementa s en una unidad. Si $s = -1$ se decrementa d y aquí puede ocurrir dos casos si $d < 0$ entonces decrementamos s en una unidad y reinicializamos d en función de un parámetro de población fijo, para este caso lo llamamos $r = 0.9$, y en este punto estaríamos indicando que $s = -2$ por lo cual debería reinicializarse la población; caso contrario que $d > 0$, s adopta el valor 1, brindando de esta forma una nueva oportunidad a la población para no reinicializarse. Si $s > 0$ se incrementa en una unidad y se procede el ciclo de actualizaciones posteriores. Una vez $s = -2$ esto significa que todos los individuos poseen el mismo fitness y además de esto se le brindó d oportunidades a la población aplicando cruzamiento para diversificar el conjunto; puesto que esto no se logra, se procede a reinicializar la población con un criterio sencillo, tomamos los individuos de esa muestra y aplicamos operador de divergencia con una probabilidad de reinicialización llamada P_R a cada individuo de la selección. Básicamente reinicializamos la población alterando bits en cada individuo, con distintas probabilidades se logra tener un gran espectro de reinicializaciones, con probabilidades altas obtendremos gran parte de las características del individuo que pertenece al grupo de convergencia, pero en su totalidad el valor de fitness será distinto, por lo cual la diversidad estará asegurada y al mismo tiempo permitirá que en gran parte de los casos se mantengan las características deseables de los individuos que presentan la solución óptima al problema.
Con esto se culmina la explicación de las características del CHC implementado, y se procede directamente a la explicación y resultados obtenidos en la evaluación experimental.

V. Evaluación Experimental

Para este punto se tendrá en cuenta resultados numéricos que permitan evaluar la calidad de la solución al problema, así como la eficiencia computacional del algoritmo propuesto. Para ello se propone veinte instancias del problema y para cada una de ellas se procede a la ejecución de los algoritmos en cuestión treinta veces. Para cada instancia del problema se propone el reporte de los siguientes datos:

- Mejor fitness.
- Valor promedio de los mejores fitness obtenidos en las ejecuciones.
- Desviación estándar de los mejor fitness.
- Tiempo promedio de ejecución.
- Mejor tiempo ejecución.

Nombre	Operaciones(JxM)	F_{BEST}
abz5.fjs	10 x 10	1234
abz6.fjs	10 x 10	943
abz7.fjs	20 x 15	656
abz8.fjs	20 x 15	665
abz9.fjs	20 x 15	679
la01.fjs	10 x 5	666
la02.fjs	10 x 5	655
la03.fjs	10 x 5	597
la04.fjs	10 x 5	590
la05.fjs	10 x 5	593
la06.fjs	15 x 5	926
la29.fjs	20 x 10	1153
mt06.fjs	6 x 6	55
mt10.fjs	10 x 10	930
mt20.fjs	20 x 10	1165
orb1.fjs	10 x 10	1059
orb2.fjs	10 x 10	888
orb3.fjs	10 x 10	1005
orb4.fjs	10 x 10	1005
orb5.fjs	10 x 10	887

TABLE I: Instancias

Se procede con un análisis estadístico de los datos, para ello se debe analizar la distribución de los valores de fitness obtenidos en cada instancia. Se debe plantear como hipótesis nula que la distribución en cuestión sea normal, para ello se utilizará test de D'Agostino para determinar si el comportamiento del espacio muestral obtenido puede asociarse con una distribución normal.

A. Instancias

Es importante contar con número razonable de instancias de un problema dado para realizar el análisis pertinente. Para ello se utilizaron las instancias mencionadas en la tabla I. Además se presenta los mejores valores fitness encontrados hasta el momento[7]. Los resultados obtenidos en la ejecución se agregan en referencia.[6]

B. Calidad de Soluciones

La evaluación de la calidad del algoritmo se deduce del estudio de los siguientes valores:

- Valores promedio de fitness: este dato resulta importante en el momento de comparar algoritmos, más adelante se describirá la metodología a tener en cuenta para comparar algoritmos, así como también se dará a conocer qué algoritmo será utilizado para compararlo con el propuesto. Si el valor promedio de fitness del algoritmo presentado es menor que el valor promedio(en caso de tratarse de comparación entre algoritmo no determinista) o menor al valor obtenido por un algoritmo determinista, éste indicador en conjunto con otros indicadores como ser GAP pueden determinar el porcentaje de mejora del algoritmo presentado.

- Promedio número de generaciones necesarias para alcanzar el mejor resultado: A través de los datos obtenidos se debe establecer qué número poblacional es adecuado para lograr la mejor solución, el promedio es una cota inferior a utilizar; el indicador es de vital importancia ya que la calidad de la solución depende del número de generaciones. Utilizar un número de generaciones elevado aproxima a una mejor solución, pero en cambio se degrada la calidad de eficiencia computacional, ya que se requiere de mayor tiempo de cómputo. Estos dos factores implica que se debe establecer un balance correcto entre la mejora de la solución en comparación con otros algoritmos y la eficiencia computacional entre ellos.
- Representación gráfica de los mejores valores de fitness, así como valores promedio para todos los casos propuestos: La idea consiste en concluir a través de la consolidación de los datos gráficamente sobre qué casos representativos el algoritmo obtiene buenos resultados en función de la calibración de parámetros como ser las probabilidades de recombinación, mutación y cardinalidad de la población. El estudio gráfico puede llevar a deducir una relación implícita en el ajuste de los parámetros, de esta forma ajustando dichos valores se puede mejorar la calidad de la solución o reducir el tiempo de computación y por ende el aumento de la eficiencia computacional del algoritmo.

C. Eficiencia Computacional

Para este punto se debe conocer el tiempo de ejecución promedio para cada instancia del problema, así como su desviación estándar. En cuanto a las plataformas que se ejecutaron todas las instancias del problema, se contó con lo siguiente:

1) PC1:

- CPU: model name : Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz
  cache size : 3072 KB
  physical id : 0
  cpu cores : 2
- Memoria: MemTotal:5991936 kB (6 GB)
- Sistema Operativo: Ubuntu 14.04 LTS
  kernel: 3.13.0
  GNU-Compiler(gcc) 4.8
- Lenguaje de desarrollo: c++ [3] con librería malva[4], conjunto de librerías g++11.

D. Análisis comparativo de algoritmos

Se cuenta con los algoritmos propuestos a continuación:

1) GA: Implica el algoritmo evolutivo propuesto para la resolución del problema, con operadores descriptos, en particular con operador de mutación aleatorio.
2) CHC: Con operador de cruzamiento HUX, utiliza una representación binaria de la solución del problema. [1]

En cuanto a la eficiencia computacional, se propone una comparación entre los algoritmos propuestos y los mejores

Instancia	**#Op**	T_{AVG}	T_{AVG}
mt06	36	1.293	6.486
la01	50	1.671	8.967
la02	50	1.679	10.438
la03	50	1.680	10.114
la04	50	1.674	9.656
la05	50	1.683	10.338
la06	75	2.382	18.700
abz5	100	3.120	49.228
abz6	100	3.130	48.764
mt10	100	3.070	45.640
mt20	100	3.124	28.960
orb1	100	3.080	44.237
orb2	100	3.126	56.199
orb3	100	3.083	49.598
orb4	100	3.128	44.281
orb5	100	3.134	49.551
la29	200	6.140	167.773
abz7	300	9.533	644.402
abz8	300	9.471	564.869
abz9	300	9.498	544.018

TABLE II: Caso 27, $P_C = 0.8, P_M = 0.1, P_R = 0.9, \#ind = 200$

casos en el cual se obtuvo mejor fitness promedio para cada algoritmo. Para el caso de GA la mejor combinación de parámetros corresponde al caso 27 para el algoritmo CHC el mejor caso reportado fittness promedio fue el número 20. Se obtiene la siguiente gráfica (ver Figura 1) de datos, ordenado por la cantidad de operaciones en cada instancia, y a continuación la gráfica pertinente de cómo evoluciona el tiempo en función de la cantidad de operaciones de la instancia. Se utilizó una escala logarítmica para graficar los datos en el eje de las ordenadas. Observar que el tiempo crece exponencialmente en el caso de CHC, y linealmente para el caso de GA. Dado los resultados obtenidos luego de las ejecuciones, se obtienen mejores tiempos de ejecución con el algoritmo propuesto(GA) por la representación de la solución y debido a que los mecanismos de verificación de factibilidad y reparación son eficientes en comparación a los del algoritmo CHC.

Los valores reportados para el caso 27 se pueden observar en la tabla V-D.

Para deducir el el tiempo de ejecución en función de la cantidad de operaciones se aproximó una función exponencial para el caso de CHC y para el algoritmo GA se trazó una función lineal, ambas funciones se encuentran en la gráfica, con los parámetros de cada en el caso que se desee aproximar el tiempo estimado dado una instancia con mayor cantidad de operación. Cabe destacar que en eficiencia relativa al costo de ejecución el algoritmo propuesto(GA) es superior a CHC, implica fuertemente que la representación seleccionada ante la binaria, logra transformar resultados exponenciales en lineales.

1) GA vs CHC: Para entrar en contexto se sabe que los algoritmos genéticos (y en particular el algoritmo propuesto GA) enfatiza la importancia del operador de recombinación utilizándolo como operador primario mientras que se tiene un operador de mutación como mecanismo secundario secundario. Sin embargo el CHC implementado que utiliza selección elitista y un operador de cruzamiento HUX no utiliza mutación. Esto lleva consigo a crear algún método que

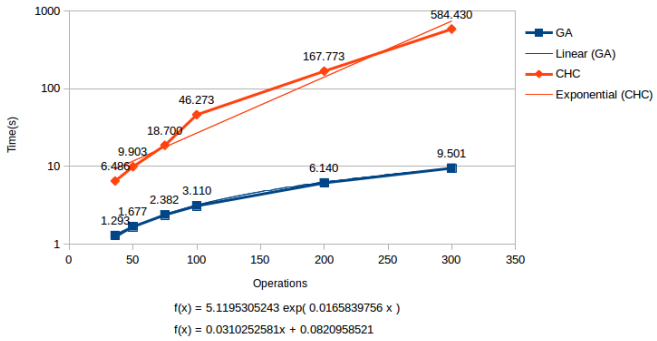

Fig. 1: $GA(27) vs. CHC(20)$

mantenga la diversidad; para ello se utiliza un mecanismo de re inicialización "secundario" que se aplica cuando se detecta convergencia en el conjunto de soluciones. Esta comparación se realizará en igualdad de condiciones, con el objetivo de determinar si la presencia de un operador de mutación logra mejorar a gran escala los valores de fitness obtenidos. Básicamente la idea es contrastar dos aspectos importantes en cada algoritmo, a lo que refiere representación de la solución y la inclusión de un operador evolutivo secundario. El análisis consistirá en observar el impacto de dichos factores en la eficiencia computacional como la calidad de la solución. Se establecerá si el operador secundario es determinante en la calidad de la solución, así como la codificación puede tener un impacto positivo en la eficiencia computacional. En resumen, la evaluación consistirá:

1) Determinar si la diferencias entre los mecanismos utilizados para obtener diversidad en los algoritmos conllevan soluciones diferentes y ver cuáles de ellas obtuvo mejores soluciones globales.
2) Determinar si la codificación de la solución presenta alto impacto, en eficiencia computacional como en la calidad de las soluciones. Teniendo en cuenta que la codificación del algoritmo GA es un arreglo con la cantidad total de operaciones de una instancia del problema; en cambio CHC la codificación es binaria, la cual promete obtener con el operador de recombinación un conjunto de soluciones con la calidad esperada.

2) Análisis estadístico: Los procedimientos a realizar para determinar la mejor calidad de un algoritmo frente a otro, dependerán de las distribuciones probabilísticas de los valores obtenidos para cada algoritmo. Se utilizará tests paramétricos en el caso que ambos conjuntos de muestras de los algoritmos involucrados posean una distribución normal. Para ello se procede a realizar test de normalidad de D'Agostino, posteriormente se realizará test t-Student utilizando GNU-OCTAVE [5] para determinar si existen diferencias significativas entre las medias de los conjuntos en cuestión, en caso las instancias presenten un comportamiento bajo una distribución normal. En caso de no tratarse de distribuciones normales, el procedimiento será análogo pero el objetivo es analizar la existencia de una diferencia significativa entre las medianas de los conjuntos de muestra, para ello se utiliza test de Mann-Witney(s-Student).

Una forma de poder comparar los algoritmos, puede ser simplemente observar los valores fitness hallados por un algoritmo y por el otro. Como podemos apreciar el algoritmo propuesto(GA), obtiene resultados notablemente mejores, es decir un fitness mejor en promedio e individual, que el CHC en todos los conjuntos de entradas testeados y para cualquier combinación paramétrica. Este resultado nos lleva a concluir que el algoritmo propuesto, en término de calidad de soluciones, obtiene mejoras a las soluciones buscadas.

a) Descripción Tablas: El análisis estadístico será adjunta como parte de la entrega, esto se debe a la gran cantidad de tablas que presenta el mismo. Se trata del archivo Statistical_Analysis.pdf, el cual posee la siguiente información relevante:

- Tablas fitness GA : En esta sección se presenta todos los valores de fitness encontrados en las 30 ejecuciones realizadas, así como el fitness promedio y el mínimo encontrado por cada instancia del problema propuesto.
- Tablas tiempos GA : Análogo al punto anterior, pero aplicado al tiempo de ejecución, incluye tiempo promedio para cada instancia al igual que el tiempo mínimo de ejecución.
- Tablas fitness CHC: Análogo al punto 1 pero aplicado al algoritmo CHC.
- Tablas tiempos CHC: Análogo al punto 2 pero aplicado al algoritmo CHC.
- Análisis del algoritmo GA: En esta sección se describen los datos obtenidos mediante en cada configuración de parámetros propuesto. Las columnas de las tablas para esta sección son $Instancia$ indica que instancia del problema corresponde los resultados; F_{OPT} es el mejor valor de fitness conocido hasta el momento; F_{BEST} es el mejor fitness encontrado mediante la ejecución del algoritmo; F_{AVG} es el fitness promedio para la instancia dada; σ es la desviación estándar; T_{AVG} es el tiempo promedio de ejecución; DA es el estádistico de D'Agostino para dicha instancia y $Normal$ describe dado el valor de DA si la distribución es normal, es decir si DA se encuentra en el intervalo $[0.2662, 0.2866]$, dado $\alpha = 0.05$ y un tamaño de muestra $N = 30$.
- Análisis del algoritmo CHC: Análogo al item anterior pero aplicado a CHC.
- Comparación GA - CHC: Se trata de una tabla que compara los atributos de calidad entre ambos algoritmos; F_{BEST} describe el mejor valor fitness encontrado por el algoritmo en cuestión, F_{AVG} es el fitness promedio, σ la desviación estándar para la instancia, y T_{AVG} es el tiempo promedio de ejecución según el algoritmo en cuestión.
- s-student y t-student: Para en análisis estadístico se realizaron los test s-student para el caso de que las distribuciones de la instancia no sean normales, y t-student para el caso que ambas distribuciones lo sean. Para ello se describe mediante la columna $p-value$ el p-valor correspondiente al resultado del test, y la columna $Test$ indica el test realizado en ese caso. Se aplica s-student se aplica para analizar cual de las distribuciones

presenta una mayor mediana, análogamente se utiliza t-student para analizar la diferencia entre la media de dos distribuciones normales, en este caso no interesa conocer quién posee la menor media, ya que el objetivo es minimizar el valor de fitness. Para el resultado de $p - valor$ lo interpretamos de la siguiente forma; la hipótesis nula que tomamos es la media de GA es mayor a la media de CHC, para el caso del test t-student, análogamente para el caso de las medianas pero considerando la hipótesis nula como la mediana de GA es mayor a la de CHC; entonces tenemos si $p - value < \alpha$, con $\alpha = 0.05$, entonces rechazamos la hipótesis nula; y por lo tanto podemos deducir que efectivamente la media o la mediana de la muestra de la izquierda(GA) es menor que la muestra de la derecha(CHC), dependiendo del test realizado. En caso contrario no podemos afirmar que la hipótesis alternativa es cierta, pero en la mayoría de los casos se termina rechazando la hipótesis nula o el $p - valor$ es próximo a uno y terminamos aceptando la hipótesis alternativa.

- Caso 28: El caso 28 de prueba corresponde a la selección de la mejor configuración de parámetros encontrada, para el CHC corresponde a la configuración de parámetros 20, mientras que para GA corresponde a la configuración 27; para más detalles ver sección V-E1.
- Todos los casos: La última tabla del documento en cuestión realiza una comparación a nivel macro, involucrando el fitness promedio(F_{AVG}) y el tiempo promedio(T_{AVG}) para cada configuración de parámetros. Para tener una noción sobre como evoluciona el comportamiento tanto a nivel de calidad de la solución promedio como en la eficiencia computacional expresada mediante el tiempo promedio de ejecución.

3) GAP: Se utilizó también un indicador que es utilizado para hacer comparaciones y saber cuan bueno es, en este caso, un algoritmo respecto a otro. El indicador es el GAP y lo definimos como $GAP = \frac{CHC-GA}{CHC}$. El estudio se hizo en base al algoritmo propuesto GA y a CHC; el análisis se llevó a cabo para la calidad de las soluciones y para la eficiencia computacional en términos de tiempo de ejecución.

Como se observa en la tabla V-D3, existe una mejora del algoritmo propuesto respecto al CHC en término del promedio del fitness hallado; donde en algunos casos se puede notar una mejora de hasta el 8% (0.08) como ser en el caso 9. Igual cabe aclarar que en ciertos casos como ser 19, 20 y 21 el CHC llegó a mejores promedios del fitness. La notable diferencia se encuentra aquí en los tiempos de ejecuciones de los algoritmos. Claramente el algoritmo propuesto llega a las soluciones en tiempos inalcanzables por el CHC, llegando a una diferencia del 97% (0.97) como ser los casos 19, 20 y 21.

Caso	F_{AVG}	T_{AVG}
1	0.0235	0.9263
2	0.0249	0.8961
3	0.0226	0.8854
4	0.0713	0.9082
5	0.0679	0.8985
6	0.0697	0.9011
7	0.0825	0.9066
8	0.0811	0.8955
9	0.0839	0.8899
10	0.0103	0.9583
11	0.0128	0.9513
12	0.0133	0.9467
13	0.0507	0.9555
14	0.0536	0.9496
15	0.0521	0.9423
16	0.0627	0.9521
17	0.0649	0.9499
18	0.0634	0.9396
19	-0.0021	0.9771
20	-0.0025	0.9742
21	-0.0002	0.9709
22	0.0347	0.9769
23	0.0333	0.9747
24	0.0357	0.9698
25	0.0431	0.9750
26	0.0445	0.9717
27	0.0488	0.9695

TABLE III: GAP CHC vs GA

E. Ajuste de Parámetros

Básicamente se realizó la ejecución de ambos algoritmos, con valores de parámetros que aporten calidad a la solución del problema. El análisis es exhaustivo por lo que para el algoritmo CHC se realizaron 27 ejecuciones de cada instancia del problema, combinando la probabilidad de cruzamiento (P_C), la probabilidad de reemplazo (P_R) y el tamaño de la población ($\#pob$). Para el algoritmo GA se procedió con 27 ejecuciones también de las 20 instancias propuestas.

1) Parámetros: A continuación se listan los parámetros en cuestión:

1) Tamaño de Población: El objetivo es trabajar con diversos tamaños de población y observar el comportamiento de los valores de fitness óptimos para cada caso. Se puede establecer en este punto una cota superior para el número de población para el cual, a mayor cardinalidad del conjunto los resultados presentan una variación despreciable. Para ello tomamos los siguientes valores $\#pob = \{50, 100, 200\}$.
2) Probabilidad de mutación: Se alternó el valor de probabilidad de mutación entre valores razonables $P_M = \{0.1, 0.05, 0, 01\}$. A partir de estos resultados se arriba a una conclusión respecto a la comparación de la mutación utilizado por GA y al operador secundario utilizado por CHC.
3) Probabilidad de cruzamiento: Diversificar la población con una probabilidad adecuada es importante para lograr mejores resultados en menor tiempo, por ello se utilizaron probabilidades del conjunto $P_C = \{0.6, 0.7, 0.8\}$.
4) Probabilidad de reemplazo: En el caso de existir convergencia, se utilizará un mecanismo de reemplazo de la población utilizando este parámetro de configuración, el mismo se aplicará a todos los individuos de la población, con la finalidad de generar una nueva población a partir de dichos parámetro. Se utilizaron los siguientes parámetros $P_R = \{0.5, 0.7, 0, 9\}$.

Instancia					GA		CHC	
	P_C	P_M	$\#ind$	P_R	F_{AVG}	T_{AVG}	F_{AVG}	T_{AVG}
1	0.6	0.01	50	0.5	981.486 7	2.461 7	1 005.150 0	33.415 6
2	0.7	0.01	50	0.5	982.276 7	2.817 9	1 007.321 7	27.116 7
3	0.8	0.01	50	0.5	981.823 3	3.204 7	1 004.481 7	27.976 0
4	0.6	0.05	50	0.7	939.166 7	2.596 4	1 011.248 3	28.294 3
5	0.7	0.05	50	0.7	940.508 3	2.954 5	1 009.015 0	29.103 2
6	0.8	0.05	50	0.7	939.506 7	3.340 3	1 009.871 7	33.764 5
7	0.6	0.1	50	0.9	927.600 0	2.770 2	1 011.010 0	29.670 7
8	0.7	0.1	50	0.9	927.245 0	3.138 2	1 009.085 0	30.031 4
9	0.8	0.1	50	0.9	925.511 7	3.525 5	1 010.275 0	32.010 9
10	0.6	0.01	100	0.5	971.583 3	2.527 6	981.670 0	60.637 4
11	0.7	0.01	100	0.5	969.475 0	2.892 2	981.996 7	59.396 5
12	0.8	0.01	100	0.5	969.131 7	3.270 3	982.243 3	61.333 0
13	0.6	0.05	100	0.7	936.066 7	2.674 3	986.008 3	60.147 5
14	0.7	0.05	100	0.7	932.565 0	3.040 1	985.381 7	60.267 6
15	0.8	0.05	100	0.7	935.060 0	3.425 2	986.473 3	59.321 1
16	0.6	0.1	100	0.9	921.988 3	2.835 7	983.706 7	59.216 8
17	0.7	0.1	100	0.9	920.906 7	3.221 2	984.770 0	64.276 1
18	0.8	0.1	100	0.9	922.305 0	3.629 0	984.773 3	60.087 7
19	0.6	0.01	200	0.5	965.265 0	2.753 1	963.235 0	120.062 3
20	0.7	0.01	200	0.5	964.285 0	3.108 9	961.920 0	120.610 9
21	0.8	0.01	200	0.5	962.735 0	3.460 3	962.573 3	119.065 3
22	0.6	0.05	200	0.7	932.261 7	2.810 4	965.775 0	121.898 2
23	0.7	0.05	200	0.7	932.070 0	3.190 3	964.196 7	125.922 5
24	0.8	0.05	200	0.7	929.155 0	3.669 0	963.553 3	121.572 4
25	0.6	0.1	200	0.9	922.281 7	3.064 7	963.866 7	122.529 6
26	0.7	0.1	200	0.9	919.945 0	3.340 0	962.833 3	118.003 8
27	0.8	0.1	200	0.9	916.658 3	3.734 8	963.673 3	122.399 9
28	0.8	0.1	200	0.9	901.173 3	18.798 9	959.650 0	502.775 3

TABLE IV: Ajuste de parámetros

Observar que combinar todos los parámetros es una tarea costosa, de todas formas como estamos ante un análisis exhaustivo de los datos, se procedió a la ejecución de todas las combinaciones posibles, logrando un alto espectro de combinación de parámetros. Una vez que se obtuvo la mejor combinación posible se realizó una ejecución con un número de generaciones lo suficientemente elevado para llegar a mejores soluciones obtenidas anteriormente. Si bien se utilizó un numero de generaciones razonable para todas las instancias en cuestión, en algunas instancias no se llegó a la mejor solución conocida hasta el momento. Es por ello que una vez definida la mejor combinación de parámetros, se procede a ponderar un número de generaciones elevado para aquellas instancias donde no se alcanzó el óptimo. Para cada combinación de parámetros se procedió a realizar una gráfica comparativa para cada algoritmo detallado. Consiste en representar dado un número de población, el valor de firness global a partir de los parámetros de cruzamiento y mutación, para el caso del algoritmo GA, en cambio para el algoritmo CHC se altera los valores de la probabilidad de cruzamiento y la probabilidad de reemplazo.

Básicamente como se describió anteriormente, se realizaron 27 combinaciones de parámetros, cada una de ellas con 20 instancias del problema ejecutadas treinta veces para obtener un conjunto de datos razonables para realizar el análisis estadístico pertinente. En la tabla se resume todos los resultados logrados de las 27 combinaciones, comparando los dos algoritmos propuestos; los parámetros de calidad seleccionados fueron el tiempo promedio de procesamiento y el fitness promedio de todas las instancias. La tabla IV muestra dados los parámetros de configuración seleccionados, los ítems de calidad mencionados.

Como se mencionó anteriormente, una vez se obtienen las posibles combinaciones con los parámetros mencionados, se deja a disposición todos los valores obtenidos para dicho caso. Cabe mencionar también que los datos de todos los casos citados en el presente informe se encuentran adjuntos en el archivo Statistical_Analysis.pdf, de esta forma se agrega información extra de los resultados obtenidos, con los parámetros correspondientes. Se puede observar que el mejor fitness promedio se obtuvo en el caso 27, donde los parámetros son $P_C = 0.8$, $P_M = 0.1$ y $\#ind = 200$; es decir, con la mayor probabilidad de cruzamiento y mutación dentro de las probabilidades designadas. Cabe aclarar también que el mejor promedio de fitness se obtuvo con la mayor cantidad de individuos. Si observamos detenidamente el archivo Statistical_Analysis.pdf podemos ver que a igual P_C y P_M en caso del algoritmo GA y a mayor $\#ind$ el promedio de fitness mejora; al igual el CHC cuando fijamos P_C y P_R a mayor $\#ind$ el promedio de fitness mejora. Esto nos lleva a concluir que la cantidad de individuos efectivamente es un factor determinante en la búsqueda del mejor fitness. Para el caso 27 mencionado anteriormente se adjunta la tabla de valores [VII] correspondiente a las ejecuciones para cada instancia, así como el tiempo, fitness promedio, el mejor valor de fitness obtenido de las ejecuciones y el valor DA estadístico de D'Agostino que se obtuvo en cada caso, el cual determina si los datos son normales, para este caso se tomó como referencia $\alpha = 0.05$ y un tamaño de muestra $N = 30$, para que la muestra sea normal, el valor estadístico DA para cada caso debe pertenecer al intervalo $[0.2662, 0.2866]$, caso contrario se trata de una muestra que no sigue dicha distribución probabilística. Otro dato a tener en cuenta es si con el algoritmo propuesto se arriba a la mejor solución encontrada hasta el momento. Para el caso 27 cuya configuración de parámetros reporta mejor fitness promedio, se arriba a las mejores soluciones para las instancias listadas a continuación:

- la01
- la05
- la06
- mt06

Para las demás se pretende realizar una ejecución que involucre un mayor número de generaciones, con al finalidad de aproximarse o alcanzar los valores óptimos conocidos hasta el momento.

Esto se debe a que la cantidad de operaciones para estas instancias son mayores o iguales a 100, para remediar este caso se ejecutará 30 veces cada instancia mencionada con un número de generaciones mayor, en este caso 5000 en contraste a 1000 utilizadas en todos los casos. De esta forma, ejecutando el algoritmo con la mejor configuración de parámetros, puede mejorarse los resultados a un costo computacional lineal. La tabla V, muestra los valores de fitness óptimos obtenidos por el algoritmo GA.

Observar que incrementando el número de generaciones a 5000 los resultados obtenidos para las instancias se aproximaron al óptimo, pero no se alcanzó en algunas instancias, esto se debe a la complejidad de las instancias los resultados más alejados del óptimo se encuentran en las instancias $abz7$, $abz8$ y $abz9$, donde la cantidad de operaciones es de 300. Para estos casos se debe incrementar el número de generaciones, hasta converger a la solución óptima, si bien este procedimiento agrega un costo extra a nivel computacional, podemos asumirlo para las instancias que poseen más operaciones, como las mencionadas anteriormente.

Para el caso de CHC, se ejecutaron todas las instancias con un número de generaciones de 5000, en este caso se mejoraron

Instancia	F_{OPT}	F_{BEST}	F_{AVG}	T_{AVG}
abz5	1234	1263	1 311.300 0	15.727 6
abz6	943	947	981.900 0	15.552 5
abz7	656	737	779.766 7	47.824 8
abz8	665	758	802.200 0	47.635 5
abz9	679	800	834.166 7	47.845 5
la01	666	666	673.333 3	8.322 5
la02	655	655	680.566 7	8.537 9
la03	597	606	631.933 3	8.522 0
la04	590	590	606.866 7	8.557 9
la05	593	593	593.000 0	8.581 4
la06	926	926	926.000 0	11.897 9
la29	1153	1276	1 347.300 0	30.674 8
mt06	55	55	55.166 7	6.474 1
mt10	930	1005	1 056.600 0	15.684 4
mt20	1165	1228	1 315.666 7	15.754 6
orb1	1059	1119	1 195.033 3	15.822 3
orb2	888	909	958.333 3	15.843 7
orb3	1005	1086	1 173.066 7	15.501 0
orb4	1005	1046	1 088.833 3	15.591 6
orb5	887	946	1 012.433 3	15.626 7

TABLE V: $P_C = 0.8, P_M = 0.1, \#Individuos = 200, \#gen = 5000$

algunos óptimos de instancias, y el valor de fitness promedio al igual que en el caso de GA se redujo notoriamente. Sin embargo al aumentar el número de generaciones el algoritmo GA se comportó en la totalidad de las instancias mejor que CHC, que en una gran cantidad de casos en los tests de configuración de parámetros obtuvo mejores resultados individuales en algunas instancias. Los resultados obtenidos en la ejecución de 5000 generaciones para CHC se presentan en la tabla VI, a pesar de ejecutar con un mayor número de generaciones los resultados obtenidos con CHC superaron con una diferencia mínima los resultados obtenidos en el caso 20, con un costo computacional 5 veces superior. Esto se debe claramente a que la representación binaria en este problema específico conlleva a que los resultados converjan de forma lenta, el tamaño de las soluciones son demasiado complejas por lo tanto se requiere un número de generaciones muy por encima del sugerido para obtener valores semejantes al GA. Pero esto no quiere decir que CHC presente un comportamiento insuficiente, basta ver instancias individuales en las cuales CHC supera ampliamente a GA como ser en el caso 20 donde incluso en instancias de gran tamaño como ser $orb1$, $orb2$, $la29$ y $abz6$. Instancias como $la02$ CHC en casi todos los casos supera a GA, si tomamos como atributo de calidad el fitness promedio estamos ponderando el promedio de los casos, y no las instancias individuales, basta ver los mejores fitness en instancias individuales para deducir que en algunos casos CHC se comporta de mejor forma que GA. Sin embargo cuando ponderamos mayor número de generaciones, GA converge más rápido al óptimo que CHC en prácticamente todas las instancias. Podemos decir que el desempeño de CHC en instancias la supera levemente a GA, con lo cual podemos tomar como referencia este algoritmo para las instancias de este tipo cuando trabajamos con un orden de generaciones moderado(1000). Como se mencionó anteriormente no se pueden extraer conclusiones sobre el valor promedio de fitness, sin embargo cuando combinamos la calidad de las soluciones y la eficiencia computacional, GA en el segundo ítem tiene un mejor desempeño, de tal forma que se podría ejecutar con dicho algoritmo un número de generaciones lo suficientemente extenso como para alcanzar la mayoría de las soluciones óptimas en el mismo tiempo que se ejecuta CHC en un número bajo de generaciones. Otro punto interesante que puedo resultar relevante para el problema; el algoritmo CHC brinda una desviación estándar en todos los casos más ajustada que el algoritmo GA, esto acota el espacio y puede ser útil cuando queremos que dicha desviación sea minimizada, caso contrario es conveniente utilizar GA puesto que la desviación estándar para ese caso presenta un intervalo mayor, en el cual las soluciones varían. Puede ser un criterio a tener en cuenta cuando se selecciona el algoritmo para el problema.

Instancia	F_{OPT}	F_{BEST}	F_{AVG}	T_{AVG}
abz5	1234	1351	1 413.633 3	201.458 8
abz6	943	975	1 014.833 3	206.788 8
abz7	656	884	921.700 0	2 354.218 7
abz8	665	901	935.700 0	2 365.644 0
abz9	679	909	950.500 0	2 302.667 7
la01	666	688	694.100 0	47.670 9
la02	655	664	672.033 3	64.780 0
la03	597	622	640.666 7	46.690 0
la04	590	613	631.933 3	48.944 9
la05	593	593	593.000 0	50.559 3
la06	926	926	926.166 7	88.176 4
la29	1153	1448	1 516.866 7	774.162 5
mt06	55	55	55.000 0	36.557 4
mt10	930	1087	1 119.233 3	215.096 5
mt20	1165	1341	1 396.266 7	135.522 1
orb1	1059	1215	1 269.066 7	225.285 1
orb2	888	948	998.166 7	217.952 5
orb3	1005	1152	1 219.166 7	220.801 9
orb4	1005	1099	1 149.833 3	227.769 4
orb5	887	1023	1 075.133 3	224.758 9

TABLE VI: $P_C = 0.7, P_R = 0.5, \#Individuos = 200, \#gen = 5000$

1) En primer lugar, la representación seleccionada para GA es más eficiente computacionalmente que en el caso de CHC, esto se explicó anteriormente, pero con los datos experimentales se confirma la hipótesis inicial.
2) El mecanismo de reparación para que una solución sea válida en el algoritmo CHC es costosa, esto se debe a la representación de la solución, lo cual implica un proceso de 'local harmonization', el cual implica la construcción de matrices de amplias proporciones para verificar si la solución es factible, y en caso de no ser repararla. Es importante destacar que forzar soluciones provee mejores desempeños, para ello se adjunta documento que describe la eficiencia de CHC forzando soluciones que satisfacen el problema en lugar de invalidar la solución[1].
3) Sin dudas la representación es el principal factor que lleva a CHC a obtener resultados por debajo del algoritmo GA, ya que CHC requiere que la codificación de la solución sea binaria, esto repercute en la cantidad de bits que se deben dedicar para representar la solución, al tratarse de una relación entre operaciones, el tamaño de la solución es extensa, lo cual a nivel computacional requiere un tiempo de procesamiento mayor al del algo-

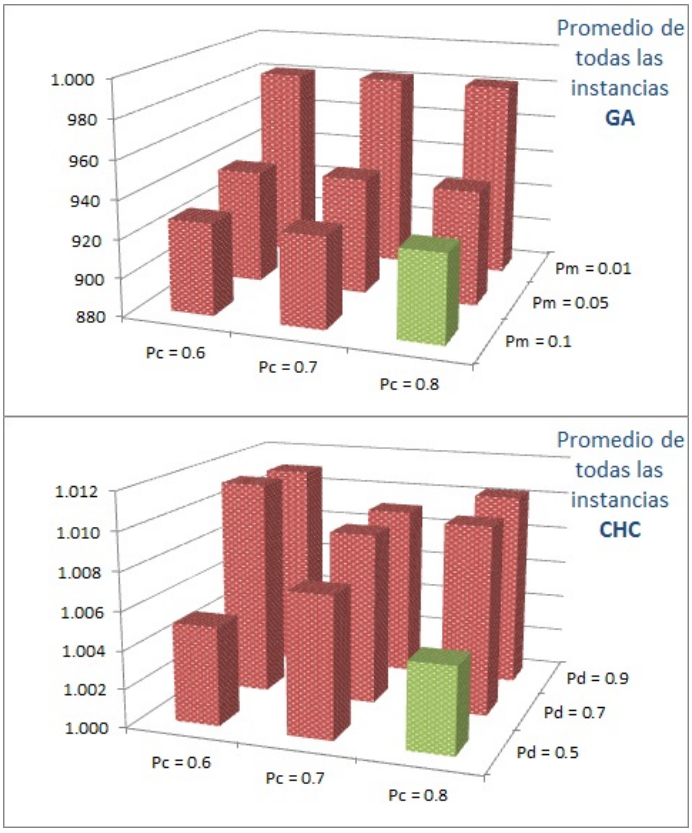

Fig. 2: $\#individuos = 50$

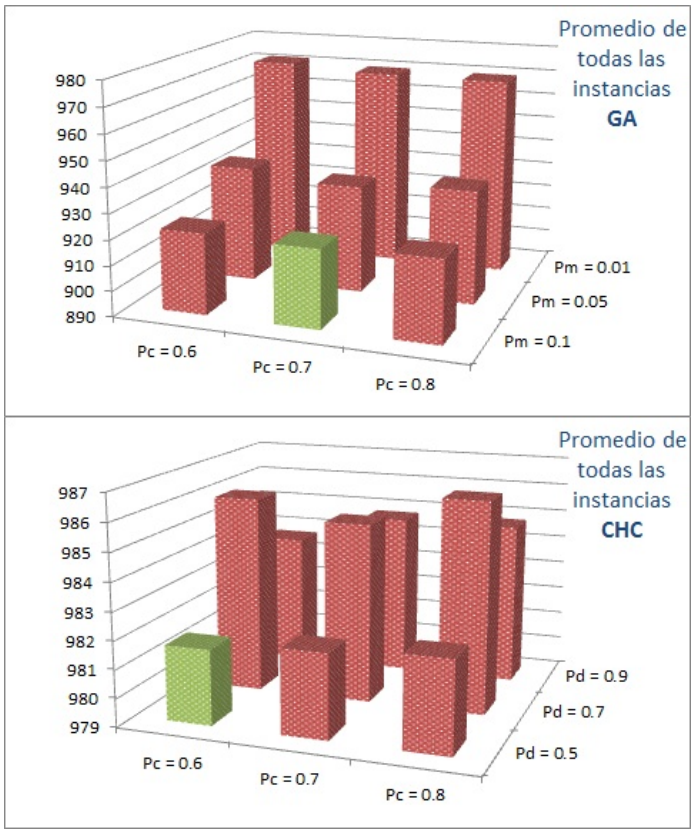

Fig. 3: $\#individuos = 100$

ritmo GA. Esto también tiene un impacto directo cuando se aplica operadores evolutivos como ser cruzamiento y reemplazo generacional en caso de convergencia.

F. Gráficas

Para la mejor representación de los datos, se adjunta para todos los casos propuestos de combinación de parámetros discriminado por $\#individuos$, se presentan las gráficas comparativas entre CHC y GA, comparando los valores fitness promedio para todas las instancias del problema.

En la figura 2 se obtiene la comparación de fitness promedio entre los algoritmos propuestos con $\#individuos = 50$, en función de los parámetros de mutación y cruzamiento. De forma análoga en la figura 3 para $\#individuos = 100$ y la figura 4 para $\#individuos = 200$.

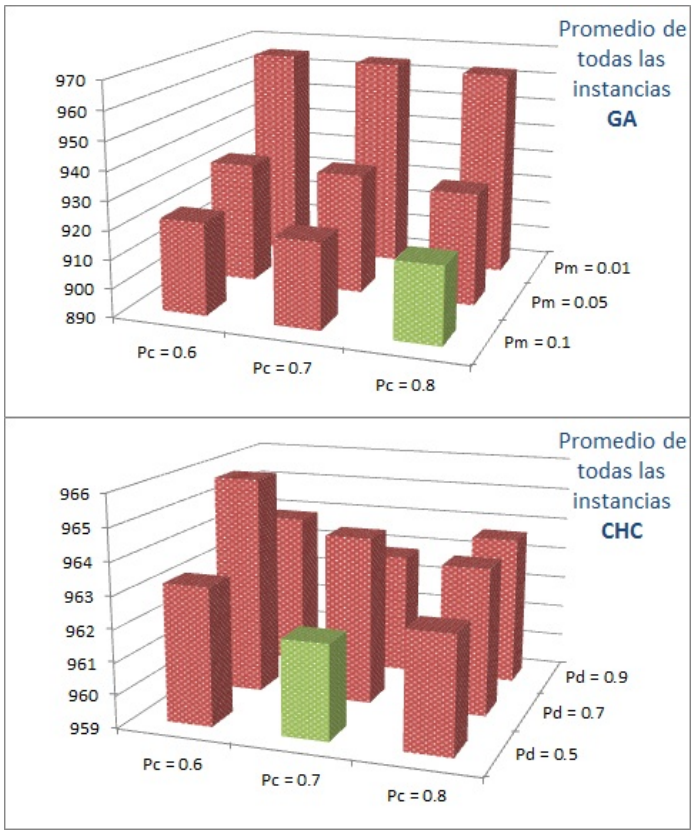

Fig. 4: $\#individuos = 200$

VI. Conclusiones

En el presente proyecto se plantea un algoritmo evolutivo que para el conjunto de casos de entradas testeado, es realmente eficiente computacionalmente hablando si lo comparamos con otro tipo de algoritmo; lo cual era uno de los principales objetivos planteados. El algoritmo consigue obtener una calidad elevada de soluciones al problema y conjunto a lo dicho anteriormente, en un tiempo más que satisfactorio. Teniendo en cuenta la probabilidad de mutación (P_M), la probabilidad de cruzamiento (P_C) y la cardinalidad de individuos ($\#ind$), podemos concluir que la mejor combinación de parámetros en la que se obtiene un mejor promedio de fitness es $P_M = 0.1$, $P_C = 0.8$ y a mayor $\#ind$ mejores resultados, para este caso $\#ind = 200$. Los algoritmos evolutivos contribuyen con los problemas considerados NP-difíciles brindando alternativas para llegar a soluciones cotizadas como 'muy buenas' en un tiempo bajo si es comparado con otras técnicas. De hecho podemos remarcar que el algoritmo propuesto presenta una mejora significativa en términos de tiempos, comparándolo con un algoritmo que utiliza la misma técnica, pero con una variante bastante importante, que es la no utilización de mutación. Podemos extraer una conclusión importante sobre la utilización de algoritmos evolutivos para la resolución de problemas; no es fácil predecir qué parámetros o criterios utilizar para solucionar el problema de forma satisfactible. Una conclusión importante es que el operador de mutación, agrega diversidad al problema y genera mejores resultados, basta comparar casos de prueba donde la probabilidad de mutación es menor, y los resultados son evidentes, a mayor diversidad, mejor es fitness promedio, por ende contribuye a obtener mejores soluciones globales. Para este tipos de problemas, la diversidad es sin lugar a dudas el factor determinante para obtener mejores soluciones, tanto en el algoritmo GA como CHC. A medida que se obtienen los resultados, se concreta la hipótesis primaria del problema, a mayor diversidad las soluciones mejoran, esto quedó plasmado cuando se alteraron los parámetros con el fin de brindar mayor diversidad al problema. En ambos algoritmos la hipoótesis es válida obteniendo mejoras notorias en el algoritmo CHC a medida que se ajustan los parámetros para obtener mayor diversidad, es decir, que la probabilidad de cruzamiento, mutación y reemplazo en el caso de CHC, así como la cantidad de individuos sea lo mayor posible. Si bien el conocimiento global del problema es un reto difícil de asumir, ya que el problema ha sido estudiado por muchos años, no se pretendía sobrepasar las mejores marcas, pero sí lograr mediante una propuesta computacionalmente eficiente arribar a resultados semejantes a los conseguidos hasta el momento. Se concluye que a pesar de no colmar todos los objetivos como ser superar valores óptimos, la experiencia implementando una solución a JSSP mediante algoritmos evolutivos, conduce que su utilización es completamente válida para cualquier otro problema NP-difícil. Por tanto, la utilización de este tipo de algoritmos es una buena estrategia a considerar en todos los problemas de esta índole.

VII. Anexo

Instancia	F_{OPT}	F_{BEST}	F_{AVG}	σ	T_{AVG}	DA	Normal
abz5	1234	1268	1 327.866 7	33.039 1	3.119 5	0.273 3	Si
abz6	943	948	997.466 7	35.663 9	3.130 0	0.267 0	Si
abz7	656	783	815.900 0	15.721 2	9.532 7	0.284 4	Si
abz8	665	774	829.366 7	27.588 0	9.471 2	0.279 1	Si
abz9	679	813	858.333 3	23.563 2	9.497 8	0.283 3	Si
la01	666	666	682.366 7	17.480 4	1.670 5	0.263 3	No
la02	655	662	691.300 0	23.472 2	1.679 0	0.267 6	Si
la03	597	617	637.700 0	16.927 6	1.680 4	0.264 1	No
la04	590	598	615.466 7	10.032 4	1.674 0	0.282 9	Si
la05	593	593	593.000 0	0.000 0	1.682 6	*	Si
la06	926	926	926.000 0	0.000 0	2.381 6	*	Si
la29	1153	1320	1 381.733 3	36.631 4	6.140 4	0.275 7	Si
mt06	55	55	56.400 0	1.665 3	1.292 5	0.258 9	No
mt10	930	1010	1 071.200 0	30.314 4	3.069 9	0.281 5	Si
mt20	1165	1258	1 339.233 3	47.160 5	3.124 3	0.283 1	Si
orb1	1059	1135	1 219.500 0	38.282 1	3.079 7	0.280 9	Si
orb2	888	911	968.333 3	29.628 1	3.125 9	0.264 2	No
orb3	1005	1106	1 181.833 3	40.321 3	3.083 0	0.273 0	Si
orb4	1005	1041	1 105.933 3	36.719 6	3.128 1	0.280 3	Si
orb5	887	942	1 034.233 3	42.448 9	3.133 9	0.282 2	Si

TABLE VII

$P_C = 0.8, P_M = 0.1, \#Individuos = 200$

(*) $\sigma = 0$ No definido DA, se asume Normalidad.

REFERENCES

[1] Art. codificación binaria para jssp. https://github.com/leokraken/ae/blob/master/docs/articles/CHCbinario.pdf?raw=true.
[2] Art. representación directa cromosoma. https://github.com/leokraken/ae/blob/master/docs/articles/AMCS_2004_14_1_11.pdf?raw=true.
[3] g++. http://gcc.gnu.org/.
[4] Malva. https://github.com/themalvaproject/malva/wiki.
[5] Octave. http://www.gnu.org/software/octave/.
[6] Resultados obtenidos en ejecución algoritmos. https://github.com/leokraken/ae/blob/master/docs/Casos.zip?raw=true.
[7] Soluciones a instancias jssp. https://github.com/leokraken/ae/blob/master/docs/articles/JobShopSolutions.pdf?raw=true.
[8] Bruns R. *Direct chromosome representation and advanced genetic operators for production scheduling.* 1993.