



**UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INDUSTRIAS**

**COMPARACIÓN DE METAHEURÍSTICAS PARA RESOLUCIÓN DE
GRANDES INSTANCIAS DE PROBLEMAS DE SECUENCIAMIENTO EN
GROUP SHOPS**

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INDUSTRIAL

**AUTOR
KLAUS CHRISTOPHER WIDEMANN MÜLLER**

**PROFESOR GUÍA
FRANCISCO YURASZECK**

**PROFESOR CORREFERENTE
VICTOR ALBORNOZ**

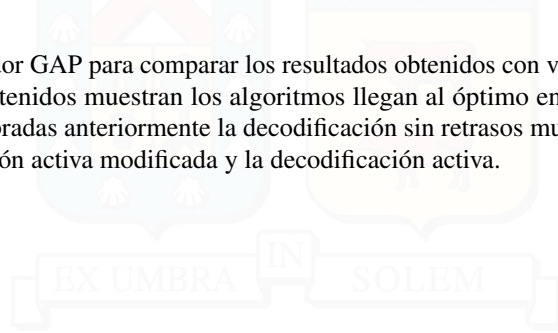
VALPARAÍSO, 30 DE OCTUBRE, 2020

RESUMEN EJECUTIVO

En la presente memoria se estudió e implementó dos metaheurísticas y tres decodificaciones para la resolución de instancias grandes de problemas de secuenciamiento en Group Shop.

Las metaheurísticas estudiadas corresponden al Simulated Annealing y al Algoritmo Genético. Ambas se diseñan en base a fenómenos naturales para resolver problemas a través de las decodificaciones. Estas fueron decodificación activa, decodificación activa modificada y decodificación sin retraso. Las decodificaciones permiten programar las soluciones bajo determinadas reglas a modo de obtener un tiempo de completación total.

Se utilizó el indicador GAP para comparar los resultados obtenidos con versiones realizadas por otros autores. Los resultados obtenidos muestran los algoritmos llegan al óptimo en problemas simples. De las tres decodificaciones nombradas anteriormente la decodificación sin retrasos muestra los mejores resultados seguida por la decodificación activa modificada y la decodificación activa.



Índice de Contenidos

1. PROBLEMA DE INVESTIGACIÓN	1
2. OBJETIVOS	2
2.1. Objetivo General	2
2.2. Objetivos específicos	2
3. MARCO TEÓRICO	3
3.1. Antecedentes Secuenciamiento en Investigación de Operaciones	3
3.2. Problema Group Shop	3
3.3. Simulated Annealing	4
3.4. Algoritmo Genético	5
3.5. Metodología	6
3.5.1. Equipo y software utilizado	6
3.5.2. Instancias utilizadas	6
3.5.3. Técnicas para análisis de datos	6
4. DESARROLLO	7
4.1. Codificación Group Shop Scheduling	7
4.2. Decodificaciones	10
4.2.1. Decodificación activa	10
4.2.2. Decodificación activa modificada	10
4.2.3. Decodificación non-delay	11
4.3. Código Algoritmo Genético	12
4.4. Código Simulated Annealing	13
5. EVALUACIÓN Y CONCLUSIONES	15
5.1. Open Shop y Job Shop	15
5.1.1. Open Shop $n=3, m=3$	15
5.1.2. Open Shop $n=4, m=4$	16
5.1.3. Open Shop $n=7, m=7$	16
5.1.4. Job Shop $n=10, m=10$	17
5.2. Algoritmo Genético	17
5.2.1. Élités y Tamaño población	17
5.2.2. Ratio mutación y tamaño población	18
5.2.3. Reglas de despacho	18
5.3. Simulated Annealing	19
5.3.1. Temperatura inicial	19
5.4. Group Shop	20
5.4.1. Comparación	20
5.4.2. Propuesta estudio de GSS SDST	20
5.5. Resumen y Conclusiones	21

REFERENCIAS	22
6. ANEXOS	24



Índice de Tablas

5.1. Instancias OSS gp03	15
5.2. GAP's promedio Instancias OSS gp04	16
5.3. GAP's promedio Instancias OSS gp07	16
5.4. GAP's promedio 5 Instancias JSS	17
5.5. Sensibilización de elitismo y tamaño de población	17
5.6. Promedios de resultados en sensibilizar población y mutación	18
5.7. GA sin reglas de despacho v/s con reglas de despacho	19
5.8. Resultados instancias ft10	20
5.9. ft10(1-10) Group Shop con Tiempos de Setup	21

Índice de Figuras

4.1. Representación trabajos	7
4.2. Representación tiempos proceso	8
4.3. Representación tiempos setup	8
4.4. Representación solución en lista	9
4.5. Representación solución en carta Gantt	9
4.6. Representación Cruza	13
5.1. Impacto mutación algoritmo genético	18
5.2. Impacto temperatura inicial	19
6.1. Tiempos setup carta Gantt ft10_05	24
6.2. Tiempos setup carta Gantt ft10_04	25

1. PROBLEMA DE INVESTIGACIÓN

Durante las últimas décadas se han investigado exhaustivamente el secuenciamiento de problemas en diversas áreas, como por ejemplo la industria aeronáutica (Dorndorf et al. (2007)) o automotriz (Andresen et al. (2012)). En el área de investigación de operaciones se ha hecho énfasis en la programación y secuenciamientos de problemas de taller, los cuales, si son relativamente pequeños, pueden expresarse como modelos lineales de programación entera en torno a la cantidad de máquinas y trabajos en el taller. De esta manera se puede obtener un resultado óptimo. Cuando la cantidad de máquinas y trabajos son menores a 10, se espera conseguir el resultado óptimo en algunos minutos. Sin embargo, cuando aumentan las dimensiones del problema, como por ejemplo operando con 20 máquinas y 10 trabajos, los tiempo de procesamiento computacional crecen exponencialmente en función de las más de $7,2651 \times 10^{183}$ soluciones posibles (Jain y Meeran (1999)). En la práctica, un encargado de secuenciamiento querrá tener un programa de acción dentro algunos segundos, para, a continuación, ponerlo en práctica. De esto se desprende que para problemas grandes como los NP-Hard (del inglés *non-deterministic polynomial-time hardness*), los procedimientos teóricos exactos, como el branch and bound o programación dinámica, no son aplicables dados sus grandes tiempos de resolución computacional (Pinedo y Hadavi (2012)).

Actualmente se reconocen tres sistemas principales de producción: el *Flow Shop Scheduling* (FSS) *problem*, *Job Shop Scheduling* (JSS) *problem* y *Open Shop Scheduling* (OSS) *problem*. En el caso del FSS, las operaciones realizadas tienen que ser procesadas en las máquinas en un mismo orden. El JSS es una instancia más general del FSS, en la cual las operaciones de cada trabajo deben ser procesadas en un orden estricto en las máquinas, no obstante diferentes trabajos pueden tener diferentes órdenes. En el OSS el orden de procesamiento de los trabajos en las máquinas queda irrestricto, siendo este último más general. Estos tres sistemas han sido estudiados desde 1950 y son conocidos por ser NP-Hard para resoluciones óptimas.

Adicionalmente existe un sistema de producción similar a los FSS, JSS y OSS. Este es el sistema de *Group Shop Scheduling* (GSS) *problem*. El término nace a partir de una competencia matemática de los Países Bajos Hurkens (1997). En un GSS las operaciones pertenecen a grupos de operaciones y no tienen restricciones mientras se respeten las restricciones de precedencia de sus grupos. En un caso extremo, si cada grupo tiene una sola operación, el GSS se transforma en un JSS y en el otro caso extremo si la cantidad de grupos es igual a 1, el GSS se transforma en un OSS.

Para resolver problemas de secuenciamiento de gran magnitud, se suele optar por meta-heurísticas que entreguen a resultados cercanos al óptimo en cuestión de segundos. Dentro de estos, Yu et al. (2010) abordan el problema de taller abierto con *Simulated Annealing* (SA), mediante el cual, simulando pérdida de temperatura en un metal, logra acercarse mucho a resultados óptimos sin caer en máximos o mínimos locales. De forma análoga Ahmadizar y Shahmaleki (2014) ocupa el algoritmo genético, el cual se basa en la teoría de la evolución darwiniana, para atacar un problema de secuenciamiento de GSS.

Cada una de estas herramientas busca encontrar una solución casi óptima en tiempos cortos. No obstante surgen las siguientes preguntas: ¿Cuál de las meta-heurísticas anteriormente mencionadas es más veloz para resolver problemas de GSS? ¿Cuál es la herramienta meta-heurística más adecuada para aplicar en la práctica?

Para resolver estas preguntas se pretende modelar un problema de GSS y resolverlo en varias instancias con cada una de las meta-heurísticas propuestas, midiendo los tiempos de resolución, resultados y estadísticas que permitan saber con qué confianza se puede tener un buen resultado con dicho modelo.

2. OBJETIVOS

2.1. Objetivo General

Evaluar distintas herramientas de optimización, a través de técnicas de investigación de operaciones, con el objetivo de determinar cuál de estas es más adecuada para resolver instancias grandes de problemas de secuenciamiento en *Group Shops*.

2.2. Objetivos específicos

Construir y formular el modelo de *Group Shops* para cada una de las meta-heurísticas a trabajar y así disponer de las herramientas a evaluar. Emplear las meta-heurísticas sobre el modelo de *Group Shops* ya formulado, con el fin de obtener los resultados del modelo y sus respectivos tiempos de resolución, en varias instancias, para tener información estadística sobre estas herramientas.

Comparar los resultados y tiempos de resolución de las herramientas meta-heurísticas empleadas en el modelo de *Group Shops*, con el propósito de juzgar sobre la base de esto, cuál de estas herramientas es más adecuada para resolver problemas de secuenciamiento en *Group Shops* en la práctica.

3. MARCO TEÓRICO

3.1. Antecedentes Secuenciamiento en Investigación de Operaciones

En las últimas décadas se ha realizado una vasta investigación en distintas ramas de secuenciamiento para desarrollo de investigación de operaciones, ingeniería industrial y computación. Esto ha dado paso a variadas técnicas para la resolución de problemas determinísticos de secuenciamiento desde la década de 1960 ([Pinedo y Hadavi \(2012\)](#)).

En el marco de los sistemas de producción más importantes se encuentran tres tipos principales de problemas de secuenciamiento: el *Flow Shop Scheduling* (FSS) *problem*, *Job Shop Scheduling* (JSS) *problem* y *Open Shop Scheduling* (OSS) *problem*. Estos problemas han sido investigados desde hace más de 50 años como por ejemplo, el trabajo de [Crowston et al. \(1963\)](#) en el cual aborda el estudio del JSS en 1963. En otros estudios se conocen los algoritmos de Johnson para para FSS y Jackson que generaliza el modelo de Johnson ([Johnson \(1959\)](#), [Hall y Shmoys \(2016\)](#)) que existen desde 1954 y han sido un fundamento para los estudios de investigación de operaciones en secuenciamiento. A pesar de que estos mismos trabajaron en sus modelos en años similares, no está claro quién fue el primero en estudiar este tipo de problemas, no obstante, se ha llegado al consenso de que el libro *Industrial Scheduling* ([Muth y Thompson \(1963\)](#)) editado por Muth y Thomson reúne los mayores descubrimientos en el área y ha sido una base para investigaciones futuras ([Jain y Meeran \(1999\)](#)).

El problema de secuenciamiento Group Shop (GSP) nace en los Países Bajos en una competencia donde participaron 343 profesionales en Eindhoven ([Hurkens \(1997\)](#)). La instancia fue resuelta en 469 minutos, cuyo mejor resultado alcanzó 5241 minutos, el cual fue comparado con 65 horas de branch and bound logrando la misma cota inferior. Con el paso del tiempo este problema ha llamado la atención de varios investigadores, como por ejemplo, [Sampels et al. \(2002\)](#) quien compara diferentes meta-heurísticas para resolver el GSP y [Ahmadizar y Shahmaleki \(2014\)](#), además incluye tiempos de set-up dependientes y duración de transporte. [Liu \(2005\)](#) aborda el GSP a través de un algoritmo de búsqueda tabú.

En cuanto al desarrollo del algoritmo genético (GA), este tiene registros ya en 1957 por [Fraser \(1957\)](#), quien implementa los principios de genética y selección natural como método para resolver problemas simples. En 1999 ([Ghedjati \(1999\)](#)) ocupa GA para resolver problemas de secuenciamiento con máquinas en paralelo, con rutas de proceso lineales y no lineales. ([Jia et al. \(2002\)](#)) implementa un algoritmo genético modificado que resuelve problemas de secuenciamiento y de distribución.

Los registros de la primera aplicación del algoritmo *Simulated Annealing* datan de 1988, año en el que [van Laarhoven y Aarts \(1987\)](#) ocupa SA para resolver problemas de *Job Shop* y este a su vez ha sido útil para numerosas investigaciones futuras ([Jain y Meeran \(1999\)](#)).

3.2. Problema Group Shop

Este modelo es una generalización de los problemas *Job Shop* y *Open Shop*. En este modelo se tienen m máquinas y n trabajos, donde cada trabajo posee una serie de operaciones que deben ser realizadas en máquinas específicas sin interrupción. Las operaciones de cada trabajo están divididas en grupos, los cuales

poseen un orden de precedencia para sus operaciones (Sampels et al. (2002)). El objetivo del problema es minimizar el tiempo en el que todas las operaciones son ejecutadas por sus respectivas máquinas, considerando para esto que cada máquina puede procesar como máximo una operación a la vez.

Blum y Sampels (2004) plantea este modelo de la siguiente manera: Tenemos un conjunto de operaciones $O = \{o_1, \dots, o_n\}$ donde cada una de estas posee un tiempo de procesamiento $p(o_i) \geq 0$. Este conjunto O se subdivide en los subsets $M = \{M_1, \dots, M_{|M|}\}$ que representan los conjuntos de operaciones que se operan en la misma máquina y al mismo tiempo por simplicidad llamaremos a las máquinas M_i donde i es el subíndice que representa al conjunto de operaciones que requieren la misma máquina. Paralelamente tendremos un conjunto de trabajos $J = \{J_1, \dots, J_{|J|}\}$ el cual es una división del conjunto de operaciones. Este último se subdivide en un subconjunto de grupos $G = \{G_1, \dots, G_{|G|}\}$, en el cual cada operación en un mismo grupo pertenece a un mismo trabajo. Esto se debe a que es un subconjunto de los trabajos. Adicionalmente se tiene como restricción que, en cada trabajo, los grupos están linealmente ordenados. Es decir, si tenemos un grupo G_l que venga antes de un grupo G_k , denotaremos $G_l \leq G_k$. Luego para cada operación $o \in G_l$ y para cada $o' \in G_k$, tendremos que $o \leq o'$. En consecuencia la operación o debe finalizar antes de que empiece la operación o' . Generalizando un poco esta idea, se llega a la conclusión de que el set de operaciones predecesoras para cualquier operación $o \in O$ esta dado por $pred(o) = \{o' \in O | o' \leq o\}$. Definiremos un predecesor directo con el operador $\leq_d$, entonces si tenemos $o, o' \in O$ tal que $o' \leq_d o$, no existe una operación o'' que cumpla con $o' \leq o'' \leq o$.

En vista de los puntos previamente explicados, se concluye que cada operación $o \in O$ tiene que ser procesada por una máquina $m(o) \in M$, pertenece a un trabajo $j(o) \in J$ y a un grupo $g(o) \in G$. En el marco de este trabajo de investigación se trabajará con el supuesto de que una vez que una operación comienza, esta debe de finalizar, sin posibilidad de que esta se interrumpa para dar paso a una operación diferente.

3.3. Simulated Annealing

Simulated Annealing (SA) es un algoritmo que busca un resultado óptimo local para problemas de optimización combinatorial (Liaw (1999)). Esto lo hace permitiendo soluciones peores, pero cada vez menos, en cada iteración. Este algoritmo nace a partir de la analogía del templado de metales, donde un sólido es calentado hasta tal punto, que sus partículas se pueden reordenar aleatoriamente en estado líquido para luego ser enfriado lentamente. De esta manera las partículas del metal pueden ubicarse en el nivel de energía correspondiente a la temperatura del metal, logrando así, siempre y cuando el proceso de enfriamiento sea suficientemente lento con la temperatura adecuada para cada nivel (van Laarhoven y Aarts (1987)), formando cristales. En caso de que la temperatura del metal baje muy rápido se obtienen resultados de cristalización deficientes (Yu et al. (2010)).

Si consideramos un problema combinatorial $\min\{C(s) | s \in S\}$ donde C es la función objetivo y S es el conjunto de todas las posibles soluciones, cada solución s tiene asociado un set de soluciones vecinas $N(s) \subset S$ asociadas, a las que llamaremos vecindad. Cada solución $s^* \in N(s)$ puede ser alcanzada directamente desde s a través de un operador de movimiento o de perturbación (Liaw (1999)). Definiremos un parámetro de control T que llamaremos temperatura y que será alto al comienzo del algoritmo.

El proceso comienza al obtener una solución aleatoria inicial para luego comenzar a revisar soluciones de la vecindad en la que se encuentre. De esta manera el algoritmo va tomando cualquier solución s cuyo valor de $C(s)$ sea mejor que el anterior. Para disminuir las posibilidades de caer en los óptimos locales, el algoritmo permite soluciones de su vecindad con peores funciones objetivo, bajo un criterio de aceptación que depende de la temperatura y de un valor aleatorio uniforme U . Este criterio es $U < e^{-\frac{\Delta C}{T}}$, donde $\Delta C = C(s^*) - C(s)$. Si una solución nueva posee una función objetivo peor que la anterior, se revisa el criterio. Si se cumple que el valor aleatorio es menor que el criterio de aceptación, se toma la nueva solución y se prosigue con la siguiente iteración. Mientras que la temperatura sea alta, la expresión $e^{-\frac{\Delta C}{T}}$ será muy cercana a 1 por lo es muy probable que U sea menor, aceptando así varias soluciones con peores funciones objetivos. Cuando la temperatura sea baja, la expresión tendrá valores cercanos a 0, por lo que no aceptará muchas soluciones peores. Posteriormente en cada iteración del código la temperatura deberá ir bajando para que eventualmente

solo acepte valores mejores.

3.4. Algoritmo Genético

El algoritmo genético es un procedimiento que se basa en los principios de la selección natural y la genética para encontrar de forma eficiente soluciones a ciertos problemas (Fraser (1957)). A partir de una población inicial, el algoritmo opera a los individuos para tener descendencia potencialmente mejores que los padres y tiende a eliminar a los individuos que no estén tan capacitados para sobrevivir. Cada operación de reproducción produce nuevas generaciones, las cuales se iteran hasta llegar a una solución aceptable o una determinada cantidad de veces (Pezzella et al. (2008))

Este algoritmo codifica las variables de decisión de un problema en un conjunto de caracteres que se llamarán cromosomas, donde cada carácter se denominará gen. Todos los cromosomas que existan simultáneamente se llamarán población. Con esto en mente, la forma en que se codifique cada cromosoma puede repercutir en la eficiencia y tiempos computacionales del algoritmo, puesto que se tiene que recurrir a él en múltiples instancias. Luego, el tamaño de la población puede jugar un rol fundamental, ya que una pequeña población puede hacer que el algoritmo caiga en óptimos locales fácilmente, mientras que una población excesivamente grande alarga los tiempos de procesamiento, perdiendo las ventajas que podría traer el algoritmo en sí (Sastry et al. (2005)). A modo de ejemplo, en un GSP, un cromosoma será una solución que represente una posible asignación de trabajos a las máquinas disponibles. En ese cromosoma cada gen representará a un trabajo ocupando una máquina en particular.

Para poder aplicar la lógica de la selección natural, primero se requiere de un método para la medición del desempeño de cada uno de los cromosomas en la población. Para eso se emplea una función objetivo, la cual puede ser un modelo matemático, simulación o una función subjetiva. De esta manera evaluamos a los individuos y guiamos al algoritmo a una mejor solución.

El algoritmo se basa en la siguiente estructura (Sastry et al. (2005)):

1. Representación de cromosomas: Es la manera en la que el problema se codifica en los cromosomas para que estos puedan ser evaluados por la función objetivo y poder ser discriminados de buenos y malos cromosomas. Cada cromosoma, definirá mediante sus genes una posible solución para el problema dado y la complejidad de esta representación puede tener consecuencias directas en los tiempos de resolución del código.
2. Inicialización: Los cromosomas candidatos iniciales suelen elegirse de forma aleatoria en función del tamaño deseado de la población, el cual conforma un parámetro arbitrario que puede ser sensibilizado para optimizar tiempos de resolución. Tener conocimiento del dominio de soluciones para el problema dado puede ser de utilidad para iniciar el procedimiento con una población de mejor rendimiento.
3. Evaluación: Para cada cromosoma de la población actual se calcula el rendimiento según la función objetivo en función de sus genes.
4. Selección: Esta etapa toma copias de la población actual con buenos rendimientos, siguiendo la idea de la supervivencia del más fuerte. El objetivo es preferir mejores cromosomas por sobre los peores, objetivo para el cual existen varios mecanismos como la selección por ruleta, estocástica universal, ranking o por torneo, entre otras. En algunas situaciones el permitir la selección de cromosomas con peor rendimiento puede causar que el algoritmo se salga de los óptimos locales. La cantidad de cromosomas seleccionados es un parámetro que puede ser alterado para cambiar los tiempos de resolución del algoritmo. Una selección demasiado grande o muy pequeña puede dificultar el funcionamiento del algoritmo para encontrar mejores soluciones y, por consecuencia, extender los tiempos de resolución.
5. Reproducción: Esta etapa combina genes de dos o más cromosomas seleccionados en la etapa anterior. Esta combinación es realizada hasta volver a tener una población del tamaño deseado, existiendo para

esto diversas maneras de llevar a cabo esta combinación, como por ejemplo selección aleatoria de genes de los cromosomas padres o *crossover* en permutación de lista (Poon y Carter (1995)).

6. Mutación: Mientras que la reproducción de cromosomas recombina genes de dos o más padres, la mutación es un cambio aleatorio en el gen de un cromosoma. Existen muchas formas de llevar a cabo la mutación y estas permiten soluciones en la vecindad de la población actual, logrando potencialmente encontrar nuevas y mejores soluciones.

3.5. Metodología

3.5.1. Equipo y software utilizado

Este proyecto se desarrolló y llevó a cabo en un computador Acer E5-573G-50G1. El equipo cuenta con un procesador Intel Core i5-5200U CPU @ 2.20GHz y una memoria RAM de 4 GB. Sistema operativo Windows 10 Pro. El código fué escrito en Python 3.7 a través del editor de texto JetBrains Pycharm Community Edition 2019.

3.5.2. Instancias utilizadas

Para comparar resultados de los algoritmos a desarrollar, se puede poner a prueba problemas ya evaluados por otros autores. Dado a que el problema de Group Shop incluye al Job Shop y al Open Shop como se explica en la sección 3.2, se probarán dichas instancias a modo de prueba. Se referirá a cada una de las versiones de un problema resuelto por otro autor como instancia. Las instancias evaluadas serán obtenidas de Blum (2020), Tamura (2020a)(JSS) y Tamura (2020b)(OSS) y corresponden a las siguientes:

- gp03(1-10)(OSS)
- gp04(1-10)(OSS)
- gp07(1-10)(OSS)
- abz5(JSS)
- abz6(JSS)
- ft10(JSS)
- orb08(JSS)
- orb09(JSS)
- ft10(1-10)(GSS)

3.5.3. Técnicas para análisis de datos

El análisis que se llevará a cabo será cuantitativo y, a través de la recolección de datos, permitirá evaluar los objetivos del proyecto. Esta recolección será de carácter secuencial, obteniendo datos un intento a la vez, programando estas iteraciones a través del mismo código. Los datos serán guardados en matrices de datos las cuales podrán ser analizadas y comparadas con datos de otros autores por medio del GAP.

$$GAP = \frac{\text{resultado} - \text{óptimo}}{\text{óptimo}} \quad (3.1)$$

4. DESARROLLO

4.1. Codificación Group Shop Scheduling

La representación de un problema de GSS de m máquinas y n trabajos es representada por una lista. Esta posee n elementos que representan cada uno de los trabajos. Estos a su vez, son listas que poseen cada una de las máquinas en las cuales deben ser procesadas las operaciones de dicho trabajo. Las máquinas de cada uno de los trabajos son agrupados a su vez en sus correspondientes grupos. A modo de ejemplo se puede ver en la figura 4.1 que el trabajo 1 posee dos grupos. El primero posee las operaciones del trabajo en la máquina 1 y en la máquina 2. El segundo grupo corresponde a la operación del trabajo 1 en la máquina 3, teniendo este grupo una restricción de precedencia del grupo 1. Esta representación permite tener información sobre los trabajos y máquinas de cada operación, visualizando los distintos grupos y sus restricciones.

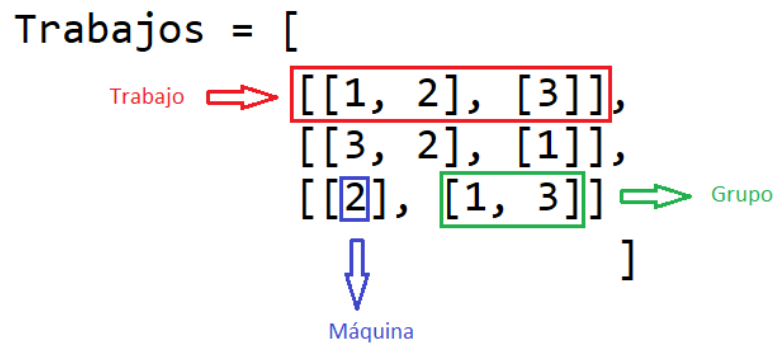


Figura 4.1: Representación de trabajos en GSS.
Fuente: Elaboración propia

Luego los tiempos de proceso de cada operación en su máquina y trabajo correspondiente está dado por una matriz representada por una lista de listas como se ve en la figura 4.2. Donde las M denotan máquinas y las J representan los trabajos. M0 se incorpora como una máquina de apoyo (*Dummy*) que permite generalizar el código sin alterar el resultado.

```

Tiempos_proceso = [
    J1 [0, 661, 6, 333],
    J2 [0, 168, 489, 343],
    J3 [0, 171, 505, 324]
]

```

M0 M1 M2 M3

Figura 4.2: Representación de tiempos de proceso en GSS.

Fuente: Elaboración propia

Para representar los tiempos de setup entre trabajo y trabajo para cada máquina, se tiene una matriz S_{ijk} como la que se ve en la figura 4.3, donde i es la máquina, y el trabajo j viene después de k. Se tienen máquinas y trabajos *dummy* que representan el tiempo de setup en una máquina que inicia o finaliza el proceso.

```

Sijk0 = [
    [[0, 2, 0, 0],
     [0, 0, 3, 1],
     [2, 1, 1, 0],
     [0, 2, 0, 0]],
    [[0, 1, 0, 0],
     [0, 1, 0, 3],
     [2, 0, 0, 2],
     [0, 1, 2, 0]],
    [[0, 2, 1, 0],
     [2, 0, 0, 1],
     [0, 1, 1, 0],
     [0, 1, 0, 0]],
    [[0, 0, 1, 0],
     [0, 2, 0, 1],
     [0, 0, 2, 0],
     [0, 1, 0, 3]]
]

```

 Máquina i
 Trabajo j
 Tiempo de setup
del trabajo j,
después del trabajo k,
en la máquina i

Figura 4.3: Representación de tiempos de setup en GSS.

Fuente: Elaboración propia

Cada una de las soluciones posibles del problema serán de la forma de una lista con parejas $[n, m]$, como se ve en la figura 4.4. Esta lista incluye todas las combinaciones requeridas para completar los requisitos de cada uno de los trabajos.

$[[1, 3], [2, 1], [2, 2], [1, 1], [2, 3], [3, 2], [3, 3], [1, 2], [3, 1]]$

Figura 4.4: Representación de una solución GSS.

Fuente: Elaboración propia

Las soluciones descritas por listas como están representadas en la figura 4.4, después de pasar por los algoritmos, pueden ser representadas en forma de operaciones en una carta gantt como se ve en la figura 4.5.

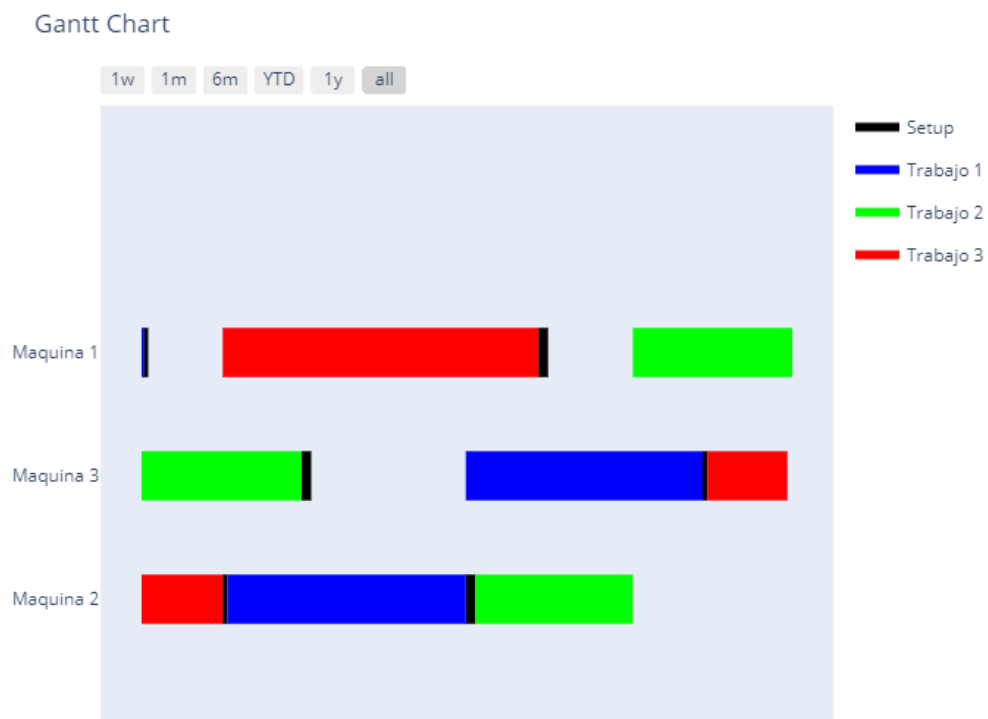


Figura 4.5: Representación de una solución GSS.

Fuente: Elaboración propia

$n=3, m=3$

4.2. Decodificaciones

Las diferentes decodificaciones corresponden a cómo se programa una solución para que esta entregue un resultado que pueda ser evaluado. En el contexto de este trabajo se utilizarán tres tipos de decodificaciones: Decodificación activa (Algoritmo 1), decodificación activa modificada (Algoritmo 2) y decodificación sin retraso(también llamada non-delay)(Algoritmo 3). Cada uno de los algoritmos nos entregará el tiempo máximo de finalización y un conjunto S_t , que representa las operaciones ordenadas de acuerdo a cada decodificación.

4.2.1. Decodificación activa

Aquella en la cual ninguna operación de un trabajo puede ser programada con anterioridad sin tener que reprogramar otras tareas. En esta decodificación se busca, dentro de las operaciones posibles, la máquina con menor tiempo de término. Posteriormente se selecciona una operación que tenga un inicio previo al tiempo de finalización para dicha máquina, como se puede ver en el algoritmo 1.

Algorithm 1 Decodificación Activa

```

1: procedure DECODIFICACIÓN_ACTIVA()
2:    $t = 1$ 
3:    $S_t = \emptyset$ 
4:    $NoS_t = \text{Operaciones\_por\_programar}()$ 
5:   while  $NoS_t \neq \emptyset$  do
6:      $C_t = \text{Operaciones\_posibles\_en\_t}(NoS_t)$ 
7:      $e_t = \text{tiempos\_inicio}(C_t)$ 
8:      $f_t = \text{tiempos\_término}(C_t)$ 
9:      $m_{t\_min} = \text{máquina\_con\_menor\_tiempo\_de\_término}(f_t)$ 
10:     $a_t = \text{tiempo\_término}(m_{t\_min})$ 
11:     $G_t = \text{operaciones\_en\_maquina}(m_{t\_min})$  tal\_que  $e_t \leq a_t$ 
12:    Seleccionar\_una\_operación\_de\_ $G_t()$ 
13:    Programar\_operación\_seleccionada()
14:    Eliminar\_operación\_seleccionada\_de\_ $NoS_t()$ 
15:    Agregar\_operación\_seleccionada\_a\_ $S_t()$ 
16:     $t = t + 1$ 
17:  end while
18:  Calcular\_tiempo\_completación\_total()
19: end procedure

```

4.2.2. Decodificación activa modificada

Es similar a la decodificación activa pero no se restringe a la máquina con menor tiempo de término. Esta decodificación busca en todas las operaciones que puedan iniciar en el programa, tal que su tiempo de inicio sea menor al menor tiempo de término. En este caso se busca el menor tiempo de término (a_t) de las operaciones posibles para programar y luego ver cuáles son las operaciones que inician antes que (a_t). Esto se ve explicado en detalle en el algoritmo 2.

Algorithm 2 Decodificación Activa Modificada

```

1: procedure DECODIFICACIÓN_ACTIVA_MODIFICADA()
2:    $t = 1$ 
3:    $S_t = \emptyset$ 
4:    $NoS_t = \text{Operaciones\_por\_programar}()$ 
5:   while  $NoS_t \neq \emptyset$  do
6:      $C_t = \text{Operaciones\_posibles\_en\_t}(NoS_t)$ 
7:      $e_t = \text{tiempos\_inicio\_}(C_t)$ 
8:      $f_t = \text{tiempos\_término\_}(C_t)$ 
9:      $a_t = \min(f_t)$ 
10:     $G_t = \text{operaciones\_tal\_que } e_t \leq a_t$ 
11:    Seleccionar_una_operación_de_ $G_t$ ()
12:    Programar_operación_seleccionada()
13:    Eliminar_operación_seleccionada_de_ $NoS_t$ ()
14:    Agregar_operación_seleccionada_a_ $S_t$ ()
15:     $t = t + 1$ 
16:  end while
17:  Calcular_tiempo_completación_total()
18: end procedure

```

4.2.3. Decodificación non-delay

En este programa ninguna máquina permanece inactiva cuando puede procesar algún trabajo. Se busca programar las operaciones que cuenten con los menores tiempos de inicio, dentro de las operaciones que sea posible programar.

Algorithm 3 Decodificación Non-Delay

```

1: procedure DECODIFICACIÓN_NON_DELAY()
2:    $t = 1$ 
3:    $S_t = \emptyset$ 
4:    $NoS_t = \text{Operaciones\_por\_programar}()$ 
5:   while  $NoS_t \neq \emptyset$  do
6:      $C_t = \text{Operaciones\_posibles\_en\_t}(NoS_t)$ 
7:      $e_t = \text{tiempos\_inicio\_}(C_t)$ 
8:      $\min\_e_t = \min(e_t)$ 
9:      $\text{index\_min\_}e_t = e_t.\text{index}(\min\_e_t)$ 
10:    Seleccionar_operación_ $C_t[\text{index\_min\_}e_t]$ 
11:    Programar_operación_seleccionada()
12:    Eliminar_operación_seleccionada_de_ $NoS_t$ ()
13:    Agregar_operación_seleccionada_a_ $S_t$ ()
14:     $t = t + 1$ 
15:  end while
16:  Calcular_tiempo_completación_total()
17: end procedure

```

4.3. Código Algoritmo Genético

Este procedimiento tiene por representación de cromosomas una lista de parejas, que especifican trabajo y máquina, tal como se muestra en la figura 4.4. El orden de estas parejas en conjunto con la decodificación utilizada determinará los distintos tiempos de completación total para cada uno de los individuos. Tomando en cuenta que se busca minimizar el tiempo de completación total, un mejor resultado será aquel que tenga un menor tiempo de completación.

La inicialización consiste en crear la población inicial en la línea 2 del algoritmo 4. Esto se lleva a cabo por selección aleatoria dentro de las combinaciones viables.

También se estudia la incorporación de heurísticas simples para generar la población inicial. Las heurísticas que se estudiaron fueron:

- Shortest Procesing Time (SPT): Consiste en ordenar las parejas de trabajos y máquinas en función de su tiempo de proceso de menor a mayor.
- Longest Procesing Time (LPT): Consiste en ordenar las parejas de trabajos y máquinas en función de su tiempo de proceso de mayor a menor.
- Most Work Remaining (MWR): Consiste en ordenar las parejas de trabajos y máquinas dando prioridad a los trabajos con mayor tiempo de trabajo pendiente.
- Least Work Remaining (LWR): Consiste en ordenar las parejas de trabajos y máquinas dando prioridad a los trabajos con menor tiempo de trabajo pendiente.
- Most Operations Remaining (MOR): Consiste en ordenar las parejas de trabajos y máquinas dando prioridad a los trabajos con mayor cantidad de operaciones pendientes.
- Least Operations Remaining (LOR): Consiste en ordenar las parejas de trabajos y máquinas dando prioridad a los trabajos con menor cantidad de operaciones pendientes.

La evaluación se realiza por medio de las decodificaciones ya mencionadas previamente en el capítulo 4.2 y la condición de término puede ser una cantidad fija de generaciones o una cantidad fija de tiempo de ejecución.

Algorithm 4 Algoritmo genético

```

1: procedure ALGORITMO_GENÉTICO()
2:   crear_población_inicial()
3:   mejor_tiempo = mejor_resultado_población_inicial()
4:   while condicion_término ≠ True do
5:     puntuar_soluciones()
6:     seleccionar_reproductores()
7:     cruzar_reproductores()
8:     mutar_población()
9:     tiempo_completacion = individuo_mejor_evaluado()
10:    if tiempo_completacion < mejor_tiempo then
11:      mejor_tiempo = Tiempo_completacion
12:      guardar_mejor_individuo()
13:    end if
14:  end while
15:  entregar_mejor_resultado()
16: end procedure

```

La selección de reproductores se realiza después de ser evaluados los individuos de la población. En función de como se evalúen los resultados de la población por selección de ruleta, se escogerán los posibles reproductores, donde los mejores resultados tendrán mayor posibilidad de ser elegidos. Sin embargo, para

obtener mejores resultados se incorporará una regla de elitismo que consiste en asegurar una pequeña cantidad de los mejores individuos.

La combinación de cromosomas se realiza tomando un tramo aleatorio de los genes de un padre y luego se toman en orden los genes de la madre que no coincidan con los que se tomaron del padre, como se puede ver en la figura 4.6.

En cuanto a la mutación, cada individuo tiene una pequeña posibilidad de intercambiar dos de sus genes ampliando la capacidad de exploración de nuevas soluciones del algoritmo.

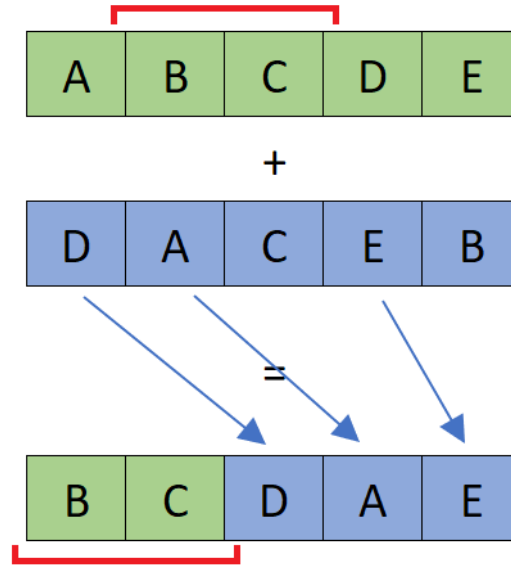


Figura 4.6: Representación de cruce de genes.
Fuente: Elaboración propia

4.4. Código Simulated Annealing

Este procedimiento tiene por representación de cada configuración de operaciones una lista de parejas, que especifican trabajo y máquina, como se puede ver en la figura 4.4. El orden de estas en conjunto con la decodificación utilizada determinará los distintos tiempos de completación total para cada uno de los casos evaluados.

El algoritmo inicia estableciendo los parámetros que ocupará más adelante y una solución inicial x_i . La variable x_F se encarga de almacenar los mejores resultados independiente de la capacidad del algoritmo de explorar resultados peores. El algoritmo itera hasta que se cumpla la condición de término, que en este caso puede ser una cantidad fija de iteraciones o una cantidad fija de tiempo.

La línea 11 del algoritmo 5 evalúa si el movimiento realizado es mejor. En caso positivo se toma la solución, en caso negativo se compara con un número aleatorio uniforme entre cero y uno con la ecuación 4.1, como se puede ver en la línea 14 del algoritmo 5.

$$e^{-\left(\frac{f(x_{temp}) - f(x_i)}{Temp_i}\right)} \quad (4.1)$$

La comparación permitirá tomar la decisión de aceptar o no la solución que, ya se sabe, era peor que la anterior, con el fin de explorar nuevas soluciones. Además restringirá la aceptación de soluciones peores en

la medida que las temperatura vayan disminuyendo y en la medida que la diferencia de las evaluaciones sea mayor.

Algorithm 5 Simulated Annealing

```

1: procedure SIMULATED_ANNEALING()
2:    $i = 1$ 
3:    $x_i = \text{crear\_solución\_inicial}()$ 
4:    $x_F = x_i$ 
5:    $Temp_i = \text{temperatura\_inicial}$ 
6:    $n = \text{movimientos\_por\_etapa}$ 
7:    $\alpha = \text{Factor enfiamiento}$ 
8:   while condición_término()  $\neq$  True do
9:     for  $\_$  in range( $n$ ) do
10:       $x_{temp} = \text{movimiento}(x)$ 
11:      if  $f(x_{temp}) \leq f(x)$  then
12:         $x_{i+1} = x_{temp}$ 
13:      else
14:        if  $U(0, 1) \leq e^{-\left(\frac{f(x_{temp}) - f(x_i)}{Temp_i}\right)}$  then
15:           $x_{i+1} = x_{temp}$ 
16:        else
17:           $x_{i+1} = x_i$ 
18:        end if
19:      end if
20:      if  $f(x_{i+1}) \leq f(x_F)$  then
21:         $x_F = x_{i+1}$ 
22:      end if
23:       $i = i + 1$ 
24:    end for
25:     $Temp_{i+1} = \alpha \cdot Temp_i$ 
26:  end while
27: end procedure

```

5. EVALUACIÓN Y CONCLUSIONES

5.1. Open Shop y Job Shop

En esta sección se estudiarán instancias de Open Shop y Job Shop en el algoritmo de Group Shop, definiendo la matriz de tiempos de setup como una matriz de ceros para hacer comparable los resultados obtenidos del algoritmo con los de otros autores que no ocupan tiempos de setup. Como ya se vió en la sección 3.2, el problema de Group Shop es una generalización de Open Shop y Job Shop, por lo que es un buen comienzo estudiar estos casos extremos para revisar si llega al óptimo.

El GAP permitirá tener una noción de que tan lejos se está en relación a la mejor solución encontrada por otros autores. Mientras más cercano a cero, mejor será el resultado. Su ecuación es la vista en la ecuación 3.1

El problema Open Shop se puede considerar como un Group Shop que tiene un solo grupo por cada trabajo, por otro lado el Job Shop se puede considerar como un Group Shop donde hay tantos grupos como máquinas, donde cada grupo posee una máquina.

5.1.1. Open Shop $n=3, m=3$

En el caso de Open Shop en instancias de 3×3 se obtienen óptimos consistentemente, como se puede ver en la tabla 5.1 que muestra los GAP iguales a 0 %. Estos resultados se tienen tanto para el Algoritmo Genético (GA) como para Simulated Annealing (SA), para las tres decodificaciones estudiadas (Decodificación activa, decodificación activa modificada y decodificación non delay). Los óptimos para las instancias gp03 fueron obtenidas de [Tamura \(2020b\)](#)

Instancia	GA_da	SA_da	GA_dam	SA_dam	GA_nd	SA_nd	Óptimo
gp03_1	0 %	0 %	0 %	0 %	0 %	0 %	0 %
gp03_2	0 %	0 %	0 %	0 %	0 %	0 %	0 %
gp03_3	0 %	0 %	0 %	0 %	0 %	0 %	0 %
gp03_4	0 %	0 %	0 %	0 %	0 %	0 %	0 %
gp03_5	0 %	0 %	0 %	0 %	0 %	0 %	0 %
gp03_6	0 %	0 %	0 %	0 %	0 %	0 %	0 %
gp03_7	0 %	0 %	0 %	0 %	0 %	0 %	0 %
gp03_8	0 %	0 %	0 %	0 %	0 %	0 %	0 %
gp03_9	0 %	0 %	0 %	0 %	0 %	0 %	0 %
gp03_10	0 %	0 %	0 %	0 %	0 %	0 %	0 %

Tabla 5.1: Instancias OSS gp03

$n=3, m=3$, tiempo límite = 5s, Algoritmo genético (GA) y Simulated Annealing (SA), Decodificación activa(da), Decodificación activa modificada(dam), Decodificación non delay(nd)

5.1.2. Open Shop $n=4, m=4$

Al estudiar el caso de OSS 4x4 gp04 se alcanzan óptimos pero ya no en todos los casos. En la tabla 5.2 se tienen los GAP promedio de los algoritmos con sus distintas decodificaciones. En este ejercicio ya se puede notar que los algoritmos se acercan bastante al óptimo alcanzando GAP menores al 1 % y que existen diferencias entre las decodificaciones. Los óptimos para las instancias gp04 fueron obtenidas de [Tamura \(2020b\)](#)

En este caso se puede apreciar que la decodificación activa tiene peores resultados en comparación a la decodificación activa modificada, sin embargo hay buenos resultados en general (GAP menores a 1 %).

Instancia	GA_da	SA_da	GA_dam	SA_dam	GA_nd	SA_nd	Óptimo
gp04_1	0,23 %	0,36 %	0,00 %	0,00 %	0,00 %	0,00 %	1281
gp04_2	0,10 %	0,13 %	0,00 %	0,00 %	0,08 %	0,08 %	1270
gp04_3	0,41 %	0,65 %	0,00 %	0,00 %	0,00 %	0,00 %	1288
gp04_4	0,05 %	0,08 %	0,00 %	0,00 %	0,16 %	0,16 %	1261
gp04_5	0,04 %	0,10 %	0,00 %	0,04 %	0,00 %	0,00 %	1289
gp04_6	1,06 %	1,94 %	0,00 %	0,00 %	0,00 %	0,00 %	1269
gp04_7	0,33 %	0,47 %	0,00 %	0,05 %	0,95 %	0,95 %	1267
gp04_8	0,77 %	2,12 %	0,11 %	0,28 %	0,00 %	0,00 %	1259
gp04_9	0,22 %	0,27 %	0,12 %	0,21 %	0,39 %	0,39 %	1280
gp04_10	0,63 %	1,20 %	0,00 %	0,00 %	0,00 %	0,00 %	1263

Tabla 5.2: GAP's promedio Instancias OSS gp04

Replicas = 6, $n=4$, $m=4$, tiempo límite = 10s, Algoritmo genético (GA) y Simulated Annealing (SA), Decodificación activa(da), Decodificación activa modificada(dam), Decodificación non delay(nd)

5.1.3. Open Shop $n=7, m=7$

En el intento de Open Shop de dimensión 7x7 se logra llegar al óptimo, pero se destaca que la decodificación sin retraso es la que logra los mejores resultados, seguida por la decodificación activa modificada. Este tipo de problema aún no nos permite obtener conclusiones sobre las diferencias entre ambos algoritmos, siendo ambos capaces de llegar a los óptimos dependiendo de la decodificación. Los óptimos para las instancias gp07 fueron obtenidas de [Tamura \(2020b\)](#)

Instancia	GA_da	SA_da	GA_dam	SA_dam	GA_nd	SA_nd	Óptimo
gp07_1	5,72 %	5,59 %	0,65 %	1,92 %	1,04 %	1,16 %	1159
gp07_2	3,12 %	2,87 %	0,06 %	0,89 %	0,00 %	0,00 %	1185
gp07_3	1,78 %	1,54 %	0,04 %	0,81 %	0,02 %	0,00 %	1237
gp07_4	3,02 %	2,98 %	0,28 %	0,66 %	0,09 %	0,19 %	1167
gp07_5	1,92 %	6,63 %	0,95 %	0,71 %	0,17 %	0,45 %	1157
gp07_6	1,49 %	3,14 %	0,17 %	2,58 %	0,34 %	0,34 %	1193
gp07_7	2,17 %	5,70 %	0,84 %	0,68 %	0,00 %	0,02 %	1185
gp07_8	1,44 %	2,06 %	0,59 %	0,47 %	0,08 %	0,08 %	1180
gp07_9	0,76 %	1,25 %	0,39 %	0,23 %	0,16 %	0,20 %	1220
gp07_10	0,30 %	0,65 %	0,00 %	0,00 %	0,00 %	0,00 %	1270

Tabla 5.3: GAP's promedio Instancias OSS gp07

Replicas = 6, $n=7$, $m=7$, tiempo límite = 180s, Algoritmo genético (GA) y Simulated Annealing (SA), Decodificación activa(da), Decodificación activa modificada(dam), Decodificación non delay(nd)

5.1.4. Job Shop n=10, m=10

En este caso se llegaron a resultados relativamente malos en comparación a los GAP obtenidos en los ejercicios de Open Shop, aunque el tamaño de esta instancia fué mayor a los tamaños evaluados en los Open Shop. A pesar de esto se puede rescatar que la decodificación sin retraso fue en todos los casos mejor a las demás decodificaciones y en todos los casos el algoritmo genético tuvo mejores resultados que el simulated annealing.

Instancia	GA_da	SA_da	GA_dam	SA_dam	GA_nd	SA_nd	Óptimo
abz5	39 %	38 %	30 %	33 %	25 %	27 %	1234
abz6	44 %	51 %	36 %	41 %	29 %	33 %	943
ft10	27 %	32 %	20 %	25 %	16 %	20 %	930
orb08	19 %	27 %	10 %	16 %	7 %	12 %	899
orb09	32 %	33 %	22 %	24 %	14 %	20 %	943

Tabla 5.4: GAP's promedio 5 Instancias JSS

Replicas = 6, n=10, m=10, tiempo límite = 180s, Algoritmo genético (GA) y Simulated Annealing (SA), Decodificación activa(da), Decodificación activa modificada(dam), Decodificación non delay(nd)

5.2. Algoritmo Genético

5.2.1. Élites y Tamaño población

El objetivo de este ejercicio es ver el impacto del tamaño de la población y el uso de élites en la selección de reproductores en el algoritmo genético. El ejercicio se realizó con la instancia ft10_05, con 200 generaciones Y con decodificación sin retraso. Se presenta el tiempo promedio de ejecución de las réplicas y su resultado promedio correspondiente. Todo esto con decodificación sin retraso, dado a sus mejores resultados en comparación a las otras decodificaciones.

Como se puede ver en la tabla 5.5, al incorporar 2 élites mejora la calidad de las soluciones, sin embargo al incorporar más, la calidad del resultado empieza a empeorar. El resultado mejora en la medida de que se tengan pocos élites, sin llegar a cero.

Por otro lado se puede visualizar que aumentar el tamaño de la población no tiene impacto en la calidad del resultado. Por el contrario, solo tiene impacto en el tiempo de ejecución, es por este motivo que de ser posible se deben elegir cantidades pequeñas de individuos en la población.

Num_ind	Elite 0		Elite 2		Elite 5		Elite 10	
	Tiempo	Res	Tiempo	Res	Tiempo	Res	Tiempo	Res
ind 20	80,5	920,2	80,1	887,5	81,7	889,75	80,9	882,75
ind 50	195,6	923,7	196,5	888	193,9321	886,5	193,1	876,25
ind 100	387,2	916,75	387,8	893,75	385,7326	882,5	386,8	874
ind 150	581,0	925,75	588,1	889,5	585,4868	886	582,2557	880
ind 200	774,654	922,25	793,6	887	778,8437	884	779,4869	886,5
Promedio		921,75		889,15		885,75		879,9

Tabla 5.5: Sensibilización de elitismo y tamaño de población

Instancia de prueba ft10_05. Tiempo de ejecución y resultado de cada réplica

5.2.2. Ratio mutación y tamaño población

En este experimento se busca sensibilizar el factor de mutación y el tamaño de la población. El ejercicio se realiza con la instancia *ft10_05* con un tiempo límite de 180 segundos. En teoría un ratio de mutación muy grande acerca al algoritmo a un comportamiento aleatorio y un ratio de mutación muy bajo puede lograr que el algoritmo caiga en óptimos locales evitando así resultados óptimos.

Como se puede ver en la tabla 5.6 al igual que en la tabla 5.5, el aumento de la población no parece tener impacto en la calidad de las soluciones. Por otro lado ratios muy pequeños de mutación empeoran las soluciones alcanzadas. Adicionalmente vemos en la tabla 5.6 que se tuvo mejores resultados para un factor de mutación igual a 0.1 %.

Num_individuos	mut 0,001	mut 0,0005	mut 0,0001	mut 0,00005	mut 0,00001
ind 10	884,75	913,75	912	907	917,25
ind 20	886	916,75	911,5	917,5	911,5
ind 30	880,25	923,25	917,75	915,25	913,5
ind 40	885,75	920	916,25	926,75	922,5
ind 50	887,5	916,5	920,75	923	924
ind 60	877	931	928,5	926,5	927,25
ind 70	888,5	920	924,25	917,5	925,25
ind 80	890,25	924	926,75	923	920,5

Tabla 5.6: Promedios de resultados en sensibilizar población y mutación
Instancia ft10_05, tiempo límite = 180s, élites = 2, decodificación sin retraso

Para tener una idea de como afecta la mutación a la búsqueda de soluciones, en la figura 5.1 se puede ver la diferencia en el progreso de la búsqueda de una mejor solución. La mutación de vez en cuando empeora la solución y es eso lo que le permite explorar y encontrar soluciones mejores. Se destaca que el algoritmo entrega la mejor solución que ha encontrado hasta que termina de ejecutarse.

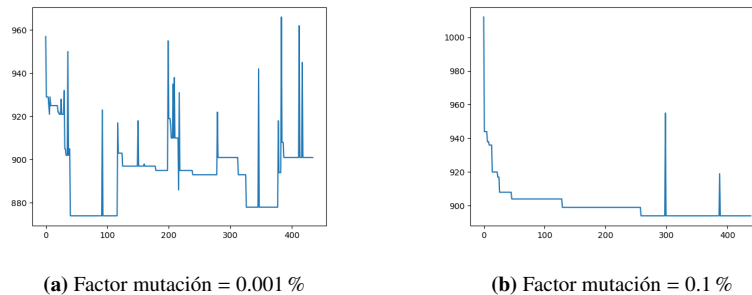


Figura 5.1: Impacto mutación algoritmo genético
ft10_5, población = 20, élites 2

5.2.3. Reglas de despacho

Para estudiar el impacto de las reglas de despacho se comparará un Group Shop con reglas de despacho incorporadas en su población inicial contra las mismas instancias pero con población inicial plenamente aleatoria. Los resultados en la tabla 5.7 muestran que los intentos con reglas de despacho en promedio, en la mayoría de los casos, obtienen mejores resultados en comparación con los intentos con poblaciones iniciales aleatorias. Esto también se ve reflejado en los mejores resultados obtenidos, donde los mejores resultados están en las versiones con regla de despacho.

Instancias	Avrg no regla	Avrg regla	Best no regla	Best regla
ft10_1	1083,5	1080,75	1067	1073
ft10_2	1051,75	1048,5	1029	1028
ft10_3	996,5	981,625	979	967
ft10_4	961,25	951	947	944
ft10_5	878,625	876,875	872	861
ft10_6	865,5	865,25	857	849
ft10_7	874,875	868,375	868	850
ft10_8	870,5	864,75	866	862
ft10_9	770,5	770,75	763	765
ft10_10	655	655	655	655

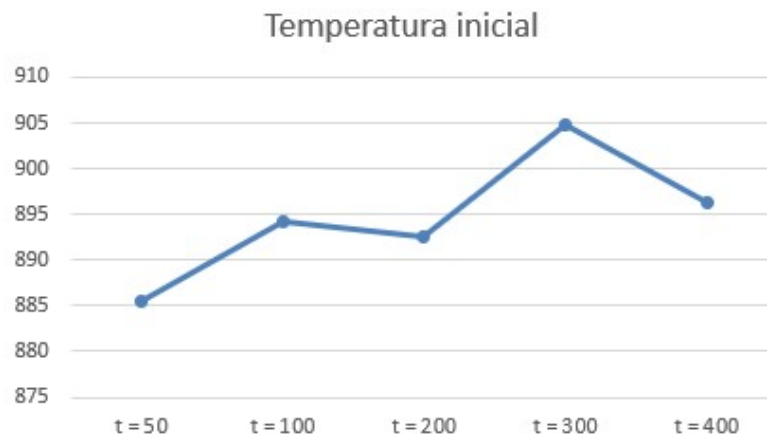
Tabla 5.7: GA sin reglas de despacho v/s con reglas de despacho

Resultados promedio y mejor resultado, instancias = ft10 (1-10), Ratio mutación = 0.1 %, tiempo límite = 180, número individuos = 20, elites = 20

5.3. Simulated Annealing

5.3.1. Temperatura inicial

En el algoritmo Simulated Annealing, como ya se mencionó antes, la ecuación 4.1 es la que va decidiendo si se acepta o no una solución peor a la actual en función de que tan mala es esta nueva solución respecto a la actual y en función de la temperatura. El objetivo de la temperatura es restringir de a poco la aceptación de soluciones peores. En la figura 5.2 se presenta el resultado de las replicas de la instancia ft10_05 con un tiempo límite de ejecución de 180 segundos. Se aprecia que en la medida que baja la temperatura inicial, mejoran los resultados.

**Figura 5.2:** Tiempos de completación para temperaturas iniciales en SA

Fuente: Elaboración propia

Promedios de 4 réplicas, 5 movimientos por temperatura, ft10_05, factor enfriamiento = 0.985

5.4. Group Shop

5.4.1. Comparación

En este ejercicio se pusieron a prueba los algoritmos con las instancias ft10 con 10 máquinas y 10 trabajos. Estas instancias comparten tiempos de proceso y tienen distinta disposición de grupos. Donde la instancia ft10_1 tiene 10 grupos de una sola operación, correspondiendo a un Job Shop, y gradualmente los grupos de cada instancia van incorporando más operaciones, hasta ft10_10 que tiene un solo grupo con las 10 operaciones del trabajo, que corresponde a un Open Shop.

Lo primero que destaca la vista es que se llega al óptimo en la instancia ft10_10 para ambos algoritmos de manera consistente. Sin embargo el resto de los resultados no es tan bueno como se preferiría para el resto de las instancias. Se puede notar que los resultados entre ambos algoritmos son relativamente parecidos.

Instancias	Simulated Annealing			Algoritmo Genético			Best Known
	Avrg	Best	Gap	Avrg	Best	Gap	
ft10_1	1092	1067	14,7 %	1080,75	1070	15,1 %	930
ft10_2	1079,75	1039	19,2 %	1058,375	1049	20,3 %	872
ft10_3	1014,25	981	18,6 %	998,75	990	19,7 %	827
ft10_4	969,375	953	21,9 %	967,875	947	21,1 %	782
ft10_5	888,125	868	16,5 %	886,125	870	16,8 %	745
ft10_6	873,125	857	18,2 %	878,75	873	20,4 %	725
ft10_7	881,125	871	27,3 %	876,25	869	27,0 %	684
ft10_8	872	860	31,3 %	875,125	871	33,0 %	655
ft10_9	778,375	768	17,3 %	773,25	766	16,9 %	655
ft10_10	655	655	0,0 %	655	655	0,0 %	655

Tabla 5.8: Resultados instancias ft10

Resultados promedio, mejor resultado y los Gap, para Simulated Annealing y Algoritmo Genético. 8 Repeticiones, decodificación non-delay, élites = 2, tamaño población = 20, ratio mutación = 0.1 %, temperatura inicial = 50, ratio enfriamiento= 0.985 %, movimientos por temperatura = 5

5.4.2. Propuesta estudio de GSS SDST

En la literatura existen problemas planteados de Group Shop, sin embargo estos no consideran los tiempos de setup dependientes entre máquinas. Para poder poner a prueba ambos algoritmos en un ejercicio de Group Shop en conjunto con los tiempos de setup se decide ocupar una versión modificada de los problemas ft10(1-10), similar a lo realizado por [Ahmadizar y Shahmaleki \(2014\)](#). Esta modificación consiste en definir una matriz de tiempos de setup a partir de valores aleatorios uniformes entre 0 y 9. Los resultados obtenidos pueden verse en la tabla 5.9.

Instancias	Simulated Annealing		Algoritmo Genético	
	Avrg	Best	Avr	Best
ft10_1	1194,4	1190	1190	1190
ft10_2	1184,1	1142	1174,1	1146
ft10_3	1101,1	1073	1078,3	1066
ft10_4	1056,4	1038	1039,4	1020
ft10_5	932,1	922	932,5	917
ft10_6	924,9	913	924,1	920
ft10_7	934,8	925	919,4	894
ft10_8	919,5	911	912,9	905
ft10_9	825,6	813	815	805
ft10_10	660,4	657	657,1	655

Tabla 5.9: ft10(1-10) Group Shop con Tiempos de Setup

Tiempo límite = 180, movimientos por temperatura = 5, temperatura inicial = 50, factor enfriamiento = 0.985, tamaño población = 20, ratio mutación = 0.1 %, elites = 2

5.5. Resumen y Conclusiones

Se presentaron dos algoritmos para la resolución de problemas de Group Shop Scheduling, un problema de complejidad NP-Hard ⁽¹⁾ y que incluye problemas como el Job Shop y el Open Shop. Uno de Algoritmo Genético y otro de Simulated Annealing. También se estudiaron tres diferentes formas de codificar soluciones: Decodificación activa, activa modificada y sin retraso. Para probar la utilidad de los algoritmos se realizaron varios experimentos con instancias pequeñas, en distintos espectros del Group Shop, las cuales mostraron la capacidad de los algoritmos para llegar a resultados óptimos en problemas simples. Se logra observar que entre las tres decodificaciones la que muestra mejores resultados es la decodificación sin retraso, seguida por la decodificación activa modificada y en último lugar la decodificación activa.

Se presenta en forma de pseudo-código la manera en la que se programó cada uno de los algoritmos y decodificaciones, para poder explicar con detalle al lector como se abordó la problemática.

En este informe se propone una variación del problema ft10 con una matriz de tiempos de setup generada con valores aleatorios uniformes entre 0 y 9, en la cual se obtienen, en general, mejores resultados para el algoritmo genético, estando relativamente cerca los mejores resultados obtenidos de ambos algoritmos.

A partir de este problema se desprenden muchas posibilidades para futuras investigaciones, como por ejemplo estudiar otro tipo de decodificaciones y experimentar más en instancias de grandes dimensiones. Eventualmente se podría buscar una industria en la cual se puedan recolectar datos y aplicar estos algoritmos para programar operaciones en contextos reales.

¹Del inglés *non-deterministic polynomial-time hardness*

REFERENCIAS

- Ahmadizar, Fardin y Shahmaleki, Parmis (2014). Group-shop scheduling with sequence-dependent set-up and transportation times. *Applied Mathematical Modelling*, 38(21-22), 5080–5091.
- Andresen, Michael; Brsel, Heidemarie; Plauschin, Mathias; y Werner, Frank (2012). Using Simulated Annealing for Open Shop Scheduling with Sum Criteria. *Simulated Annealing*.
- Blum, Christian (2020). Christian Blum Instances.
- Blum, Christian y Sampels, Michael (2004). An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms*, 3(3), 285–308.
- Crowston, Wallace B.; Glover, Fred; Thompson, Gerald L.; y Trawick, Jack D. (1963). Probabilistic And Parametric Learning Combinations of Local Job Shop Scheduling Rules. *Carregie Institute of Technology Pittsburg Graduate School of Industrial Administration*, (O.N.R. Research Memorandum No. 117).
- Dorndorf, Ulrich; Drexl, Andreas; Nikulin, Yury; y Pesch, Erwin (2007). Flight gate scheduling: State-of-the-art and recent developments. *Omega*, 35(3), 326–334.
- Fraser, AS (1957). Simulation of Genetic Systems by Automatic Digital Computers. *Australian Journal of Biological Sciences*.
- Ghedjati, Fatima (1999). Genetic algorithms for the job shop scheduling problem with unrelated parallel constraints. *Search*, 37, 39–42.
- Hall, Leslie A. y Shmoys, David B. (2016). Jackson's Rule for Single-Machine Scheduling: Making a Good Heuristic Better. *Mathematics of Operations Research*, 17(1), 22–35.
- Hurkens, Cor (1997). Homepage WHIZZKIDS 97.
- Jain, A S y Meeran, S (1999). Deterministic job-shop scheduling : Past , present and future. 113, 390–434.
- Jia, H. Z.; Nee, A. Y.C.; Fuh, J. Y.H.; y Zhang, Y. F. (2002). A modified genetic algorithm for distributed scheduling problems. *Journal of Intelligent Manufacturing*, 14(3-4), 351–362.
- Johnson, S.M. (1959). Discussion: Sequencing n Jobs on Two Machines with Arbitrary Time Lags. *Management Science*, 5(3), 299–303.
- Liaw, Ching Fang (1999). Applying simulated annealing to the open shop scheduling problem. *IIE Transactions (Institute of Industrial Engineers)*, 31(5), 457–465.
- Liu, S.Q. (2005). A fast tabu search algorithm for the group shop scheduling problem. *Advances in Engineering Software*, 36(8), 533–539.
- Muth, J.F. y Thompson, G.L. (1963). *Industrial Scheduling*. Prentice-Hall International series in management. Prentice-Hall.
- Pezzella, F.; Morganti, G.; y Ciaschetti, G. (2008). A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers and Operations Research*, 35(10), 3202–3212.

- Pinedo, Michael y Hadavi, Khosrow (2012). Scheduling: Theory, Algorithms and Systems Development. In *Operations Research Proceedings 1991* (pp. 35–42). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Poon, P.W. y Carter, J.N. (1995). Genetic Algorithm Crossover Operators for Ordering Applications. *Matrix*, 22(1), 135–147.
- Sampels, Michael; Blum, Christian; Mastrolilli, Monaldo; y Rossi-Doria, Olivia (2002). Metaheuristics for Group Shop Scheduling. (pp. 631–640).
- Sastry, Kumara; Goldberg, David; y Kendall, Graham (2005). Chapter 4 Genetic Algorithms. *Search Methodologies*, (pp. 97–125).
- Tamura, Naoyuki (2020a). CSP2SAT: JSS benchmark results.
- Tamura, Naoyuki (2020b). CSP2SAT: OSS benchmark results.
- van Laarhoven, Peter J.M. y Aarts, Emile H.L. (1987). Chapter 2 Simulated annealing 2.1 Introduction of the algorithm. *Simulated Annealing: Theory and Application*, (pp.7).
- Yu, Vincent F.; Lin, Shih Wei; y Chou, Shuo Yan (2010). The museum visitor routing problem. *Applied Mathematics and Computation*, 216(3), 719–729.

6. ANEXOS



Figura 6.1: Carta Gantt

Fuente: Elaboración propia

Tiempo = 180, población = 20, mut = 0.001 %, elite = 2, ft10_05

**Figura 6.2:** Carta Gantt

Fuente: Elaboración propia

Tiempo = 180, población = 20, mut = 0.001 %, elite = 2, ft10_04