PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

UN ALGORITMO GRASP CON DOBLE RELAJACIÓN PARA RESOLVER EL PROBLEMA DEL FLOW SHOP SCHEDULING

TESIS PARA OPTAR EL TÍTULO DE INGENIERO INFORMÁTICO

PRESENTADO POR:

CÉSAR OSWALDO RAMÍREZ RODRÍGUEZ

LIMA – PERÚ

2006

El Señor cuida de los sencillos.
Cuando yo estaba sin fuerzas me salvó.
Amo al señor porque ha escuchado mis súplicas,
Porque me ha prestado atención.
¡Toda mi vida le invocaré!
¡El Señor llevará a feliz término su acción en mi favor!.
Señor, tu amor es eterno; ¡no dejes incompleto lo que has emprendido!

Salmo 116-138

AGRADECIMIENTOS

A Dios, que me acompañó a lo largo de mi vida universitaria, ayudándome en los momentos difíciles.

A mi madre por ser un vivo ejemplo del coraje, esfuerzo y perseverancia, quién prefirió a nosotros sus hijos que al mundo.

A mi Asesor Manuel Tupia, por su apoyo invalorable en conocimientos, consejos, paciencia y confianza en mi persona.

A mis hermanas, porque confiaron en mi desde niño, en alcanzar mi objetivo.

Al profesor Carlos Vera, por su ayuda incondicional en el material bibliográfico.

A Paola Guerra, Johnny Morales y Martín León por su valioso apoyo.

A David y Adriana, mis sobrinos, que serán el futuro de esta gran nación.

RESUMEN

La mayoría de líneas de producción -incluso grandes-, no tienen una forma adecuada de planificar su producción, optando por distribuciones manuales, producto del conocimiento del jefe de planta o repitiendo alguna anterior con realidades distintas. Esto conlleva a que los recursos (trabajadores, maquinas) estén sin trabajar (ociosas) hasta ser utilizada, manteniéndolas –las máquinas- prendidas consumiendo combustible y los trabajadores especialistas a una tarea, haciendo otra tarea.

El problema de la planificación industrial (scheduling) no tiene solución exacta (problema NP) aún para instancias pequeñas (3 máquinas). Frente a esta realidad las meta heurísticas es la mejor vía para encontrar buenas soluciones –que se acerquen a la solución exacta- en tiempos cortos.

La línea de producción mostrada en la presente tesis, es del tipo Flow shop scheduling (FSS), donde las tareas son independientes y las maquinas son homogéneas (toman aproximadamente el mismo tiempo para realizar una tarea), las maquinas se dividen en familia según la función que cumpla (El FSS forzará que las tareas pasen por todas las familias).

El presente trabajo de investigación presenta un algoritmo meta heurístico del tipo GRASP para optimizar líneas como las definidas arriba. Su novedad constituye en el doble criterio de relajación, tanto para máquinas y tareas.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

RESUMEN

La mayoría de líneas de producción -incluso grandes-, no tienen una forma adecuada de planificar su producción, optando por distribuciones manuales, producto del conocimiento del jefe de planta o repitiendo alguna anterior con realidades distintas. Esto conlleva a que los recursos (trabajadores, maquinas) estén sin trabajar (ociosas) hasta ser utilizada, manteniéndolas –las máquinas- prendidas consumiendo combustible y los trabajadores especialistas a una tarea, haciendo otra tarea.

El problema de la planificación industrial (scheduling) no tiene solución exacta (problema NP) aún para instancias pequeñas (3 máquinas). Frente a esta realidad las meta heurísticas es la mejor vía para encontrar buenas soluciones –que se acerquen a la solución exacta- en tiempos cortos.

La línea de producción mostrada en la presente tesis, es del tipo Flow shop scheduling (FSS), donde las tareas son independientes y las maquinas son homogéneas (toman aproximadamente el mismo tiempo para realizar una tarea), las maquinas se dividen en familia según la función que cumpla (El FSS forzará que las tareas pasen por todas las familias).

El presente trabajo de investigación presenta un algoritmo meta heurístico del tipo GRASP para optimizar líneas como las definidas arriba. Su novedad constituye en el doble criterio de relajación, tanto para máquinas y tareas.

CAPITULO 1

1 Introducción

El trabajo de tesis expuesto a continuación define su marco teórico dentro de los siguientes conceptos:

- Organización general de la tesis.
- Optimización combinatoria: definición y tipo de problemas que son objetos de su estudio.
- El problema de la planificación industrial como antecedente al problema Flow shop Scheduling.
- Conceptos generales sobre heurísticas y meta heurísticas: algoritmos voraces y algoritmos GRASP.

1.1 Organización general de la tesis

El presente trabajo de investigación se centra en solucionar el problema del Flow Shop Scheduling (de ahora en adelante FSSP), el cual es una variante de la programación de tareas muy vista en líneas de producción de fábricas manufactureras, con la variante de máquinas semejantes y tareas independientes.

Para resolver este problema –que pertenece a la familia NP difícil por el gran número de combinaciones que presenta– se implementará un algoritmo Voraz Ramdonómico (aleatorio) Adaptativo (GRASP) correspondiente a las meta heurísticas, es decir, a algoritmos aproximados que obtienen soluciones de un alto grado de calidad, aplicando técnicas inteligentes en tiempos relativamente menores.

Justificamos el uso del algoritmo mencionado por las siguientes razones:

1. En la vida real, las líneas de producción industrial siguen una distribución compuesta por máquinas y tareas. Las máquinas, generalmente agrupadas por funcionalidad –elaboración de una tarea en particular– varían en rendimiento debido a características técnicas, de ensamblaje, tiempo de uso, entre otras; pero a aquellas que demoran aproximadamente el mismo tiempo en ejecutar una tarea se las denomina *máquinas similares*. Las tareas, las mismas que guardan relaciones de dependencia una de la otra, requieren ser elaboradas por las máquinas mencionadas anteriormente, siéndoles posible escoger la mejor máquina dentro de las otras similares, tomando en cuenta criterios como dependencia entre tareas y tiempo de finalización de la tarea predecesora, para así coordinar con la otra y evitar tiempos muertos.

   Este modelo es uno de los que más se ajusta a la problemática industrial sobre todo por las características de máquinas y tareas, caracterizándose como problema cotidiano para los encargados de producción.

2. La elaboración de un método o algoritmo creado a medida, que contempla las distintas características mencionadas, conlleva a un análisis minucioso, extenso y computacionalmente complejo producto de todas las combinaciones existentes, las cuales se tendrían que analizar una por una; además de usar lenguajes de programación específicos para tales fines. Todo lo mencionado nos revela la profunda complejidad de dichas soluciones.
   Los algoritmos heurísticos y meta heurísticos (pertenecientes a los algoritmos aproximados) analizan las alternativas más representativas, y aplicando muestras representativas de los posibles valores de manera inteligente, consiguen resultados muy cercanos (comparados con los que hubieren sido obtenidos usando métodos exactos, pero además, con un costo computacional muy bajo y fácil de implementar en todos los lenguajes de programación de alto nivel).

El trabajo se divide en los siguientes capítulos:

- Capitulo 1 (Introducción): se presenta el marco teórico general, para entender la terminología y los métodos usados, así como la problemática entorno al problema.

- Capitulo 2 (El problema de la programación industrial): se define el problema y las características y métodos (exactos o aproximados) existentes para su solución.

- Capitulo 3 (Algoritmo propuestos): se desarrolla un algoritmo GRASP sobre el cual se despliega la propuesta de solución al problema planteado en la sección anterior; además de añadirse los comentarios respectivos y las estructuras de datos utilizadas.

- Capitulo 4 (Implementación y Experimentos numéricos): circunscribe la descripción del análisis realizado para el problema en mención, exponiendo el algoritmo propuesto y los respectivos comentarios. Además se explican las pruebas a las que fueron sometidos los algoritmos; se presentan los resultados contrastados del algoritmo VORAZ frente a los del GRASP propuesto para determinadas instancias de prueba.

- Capitulo 5 (Conclusiones y recomendaciones): conclusiones, producto del análisis del presente tema de tesis.

1.2 Optimización combinatoria

1.2.1 Introducción

En las últimas décadas, la comunidad científica ha asistido al nacimiento de la disciplina conocida como Ciencias de la Computación, la misma que siendo inicialmente una rama de la Matemática aplicada, halló su propio espacio de investigación, definiéndose posteriormente como una nueva área de la ciencia. Dicha disciplina experimentó un vertiginoso ascenso desde su nacimiento, siendo considerada en la actualidad como una de las áreas con mayor actividad y desarrollo.

Una de las ramas de mayor importancia y crecimiento dentro de las Ciencias de la Computación es el conjunto de actividades conocidas como ***Investigación Operativa***, que por su impacto y resultados concretos en la industria y otros ámbitos, se ha cimentado como uno de los pilares de esta nueva ciencia.

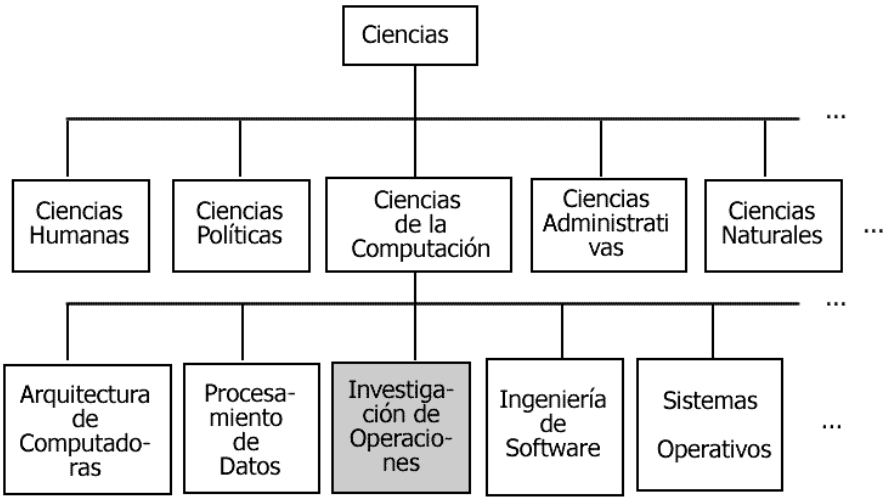


Figura 1 Ubicación de la actividad, Investigación de operaciones

Como su nombre lo indica la Investigación de Operaciones (IO) significa "investigar sobre las operaciones". Entonces, la IO se aplica a problemas que se refieren a la conducción y coordinación de operaciones dentro de una organización con el objetivo de encontrar una mejor solución (llamada solución óptima) para el problema bajo consideración, usando métodos matemáticos, estadísticos y algoritmos. Esta búsqueda del óptimo es un aspecto importante dentro de la IO.

El concepto *investigación* en el nombre revela que la IO utiliza un enfoque similar a la manera en que se lleva a cabo la investigación en campos científicos establecidos. En gran medida se aplica el método científico para investigar un problema en cuestión mediante el empleo de técnicas cuantitativas [20,22]

Dentro de las principales técnicas cuantitativas podemos nombrar las siguientes: [21]

a) Programación entera.
b) Programación matemática (lineal y no lineal).
c) Problemas de transporte.
d) Análisis de grafos y de redes. PERT y CPM.
e) Programación dinámica.
f) Teoría de juegos.
g) Teoría de colas.
h) Teoría de inventarios.
i) Procesos markovianos de decisión. Análisis de decisión.

j) Simulación.
k) Fiabilidad, entre otras.

Es importante mencionar que estas técnicas nacieron de la necesidad de resolver problemas en diversas áreas. Por ejemplo, la teoría de juegos nace de la necesidad de plantear una estrategia militar previa al enfrentamiento bélico y consistía en intuir el armamento militar, la ubicación y el número de soldados del enemigo, y en base a estos datos, se planeaba la táctica a ejecutar en el enfrentamiento. Hoy en día es también usada la simulación en el comportamiento de los clientes frente al lanzamiento comercial de un producto, entre otras funciones.

Para atacar el problema que enfrentaban a diario las líneas de despacho de bienes, se implementó la técnica que resuelve los problemas de transporte. Para agilizar y evitar la espera del cliente en los establecimientos, se creó la teoría de colas.

El problema de optimizar una función matemática para obtener su máximo o mínimo (que a su vez contiene variables) para las cuales existen restricciones sobre su rango, naturaleza, y demás, generó el inicio de la programación matemática.

La *programación matemática,* es una potente técnica de **modelado empleada** en el proceso de toma de decisiones. Cualquier problema de programación matemática requiere identificar cuatro componentes básicos [24]:

1. El conjunto de datos.
2. El conjunto de variables involucradas en el problema, junto con sus dominios respectivos de definición.
3. El conjunto de restricciones del problema que definen el conjunto de soluciones admisibles.
4. La función objetivo que debe ser optimizada (minimizada o maximizada).

En la *programación lineal*, que trata exclusivamente con funciones objetivo y restricciones lineales, las variables toman valores reales. Cuando los valores de las variables son enteras, o incluso están más restringidas siendo binarias, es decir que toman exclusivamente los valores O ó 1 se denomina *programación lineal entera-mixta.* Cuando el conjunto de restricciones, la función objetivo, o ambos, son no lineales, se dice que se trata de un problema de *programación no lineal.*

El problema de la *planificación industrial –el cual es materia de estudio en el presente trabajo de investigación – será* modelado como un problema de programación lineal por las variables de primer grado que se utilizan en su formulación; además, se empleará la Optimización combinatoria como técnica de solución debido a que persigue el optimizar un objetivo (para este caso en particular se minimizará el tiempo total que se demora en ejecutar todas las tareas relacionadas a la producción).

1.2.2 Definición

La *Optimización Combinatoria* es un área intrínsecamente relacionada a la Investigación Operativa que se encarga de buscar la mejor solución en problemas discretos (es decir, en los que participa una cantidad finita de elementos) mediante combinaciones y ordenamientos a grupos de elementos, con el fin de obtener patrones adecuados que resuelvan determinados problemas.

Los problemas pertenecientes a esta sección persiguen mejorar, optimizar o minimizar algún criterio (para el caso de una línea de producción, se trataría del consumo de tiempo y/o recursos, costo, minimización de desplazamientos, movimientos, etc.) relacionados a la obtención de un objetivo.

Un problema de optimización en su representación más general, puede ser modelado como un problema de programación lineal. Observar la siguiente Figura:

Optimizar $f(x)$
Sujeto a: $g_i(x) \geq 0,\ i = 1...m$
$h_j(x) \geq 0,\ j = 1...n$

Desde el planteamiento anterior tenemos que:

- $f(x)$ es conocida como *función objetivo* y representa un valor que debe ser optimizado (maximizado o minimizado).

Tanto $g_i(x) \geq 0, i = 1...m$ como $h_j(x) \geq 0,\ j = 1...n$ son denominadas las *restricciones del problema* y especifican las condiciones que debe poseer toda *solución viable* para el mismo.

1.3 Complejidad Algorítmica

Existen problemas de extrema dificultad con elevado índice de complejidad algorítmica, ejecución paralela, tareas intermedias, procesos previos de preparación, dependencia, ente otros. Pero se dan fundamentalmente debido a un sinnúmero de combinaciones de sus componentes durante el proceso de solución, haciendo imposible el manejarlos eficientemente.

A continuación se describe la complejidad algorítmica en mención para una mejor comprensión de la problemática. Los problemas de acuerdo a su complejidad se dividen en 2 grupos:

- Problemas con solución óptima mediante tiempo polinomial [2], que quiere decir que el tiempo que toma obtener su solución crece proporcionalmente de acuerdo a las instancias de entrada. Estos problemas son los denominados fáciles y conocidos.

- Problemas NP (polinómico no determinístico), que no tienen una solución óptima (no existe un algoritmo con complejidad polinomial que los resuelva) pero sí se ha logrado obtener una solución (no óptima ni eficiente), muchas veces tomando espacios de tiempos muy extensos para pequeñas instancias [6].

En 1971, trabajando sobre los problemas NP, Cook demostró que existen problemas en NP que son a la vez "especialmente difíciles". Él los denominó Problemas NP completos (NP difíciles). Estos problemas constituyen los verdaderamente relevantes (son la gran mayoría de problemas que se observan en las telecomunicaciones, en la economía, el comercio, la ingeniería, industria – áreas de producción – o medicina). Como pertenecen a la clase NP no es conocida su solución óptima (no existe un algoritmo con tiempo polinomial que los resuelva), pero a diferencia de los NP no existe manera para encontrar su solución exacta –ni siquiera para instancias pequeñas– debido a que la cantidad de variables es inmensa y es prácticamente imposible trabajarlas una a una [20].

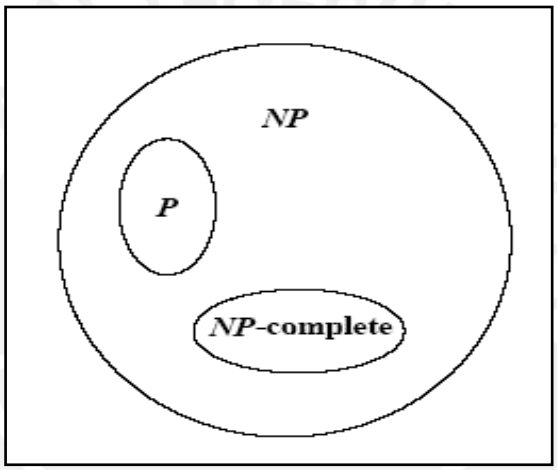


Figura 2 Ubicación de los problemas NP-complejos dentro de la complejidad algorítmica

El problema de la planificación industrial –tema del presente tema de investigación– es uno de la clase NP difícil, debido al gran número de combinaciones máquinas-tareas existentes en la búsqueda de solución, lo cual hace imposible poder precisar una salida aún en instancias pequeñas. Esta situación justifica la aplicación de algoritmos aproximados en la búsqueda de solución bajo tiempos razonables.

1.4 El problema de la planificación industrial

1.4.1 Definición

El problema de la planificación industrial, en su sentido más amplio, consiste en asignar un adecuado orden de ejecución a un conjunto de operaciones que se deben efectuar de forma concurrente a un producto, persiguiendo varios criterios: eficiencia, calidad, tiempo, costo, en una línea de procesos o producción [1].

En la búsqueda por resolver problemas de planificación industrial se opta entre dos alternativas. La primera consiste en intentar resolverlos en forma exacta, con el riesgo de incurrir en dilatados tiempos de computación, posiblemente impracticables. La segunda implica usar métodos aproximados con un alto grado de acercamiento al óptimo, invirtiendo menos tiempo en encontrarlas.

1.4.2 El problema del Flow Shop Scheduling como variante de la planificación industrial

Una de las variantes del problema de planificación industrial es el Flow Shop Scheduling Problem (en adelante FSSP). Esta variante es mayormente utilizada en líneas de producción industrial, en donde un producto recorre una línea de producción en la cual se aplican varias tareas al producto, generalmente una seguida de otra (generadas por máquinas) hasta completar su elaboración.

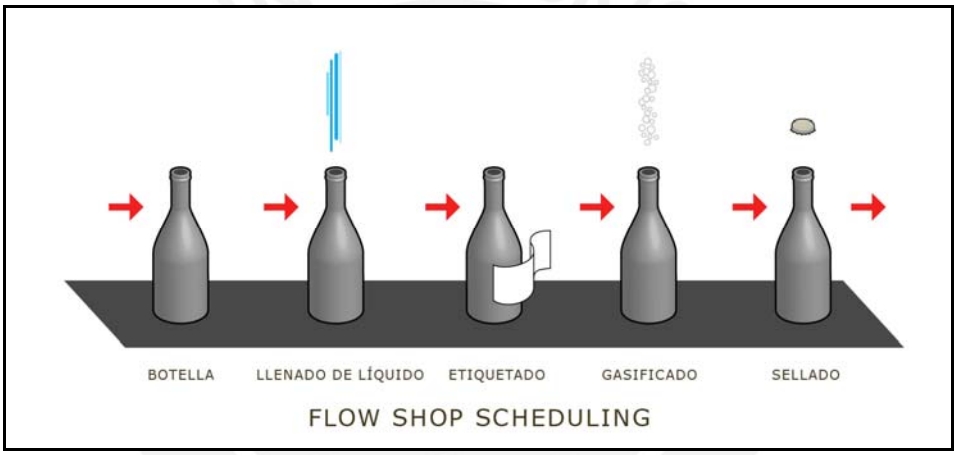


Figura 3 Ejemplo de una línea de producción FSS

Las tareas se clasifican de acuerdo a la prioridad de ejecución dentro del proceso productivo:

- Dependientes: aquellas tareas que deben esperar el término de ejecución de otras con mayor prioridad.
- Independiente: aquellas tareas que no dependen de otras para ser ejecutadas.

Las máquinas se clasifican de acuerdo al tiempo que demanda ejecutar una tarea:

- Homogéneas: aquellas que aún siendo de características mecánicas o electrónicas distintas, toman tiempos similares (no necesariamente iguales) en ejecutar una tarea.
- Heterogéneas: son aquellas que toman tiempos de ejecución por tarea marcadamente distintos.

El FSSP concentra su complejidad en escoger –dentro de un amplio marco de combinaciones– tanto la tarea como la máquina apropiada, cumpliendo restricciones previamente impuestas.

El enfoque que se ha dado, es en aquellos procesos de producción donde las máquinas son semejantes y las tareas independientes unas de otras. Diferenciándose de trabajos existentes en que se realiza doble procedimiento para escoger tanto la máquina como la tarea apropiada, respectivamente, que se denominó doble criterio de relajación novedoso dentro de la literatura sobre el tema.

Como se puede apreciar, las dificultades expuestas ameritan la creación de un algoritmo aproximado, siendo las soluciones exactas imposibles de realizar ya que con una instancia de tres máquinas y "*t*" tareas, el FSSP es un problema NP-completo.

1.5 Conceptos generales sobre heurísticas y meta heurísticas

Dado que en el presente trabajo de investigación se desarrollará una técnica heurística, se incluirán a continuación comentarios sobre la naturaleza de estas técnicas algorítmicas.

En la vida real existen problemas en los cuales el nivel de factibilidad en la resolución de los mismos varía considerablemente; algunos son sencillos, otros medianamente complicados y otros muy complejos. Por desgracia, los problemas de esta última categoría son los que se presentan en las industrias con mayor frecuencia: empresas de comunicaciones, plantas de producción, y demás, debido a que forman parte de su proceso de producción, distribución, entre otros.

Para intentar plasmar y entender mejor esta clase de problemas, se hizo necesario el apoyo de las ciencias exactas (ciencias matemáticas) explicándolos mediante modelos matemáticos, los mismos que suponen ser los más explícitos y cercanos a la situación real.

Esta tarea no era nada sencilla ya que el modelo estaba conformado por muchas variables –restricciones sobre las mismas– conduciendo a un modelo muy complejo, el cual al ser implementado mediante la informática (con el fin de ayudarse con la rapidez de ejecución de los elementos electrónicos que en ella se emplean), conllevaba a una implementación también extensa y compleja debido a las grandes cantidades de variables, lenguajes de programación específicos, y Hardware diseñado y construido a medida (computadores con más de un procesador).

En síntesis, eran muchos factores los que complicaban dicha solución: la complejidad del modelo matemático, el tiempo empleado en encontrar la solución, el personal altamente calificado matemáticamente y el excesivo costo computacional en la implementación (Hardware y tiempo de procesamiento).

Tal y como se mencionara anteriormente, tales problemas aparecen cada vez con mayor frecuencia, mas no se dispone de mucho tiempo para encontrar la solución. Estos fueron los precedentes para hallar una nueva forma de encarar problemas tales, aplicando principalmente el conocimiento previo acerca de la manera en que se ejecutan los mismos, guiándonos a encontrar una solución cercana a la óptima, con mucho menos tiempo invertido y bajo costo computacional. Tales técnicas son los algoritmos heurísticos.

1.5.1 Concepto de heurística

Se habla de heurística para referirse a una técnica o método inteligente, para realizar una tarea que no es producto de un riguroso análisis formal, sino del conocimiento experto sobre un tema a solucionar, la cual aporta soluciones a problemas combinacionales con un buen rendimiento en lo referente a calidad de soluciones y a los recursos empleados, procurando cierto grado de confianza al encontrar soluciones de alta calidad con un costo computacional razonable, aunque no garantice su óptimo rendimiento o factibilidad, e incluso en algunos casos, sin lograr establecer lo cerca que se está de dicha situación. Se usa el calificativo heurístico en contraposición a exacto [5].

Ackoff y Muller-Merbach han señalado que todo problema o situación del mundo real, puede separarse en 4 pasos para su solución.

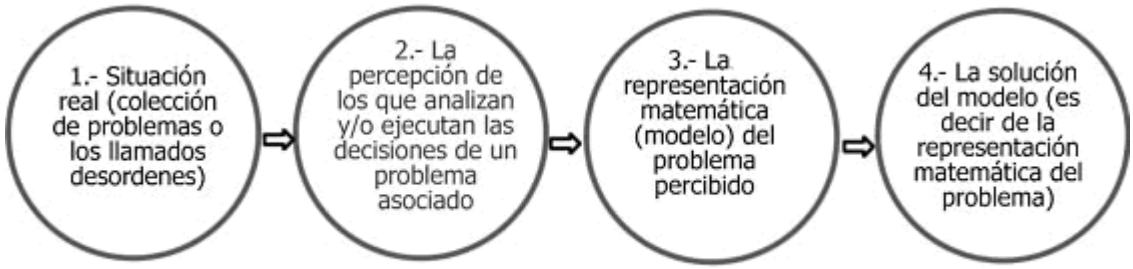


Figura 4 Del problema de la vida real a la solución

El objetivo del paso tres es encontrar un modelo matemático capaz de proveer el valor óptimo de cualquier solución específica en donde hay restricciones concretas que definen la región de solución viable. Si bien es cierto que la heurística entra a tallar más en el paso cuatro, debido a que se escogen las técnicas adecuadas, el paso tres también involucra el uso del razonamiento heurístico, pues en comparación con los modelos matemáticos, éstos no tienen la necesidad de acudir a suposiciones y rutinas de optimización complejas, en cierta medida restrictivas en la formulación del modelo. Todo ello permite que los modelos sean más representativos para el problema real [9].

1.5.2 Motivos para el uso de Heurísticas

Los principios para el uso de técnicas heurísticas se deben a que las soluciones exactas pueden ser difíciles de obtener –sino imposibles– para una formulación matemática (razonablemente representativa del problema real).

Existen tres circunstancias importantes:

1. Una *explosión de posibles combinaciones* de variables decisorias que puede llegar a ser óptimas.
2. *Dificultad en la evaluación de la función objetivo* por la presencia de restricciones probabilísticas debido las variables estocásticas (variable que puede tomar valores aleatorios).
3. *Condiciones que cambian marcadamente con el tiempo*, requiriendo una serie de soluciones a todo momento. Es preferible que una única solución exista en un punto determinado en el tiempo.

1.5.3 Factores para el uso de Heurística

Los factores que favorecen en la toma de decisión de implementar un problema aplicando heurística, son los siguientes [9]:

Facilidad en la implementación: Las personas prefieren convivir con un problema que no pueden resolver que aceptar una solución que no pueden entender. Este tipo de entendimiento es más probable con reglas heurísticas que con una rutina compleja de optimización. Tal razonamiento no necesariamente implica que las heurísticas deban ser simples en su naturaleza, pues en efecto para algunos problemas complejos las heurísticas simples no podrán producir soluciones aceptables.

Argumenta mejores soluciones sobre prácticas vigentes: Relacionados con el punto anterior, los Managers estarán más satisfechos con la solución heurística debido a que producen mejores resultados que los logros hasta la fecha obtenidos.

Resultados óptimos: Algunas veces rápidos, otras veces razonables, pero en su conjunto, menores a lo que demoraría una solución exacta, los resultados son muy cercanos al óptimo, y las heurísticas pueden ser desarrolladas y usadas con mayor velocidad que las rutinas de optimización.

Solidez: Las heurísticas son menos sensibles a la variación de calidad de información y a las características de los problemas. Las soluciones óptimas son frágiles en el sentido que son exquisitamente sensibles a los cambios en la información si la descripción del problema cambia ligeramente, el recobrar la solución óptima requiere normalmente resolver el problema entero nuevamente (lo cual computacionalmente hablando en cálculos, es costoso en un primer momento) y por otra parte, las heurísticas particionan el problema y de esa forma ignoran las interrelaciones entre sus partes. Esto permite que las modificaciones sean dirigidas a las partes afectadas de manera más rápida sin necesidad de recomputar todo el problema.

Son usadas dentro de las rutinas de optimización: Las heurísticas pueden ser usadas provechosamente dentro de rutinas de optimización por dos motivos: primero, proveen una buena solución inicial en un plan interactivo, y segundo, pueden proporcionar límites que facilitan la eliminación de porciones en un espacio de soluciones con miras al proceso de optimización.

La aplicación de las heurísticas es razonable en varios casos pero no es extensible a todo problema. Los factores que justifican su uso, son los siguientes [1]:

- Cuando la solución exacta no es primordial.
- Cuando existen limitaciones de tiempo y en Hardware
- Cuando es parte de un algoritmo mayor, siendo paso intermedio, como solución previa, para obtener la solución final.

1.5.4 Principales tipos de métodos heurísticos

Existen siete clases de métodos heurísticos entre los más destacados para resolver problemas combinatorios [1,9]:

1. Soluciones generadas aleatoriamente.
2. Problemas de descomposición y partición.
3. Métodos inductivos.
4. Métodos que reducen los espacios de solución.
5. Métodos constructivos.
6. Métodos de manipulación del modelo.
7. Métodos de búsqueda por entornos.

Estas clases de heurísticas no son necesariamente excluyentes cuando son usadas en la solución de problemas. Muy frecuentemente es necesario unir dos o más para encontrar la solución adecuada, o también es válido desarrollar un problema aplicando heurísticas en paralelo, escogiendo la mejor solución.

La decisión sobre qué heurística emplear depende de ciertos factores, mencionados a continuación:

i. El tipo de decisión estratégica, táctica u operacional.
ii. La frecuencia con que la decisión es tomada.
iii. El tiempo disponible en el desarrollo.
iv. El análisis de calidad de la decisión involucrada.
v. El tamaño del problema (incluyendo el número de variables).
vi. La ausencia o presencia de elementos estocásticos.

1.5.4.1 Soluciones generadas aleatoriamente

Un concepto relativamente enérgico consiste en aplicar soluciones generadas aleatoriamente, evaluar cada una y escoger la mejor. Usualmente, uno decide la cantidad de soluciones a generar para buscar un porcentaje alto de acercamiento a la solución real, pero es necesario tener en cuenta que tales soluciones no son de alta calidad debido a que no se piensa en las

características del problema o de retroalimentación de las mismas a lo largo de la solución (característica relevante sí, en las meta heurísticas desarrolladas más adelante).

1.5.4.2 Problemas de descomposición y partición

Consiste en descomponer o fraccionar el problema en subproblemas presumiblemente más fáciles de solucionar. Esta partición puede ser jerárquica en cuanto a las decisiones a tomar (de mayor a menor importancia), o en cuanto al tipo de recursos involucrados. Por ejemplo, máquinas diferentes en una línea de producción o cronológicamente en tiempo.

Una vez definido los subproblemas se pueden seguir cualquiera de los siguientes caminos:

- Resolver los subproblemas independientemente y unirlos en una solución viable.
- Resolver los subproblemas secuencialmente empleando como entrada del segundo el resultado (salida) del primero y así sucesivamente. Frecuentemente visto en líneas de planificación.
- Resolver los subproblemas iterativamente y no secuencialmente. Esto es muy usado en situaciones de múltiples recursos compartidos por actividades compuestas.

1.5.4.3 Métodos que reducen los espacios de solución

Estos métodos obedecen a cortar de raíz aquellas soluciones que no aportan beneficios a la característica buscada, en otras palabras, aportan a reducir el espacio de solución para aquellas soluciones que no cumplen con las restricciones del modelo; sin embargo, existe un riesgo: es esencialmente miope y elimina soluciones que pueden resultar óptimas.

1.5.4.4 Métodos constructivos

Basados en el aumento paulatino de los componentes individuales en los conjuntos solución de los problemas hasta la obtención de una solución factible (que además, cumple con las restricciones) y razonablemente óptima. Los métodos constructivos más conocidos son los algoritmos VORACES MIOPES.

1.5.4.5 Métodos de manipulación del modelo

Son aquellos que buscan modificar la estructura del modelo original del problema, generando soluciones alternas a estas modificaciones que conllevan al cese del problema base. Consiguen reducir el espacio de soluciones o de búsqueda.

1.5.4.6 Métodos de búsqueda por entornos

Constituyen la base de la mayoría de las meta heurísticas. Estos métodos parten de una solución factible inicial y, mediante alteraciones de dicha solución, van pasando de forma iterativa hasta cumplir una condición de optimalidad o criterio de parada. Así, almacenan a

cada paso los mejores óptimos locales visitados. Entre las meta heurísticas más conocidas que aplican estos métodos encontramos las búsquedas Tabú y Simulated Annealing.

1.5.5 Algoritmos voraces miopes

Los Algoritmos Voraces miopes [20] (en adelante ***AVM***) toman este nombre debido a las siguientes razones:

- *Voraces*: debido a que escoge el mejor candidato de un conjunto de soluciones, éste formará parte del conjunto solución, siempre y cuando al ser evaluado por la función objetivo, obtenga el mejor resultado respecto a los demás candidatos.
- *Miope*: porque sólo toma en cuenta la situación actual (vale decir, es el mejor en este momento) sin importar los cambios que puedan ocurrir a lo largo del procedimiento pudiendo estancarse en un valor fijo o reducirse una vez escogido. El valor escogido se mantiene hasta el final, lo mismo sucede para aquellos valores que son rechazados; no son reconsiderados nuevamente.

Los AVM producen soluciones rápidas y satisfactorias debido a que utilizan el conocimiento acerca del problema a solucionar mediante las restricciones establecidas para dicho problema. Pero tiene dos principales deficiencias:

La primera es que (estos algoritmos) dependen mucho de las características de las instancias que intentan resolver, es decir, pueden arrojar muy buenos resultados para determinadas instancias del problema, pero para otras no.

El segundo inconveniente [1] es que se limitan a los óptimos locales de las funciones que pretenden optimizar y, probablemente, no analizan vecindades más allá del criterio voraz, por lo cual pueden no considerar al óptimo global.

Por ejemplo, en un espacio de dos dimensiones (Figura 5) los algoritmos heurísticos no recorren el total de espacio de posibles soluciones, sino vecindades (subconjuntos) dejando pasar por tanto, valores que originarían soluciones globales.

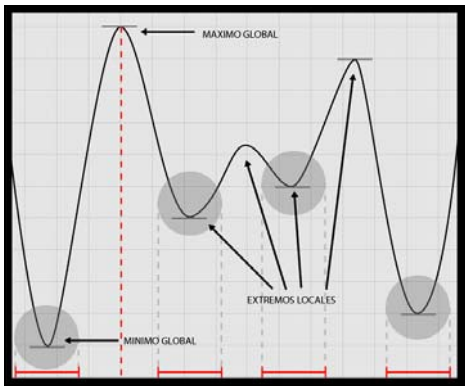


Figura 5 Máximos y mínimos globales

Expresamos un AVM como un problema de Optimización. Sean las siguientes variables:

E *= {e1, e2, e3, e4,.., eN} es una colección de N objetos o variables del problema.*
F*: es una colección de subconjuntos de elementos de E que cumple con una determinada propiedad.*
c: E → R *es la función a optimizar.*
S*: conjunto solución.*

El algoritmo voraz miope como problema de Optimización [1] sería el siguiente:

Algoritmo Voraz miope maximización (N,c,S,E,F)

Inicio

Leer N, c, E, F
1. Ordenar los elementos de E: c(e1) ≥ c(e2) ≥ ... ≥ c(e_N) >0
2. S:= ϕ
3. Para i: 1 a N, hacer
3.1 Si S ∪ {ei} es una solución viable ⇒ S = S ∪ {ei}
Fin Voraz miope maximización.

Algunos comentarios sobre el algoritmo presentado:

- La línea 1 muestra un ordenamiento ascendente de los valores que resultan de aplicar la función objetivo a todos los elementos del conjunto E. Es ascendente en cuanto persigue maximizar el resultado.
- En la línea 2 se inicializa el conjunto S, el cual será la solución final.
- En la línea 3, 3.1 para cada elemento del conjunto N, se realiza una verificación, si cumple la condición impartida –en este caso solución viable– será añadida al subconjunto solución final.

Como se mencionó anteriormente, el AVM presenta deficiencias (dependencias de las instancias relacionadas y estancamiento en óptimos locales) debidas principalmente a que escoge sólo una solución por instancia. Una mejora para este algoritmo son los algoritmos GRASP que pertenecen a las meta heurísticas. Éstos escogen no una solución, sino un conjunto de soluciones que cumplen una condición (función objetivo con cierto relajamiento), permitiendo así tener más posibilidades de obtener el óptimo global deseado.

1.6 Algoritmos Meta heurísticos

El término meta heurística apareció por primera vez en el artículo seminal sobre búsqueda tabú de Fred Glover, en 1986. A partir de entonces, han surgido un sinnúmero de propuestas y pautas –combinando técnicas de Inteligencia Artificial, evolución biológica y los mecanismos estadísticos– para diseñar buenos procedimientos en la resolución de problemas de Optimización combinatoria [7] Estos procedimientos al ampliar su campo de aplicación, han adoptado la denominación de meta heurística. Las meta heurísticas son estrategias inteligentes para diseñar, mejorar y optimizar procedimientos heurísticas muy generales, otorgándoles un alto rendimiento con respecto a las heurísticas tradicionales. De ahí que se le agrega el sufijo meta, que significa más allá o a un nivel superior.

Se detalla a continuación una técnica meta heurística (GRASP) para efectos de la presente tesis.

1.6.1 Algoritmos GRASP

GRASP (*Gredy Randonize Adaptative Search Procedure*) es una técnica meta heurística que debido a su estructura –dos partes principales, una de construcción y otra de búsqueda– pertenecen también a la clasificación de algoritmos multi-arranque [4]. El nombre proviene de:

- Greedy (voraz): porque el criterio de escoger el mejor valor después de cada evaluación prima para seleccionar los candidatos.
- Randomized (aleatorio): porque después de tener la lista de candidatos escoge a uno de ellos al azar.
- Adaptive (adaptable): porque es capaz de adaptarse a diferentes contextos de aplicaciones o modificaciones relevantes al modelo.
- Search Procedure: porque realiza una búsqueda dentro de un espacio o vecindad, evaluando aleatoriamente a cierto número de integrantes.

Desarrollada a finales de los años 80 [3] con el propósito de resolver el problema de cubrimiento de conjuntos, el término GRASP, introducido por Tomas Feo y Mauricio Resente el año1995, se presenta como una nueva técnica meta heurística de propósito general. Sin embargo, es en el campo de la Optimización Combinatoria donde alcanzó resonancia debido a su gran acercamiento y rapidez de convergencia, al encontrar soluciones muy cercanas al óptimo global.

Mientras el argumento de los Algoritmos Voraces se centra en escoger al que posee en ese instante el mejor valor, el criterio GRASP se relaja y amplía dicha restricción, seleccionando no al mejor, sino a un elemento de dentro de un conjunto de valores (previamente evaluados por la función objetivo) garantizando la superación del carácter voraz. Dichos valores no son necesariamente los óptimos locales, pero serán los principales gestores para encontrarlos mediante búsquedas locales.

El poseer no uno sino una lista de candidatos de solución –además de la aleatoriedad respectiva– permite gozar de amplia ventaja frente al criterio voraz, porque cubre una mayor cantidad de puntos espectro-solución.

A continuación, la estructura básica del algoritmo GRASP:

Procedimiento GRASP (Instancia del problema)

1. *Leer (Instancia)*
2. *Mientras <no se cumpla condición de parada> hacer*
   - *2.1 Procedimiento Construcción (S_k)*
   - *2.2 Procedimiento Mejorar (S_k)*

   Fin Mientras
3. *Retornar (Mejor S_k)*

Fin GRASP

Fase de Construcción GRASP

Como se sabe, el algoritmo Voraz se establece al escoger el mejor valor en el momento en que el conjunto de valores de ingreso son evaluados por la función Objetivo. Dicho valor pretende ser el óptimo en vista que superó a los demás; sin embargo, el criterio Miope aplicado en tal selección no garantiza que sea el óptimo-global, al no evaluar todo el espectro de ingreso (muy amplio en la mayoría de casos) que de hacerse efectivo comportaría un costo excesivo, dejando de ser una meta heurística, y sin mejora alguna.

En algunos casos, el algoritmo GRASP propone un conjunto de valores candidatos a ser los óptimos. Dicha lista se denomina RCL (*Restricted Candidates List*). Para encontrar tales valores, previamente se evalúan los valores de ingreso con la Función Objetivo, encontrándose así, dos valores representativos: el mejor y el peor valor.
En el siguiente gráfico se muestran respectivamente:

$\beta = Mejor\{c(x) : x \in N\}$

$\tau = Peor\{c(x) : x \in N\}$

Con valores semejantes se fija el criterio de selección para obtener el RCL, expuesto a continuación:

$$RCL = \{x \in N : \beta \leq c(x) \leq \beta + \alpha(\tau - \beta)\}$$

El parámetro α empleado se denomina parámetro de relajación y enmarca el tamaño del segmento RCL, por lo cual demanda un completo entendimiento del problema a resolver, así como de la estructura del mismo. Tomar valores muy pequeños puede omitir soluciones potencialmente valederas y, si, por el contrario, el valor es demasiado alto, se corre el riesgo de asimilar valores que no contribuyan en el hallazgo del resultado.

Existen dos casos extremos y relevantes a mencionar: cuando el valor de α es 0 (en donde el RCL se limita a un solo valor que para el ejemplo sería el valor de menor demora en realizar todas las operaciones) impera un criterio miope, al escoger el mejor, lo cual no garantiza que sea el óptimo, dado que posteriormente pueden practicarse operaciones de mayor tiempo.

En contraparte, cuando el valor de α es 1, son procesados TODOS los elementos (incluyendo soluciones buenas o malas, lo cual contempla un exceso al procesar con valores que no se inclinan a un óptimo, es decir, ocasionando costos innecesarios)

Seguidamente se toma al azar un valor del RCL ("*a*"). Se verifica que el conjunto solución unido con el valor "a", es viable o no [1]. Si es afirmativo, entonces se añade "a" al conjunto S, eliminándolo ("a") de los valores de entrada. Se prosigue con las siguientes iteraciones.

Tal distribución de orden es fundamental en vista que expresa el carácter aleatorio del GRASP, permitiendo que en la fase de mejora se obtenga –mediante la búsqueda local– valores distintos.

Se presenta a continuación, la estructura básica de esta fase:

Procedimiento Construcción(N, c, E, F, S, α)

Inicio

Leer c, E, α

$S = \phi$

N =E

1. Mientras <no se cumpla condición de parada> hacer

Inicio

1.1 $RCL = \phi$

1.2 $\beta = Mejor\{c(x) : x \in N\}$

1.3 $\tau = Peor\{c(x) : x \in N\}$

1.4 $RCL = \{\forall x \in N : \beta \leq c(x) \leq \beta + \alpha(\tau - \beta)\}$

1.5 $a = Aleatorio(RCL)$

1.6 Si $S \cup \{a\} \in F \Rightarrow S = S \cup \{a\}$

1.7 $N=N-\{a\}$

1.8 Adaptar c

Fin Mientras

Fin Construcción.

1.6.2 Fase de Mejora

Una vez preseleccionados los elementos X_0 por la fase de construcción, éstos son tomados como punto central de entre varios otros valores generados aleatoriamente, sea mediante perturbaciones de costos [1] sea bajo un criterio que demande relación entre ellos. Este espacio lo denominaremos Vecindad, el cual puede ser un círculo cuyo centro es el valor inicial "X_0", para el caso de un par ordenado o para el de secuencia de tareas, que a su vez pueden ser otras secuencias alteradas en el orden.

En cualquiera de los casos, el criterio es el mismo: generar valores cercanos, los mismos que serán evaluados por la función $C(x)$ y comparados con el valor de X_0. Si los supera será asignado como nuevo valor X_0; caso contrario, se tomará el siguiente valor de la vecindad hasta llegar a una regla de parada [4] debido al número de mínimos locales, (distinto de la función objetivo) o a causa del número de interacciones necesarias para alcanzar el mínimo local. Así, el valor final es agregado a un conjunto posterior C_0. En seguida, regresa a tomar el siguiente valor del RCL y repite el mismo procedimiento, para finalmente escoger el mejor valor del conjunto C_0 como óptimo global.

CAPITULO 2

2 El problema de la programación de tareas en líneas de producción

2.1 La planificación industrial como antecedente de la programación de tareas

Para referirse a la planificación es necesario definir previamente el concepto de línea de producción. Una ***línea de producción*** es el conjunto de tareas que se realizan en un organismo de producción (una fábrica, para el caso de planificación industrial) cuyo propósito es la elaboración de uno o varios productos. Por lo tanto, una ***planificación*** es la asignación adecuada del orden de ejecución de tareas en la línea de producción, logrando su ejecución concurrentemente o no.

La planificación posee tres componentes fundamentales:

- Tareas, es decir, trabajos a ejecutar.
- Máquinas que ejecutan las tareas.
- Algoritmos, o forma bajo la cual se organiza la ejecución de tareas en las máquinas de manera óptima y sin desperdiciar recursos ni tiempo.

Para entender los resultados de una planificación es necesario, asimismo, contar con un análisis del plan resultante. Éste método permitirá comprender los resultados parciales y/o finales cuyo corolario será el de esquemas fácilmente comprensibles, ordenados y jerárquicos; y, de esta manera, determinar si las expectativas fijadas van cumpliéndose o no.

La unión de ambos conceptos (planificación y análisis del plan resultante) conforman un método o estrategia de planificación, objetivo a lograr en una fábrica cuyo deseo es la óptima producción.

Los primeros trabajos sobre *scheduling* [66] se iniciaron desde el campo de la Investigación Operativa, resolviendo instancias en una línea de producción para dos máquinas [12]. El problema en mención se caracterizaba por su reducción a una formulación matemática, solucionándose con métodos algorítmicos formales.

Otros trabajos que intentaron resolver el problema de la planificación en forma exacta fueron el de Graphplan a mediados de los años 80 creado por Avrim Blum y STRIP en el año 1971 por desarrollado por Fikes y Nilsson. Ambos trabajos, revolucionaron con sus planteamientos, se anticiparon fijando el inicio para posteriores investigaciones, sin embargo, las desventajas radican en lo genérico de sus procedimientos; tanto así que, para su aplicación será necesario forzar situaciones, contextos del propio *scheduling* –necesarios para encajar en condiciones que ellos mismos proponen– por lo que la aplicabilidad de tales restricciones a variantes complejas es sencillamente imposible [1].

2.2 Definición general del problema del Scheduling

El problema del scheduling se define mediante un conjunto *J* de trabajos y un conjunto *M* de recursos (se mencionarán indistintamente recursos o máquinas).

La producción de un trabajo determinado J_i, requiere la ejecución de una secuencia de operaciones $\{o_{ij}\}$ que denotan el uso del recurso m_j en el trabajo j_i. Llamamos *O* al conjunto de operaciones especificadas en el scheduling. Como restricciones tecnológicas, cada operación o_{ij} requiere el uso exclusivo de un recurso m_j durante un determinado tiempo de procesamiento $dur(o_{ij})$.

Además, las operaciones de un mismo trabajo deben ejecutarse en un orden determinado, *precedencia* $(o_{ij},o_{(i+1)j})$, de forma que $o_{(i+1)j}$ no puede comenzar hasta que o_{ij} haya terminado completamente. Asimismo, las operaciones que comparten una misma máquina son continuas y mutuamente exclusivas.

El objetivo del proceso es encontrar una asignación de tiempos a las operaciones, de forma que se completen todos los trabajos en el mínimo tiempo posible. A la secuencia de orden de las operaciones se la denominará *patrón de flujo* a partir de este momento. El *patrón de flujo* viene determinado por las precedencias que tiene los trabajos: $precedencias(J), \quad \forall\, i=1..n.$ [11].

Las variantes existentes para el *scheduling* dependen de las características de sus componentes (máquinas, tareas, objetos a optimizar), así como de las formas en que son ejecutadas y del patrón de flujo aplicado.

2.2.1 Objetivo del Scheduling

El objetivo principal de una planificación (de la industria, en particular) es el ahorro de costos. Lo cual refiere a:

- Cero *máquinas ociosas* durante la producción, caso contrario se incurre en gastos de mantenimiento.
- Ahorro de energía al no dejar máquinas encendidas esperando turno en el proceso de producción consumiendo energía necesaria.
- Reducción en el tiempo de producción del lote, conocido como *makespan.*

2.2.2 Propiedades de los componentes del scheduling

Según lo mencionado anteriormente, las variantes de la planificación dependen de la naturaleza de sus componentes, en especial, lo concerniente a tareas y máquinas (éstas se abordan en los capítulos siguientes)

2.2.3 Propiedades de las tareas

- En cuanto a su aparición en las líneas de producción, se tiene las del tipo *lote*: tareas conocidas y disponibles, en donde existe más de una secuencia posible de ejecución. ejemplo: industria de empaquetado (luego de la caja, se echan las bolitas de tecnopor y luego el contenido, o primero el contenido y luego las bolitas). Y otras del tipo *lineal*, vale decir, que van ejecutándose una tras otra (secuencialmente), donde la siguiente tarea es única.

- En relación a su comportamiento se dividen en: dependientes o independientes en su ejecución; donde, para el caso de dependientes, se requiere de la ejecución de las tarea predecesora; mientras que para independientes, todo lo contrario, no habrá necesidad de tarea predecesora alguna.

2.2.4 Propiedades de las máquinas

- Restricción en cuanto al número de operaciones que ellas ejecutan: existen máquinas capaces de realizar una única tarea, y otras que están diseñadas para realizar dos a más (pero no paralelamente). Además, dicha restricción incluye a aquellas máquinas que por su diseño pueden trabajar en turnos continuos y otras que no, como por ejemplo, debido al riesgo de sobrecalentamiento.
- De acuerdo al tiempo que demoran en ejecutar las tareas.- siendo máquinas homogéneas (aquellas que ejecutan las tareas en tiempos semejantes) y heterogéneas (aquellas que las ejecutan en tiempos distintos).
- Restricciones de ejecución.- se refieren a máquinas que pueden ejecutar cualquiera de las tareas, algunas o exclusivamente una tarea.

2.3 Variantes del problema del scheduling

Las variantes del scheduling dependen de la naturaleza de sus componentes, pero principalmente de la secuencia de ordenamiento que es aplicada. Así, si todos los trabajos presentan la misma secuencia en el orden de las operaciones, el problema se define como ***flow shop scheduling*** (donde el orden de los elementos de cada trabajo será el mismo:

precedencias(J_x)=precedencias(J_y), ∀x,y=1..n); mientras que, si cada trabajo tiene un orden determinado, recibe el nombre de ***job-shop scheduling.***

Otra variante, en cuanto al orden de espera de ejecución frente a una máquina, es el *permutation flow shop scheduling*, donde el orden de ejecución es FIFO, mientras que otras variantes tipo *open flow scheduling*, dan la posibilidad de alterar el orden en la cola de espera, trasladando la tarea a otra maquina disponible (por la prioridad que tiene ésta tarea frente a las demás).

Para todas estas variantes el tiempo mínimo requerido para completar los trabajos es llamado ***makespan.*** El objetivo de resolver u optimizar este problema es determinar la programación de tareas que minimicen el *makespan* [11].

Se presenta a continuación una descripción de las más importantes variantes del problema del scheduling:

2.3.1 Job scheduling

Es una de las variantes del *scheduling* más difundida y estudiada, posee por tanto amplia variedad de algoritmos (heurísticos y meta heurísticos) que pretenden solucionar tales problemas.

El Job Shop Scheduling Problem (de ahora en adelante JSSP) consiste en un conjunto de trabajos y otro conjunto M de recursos (llamados comúnmente máquinas) en donde cada ***trabajo*** es dividido en sub-trabajos, a los que se denominará ***Actividades***; siendo cada una de estas *actividades* asignada a algún recurso del conjunto M, basándose en ciertos criterios.
El orden de ejecución de las *actividades* es independiente al *trabajo* al que pertenecen, vale decir que, no es prioridad ejecutarlas en orden al trabajo. La actividad es ejecutada siempre y cuando las actividades predecesoras (las que comprenden el trabajo) hayan sido concluidas. Por lo que se determina que, en esta variante (JSSP), el criterio del predecesor es importante [12,13 y 14].

2.3.2 Flow Shop Scheduling

El Flow Shop Scheduling (de ahora en adelante FSSP) *consiste en un número de trabajos que* son procesados en un conjunto de máquinas. Cada trabajo debe ser procesado por todas las máquinas en exactamente el mismo orden (es decir cada trabajo debe procesarse primero en la máquina 1, luego en la máquina 2 y así sucesivamente hasta la máquina n).
Cada máquina deberá procesar sólo un trabajo en un determinado momento y cada trabajo debe ser procesado como máximo en una máquina en un punto de tiempo. Todos los trabajos están listos para el procesamiento en el tiempo inicial y no existe un tiempo límite de entrega.

Existen buffers (lugares donde se coloca temporalmente el bien, en espera de la ejecución del precedente, o para realizar sobre él algún trabajo intermedio) con espacio limitado entre máquinas, pero la capacidad del buffer para la primera máquina es ilimitada.

El objetivo consiste, por tanto, en encontrar una planificación que asigne todos los trabajos en las máquinas disponibles en el menor tiempo posible [6,15].

Además, debido a que existe un recorrido común a seguir, se agrupan las máquinas según la función que cumplen (máquinas idénticas) disponiéndolas en paralelo, una detrás de otra, como se muestra en la siguiente Figura:

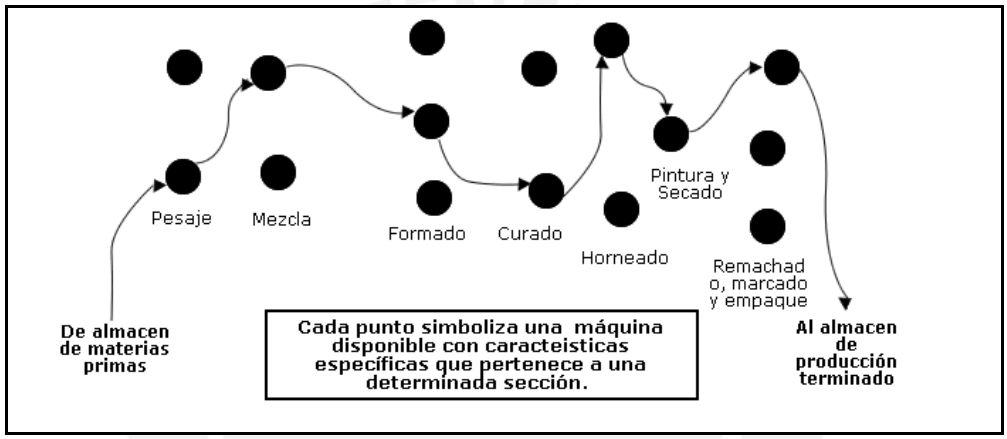


Figura 6 Recorrido en una línea de producción del tipo FSS de acuerdo a la disponibilidad de las máquinas

Los trabajos algunas veces son divididos en sub-tareas u operaciones independientes entre ellas, pero siempre se respeta la condición de pasar por todo el recorrido (flow) de máquinas [1].

Existen ambientes de línea de flujo (flow shop) donde es imprescindible que se realicen trabajos previos antes de continuar el circuito de procesamiento en las máquinas; por ejemplo, para el caso de las industrias químicas pueden ser requeridos algunos trabajos de limpieza del producto, preparación de la siguiente máquina o re-calibración para la siguiente tarea a ejecutarse en la misma máquina. Este tiempo es independiente al invertido en el procesamiento de la tarea en sí. Las características mencionadas anteriormente presentan una nueva sub-variante del modelo FSSP, llamado FSSP con tiempos de preparación dependientes [6].

Una variante de este problema consiste en considerar que las máquinas colocadas en paralelo no tienen la misma capacidad de procesamiento, lo cual puede deberse a diferencias en cuanto a tecnología adquirida a través de los años. Esto es, que un trabajo puede ser procesado utilizando más o menos tiempo dependiendo de la máquina a la que sea asignado [16].

Algunos autores [60,61] concluyen que, para las líneas de producción donde las máquinas son irregulares; la capacidad del buffer afecta negativamente a la calidad del FSS. Por lo tanto, recomiendan la utilización de buffers finitos entre dichas maquinas.

Pero bajo la forma tradicional de planificación en Flow shop, además de la gran mayoría de investigaciones en FSSP [62,63,64,65], ignoran las consideraciones del tamaño del *buffer*, es decir, lo toma como infinito [15].

El Flow shop Problem ha existido desde mucho tiempo y, a pesar del gran esfuerzo que a lo largo de estos años se ha mostrado por parte de los científicos, el problema –incluso en pequeñas escalas– continúa arrojando un alto grado de complejidad.
Un *Optimal finish time (OFT)* para dos máquinas fue propuesto por *Jonson* [66], pero para tres o más el problema ha sido considerado como NP-difícil.

2.3.3 Task Scheduling

El Task Scheduling, a diferencia de otras variantes, no se divide en actividades (sub-tareas) ni tiene como prioridad seguir una trayectoria determinada y específica, sino que, para que se ejecute una tarea, sus predecesoras ***necesariamente*** tienen que ser ejecutadas. Por tal razón la mejor manera de representar un programa de planificación en el Task Scheduling es usando un gráfico acíclico diseccionado (DAG).

Un DAG consta de un gráfico G (V,E,w,c), el cual representa un programa de planificación P:

- V está conformada por nodos (n) y estos representan tareas de programas a planificar.
- E representa las conexiones entre nodos (e_{ij} pertenece E es la comunicación entre los nodos n_i y n_j).
- w son los pesos asociados al nodo (w(n) y representa el costo computacional que demanda la tarea n).
- c representa el costo de la comunicación entre nodos ($c(e_{ij})$ costo de comunicar los nodos i y j).

Todas las instrucciones u operaciones del gráfico son ejecutadas en orden secuencial, no hay paralelismo sin tarea. Los nodos respetan sus entradas y salidas, es decir, un nodo no puede iniciar su ejecución hasta que sus entradas hayan sido concluidas y ninguna salida se encontrará disponible hasta que la ejecución del nodo haya finalizado. Un nodo sin predecesores es un nodo-inicio [17].

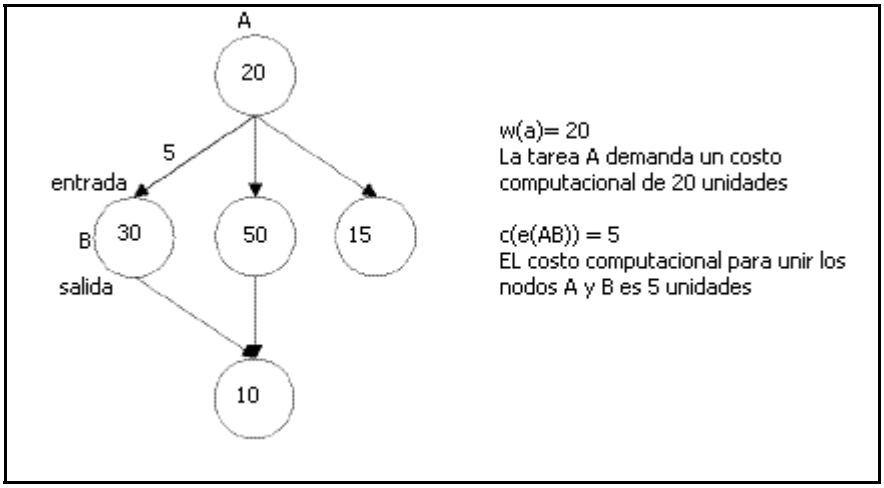


Figura 7 Gráfico Acíclico Diseccionado (DAG)

Una programación viable debe preservar todas sus relaciones precedentes, además sólo es óptimo si minimiza el *makespan* [18]. Dado un *task graf,* el problema clásico del *task scheduling* es encontrar una óptima y viable planificación para la asignación de tareas al procesador. Sin embargo, ha sido demostrado que el *task scheduling* en su forma básica (para tres máquinas) es un problema NP-difícil.

Solo existen algunos pocos algoritmos polinomiales que intentan resolver el *scheduling,* debido a que incluso cuando las condiciones son fuertes o severas, éstas llegan a ser muy complejas.

2.3.4 Programación en tiempo real

La programación en tiempo real obedece a la necesidad de encontrar una manera de planificar las acciones a tomar, frente a situaciones del mundo real (muchas veces repentinas) que interrumpen algún nivel de estado en un sistema de control; por ejemplo, el paso de una caja sobre un sensor colocado al costado de una línea de producción. Estas acciones deben ejecutarse en un tiempo establecido, implicando la colaboración de maquinaria mecánica para realizar la acción [23].

La lógica de la planificación se coloca en los microprocesadores ubicados dentro de los elementos mecánicos antes mencionados [1].

La programación en tiempo real, a diferencia de las variantes del scheduling, no centra su análisis en el comportamiento en una línea de producción, sino en la planificación de las acciones a tomar frente a un estímulo externo.

A continuación se muestra un esquema que ubica el tema de tesis dentro de las divisiones y variantes del scheduling:

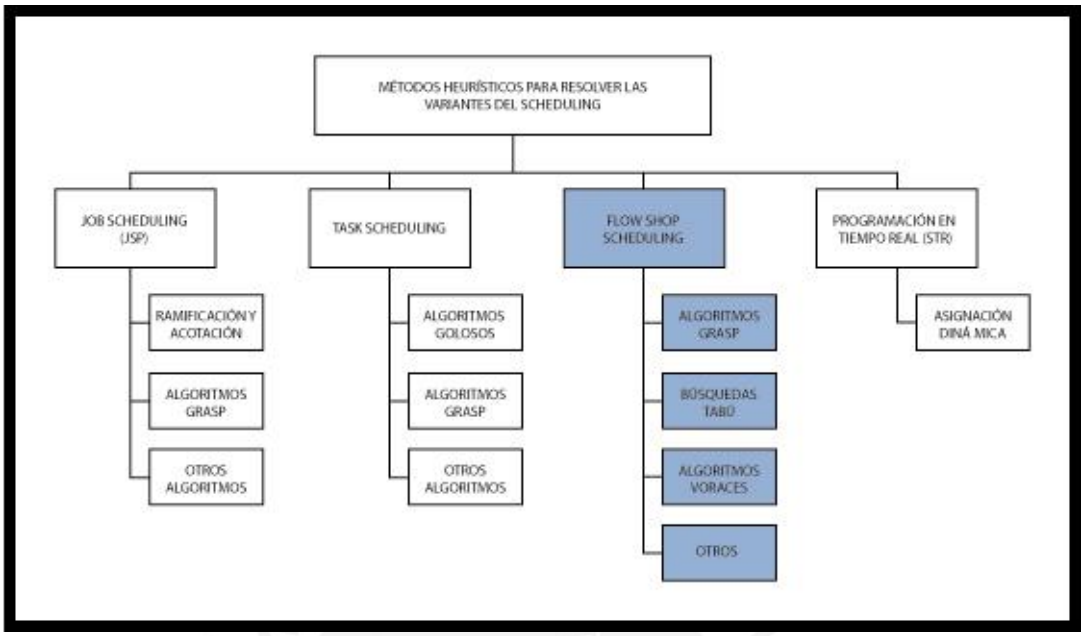


Figura 8 Métodos heurísticos para resolver las variantes del scheduling

2.4 Áreas de aplicación del scheduling

La planificación está presente en varios lugares, objetos, establecimientos y demás situaciones en donde se persiga la optimización de costos. Se detallarán dos casos en particular [32].

2.4.1 Planificación en lugares públicos

En los lugares públicos se concentran diversas y múltiples demandas de atención, y casi siempre los espacios asignados para la atención de dichas peticiones no son suficientes en espacio y número; por tal razón, es necesaria una debida administración y asignación de los recursos, con el propósito de afrontar las necesidades requeridas.

A continuación se ejemplifica la planificación dada en un aeropuerto al momento de asignar una puerta de acceso frente al arribo o partida de aviones [32].

Considerando un terminal aéreo de grandes proporciones –donde existen docenas de puertas y cientos de aviones arribando y partiendo cada día – se presentan diversas restricciones para los criterios mencionados (puertas, aviones y horarios).

- Las puertas no son todas iguales, como tampoco lo son los aviones entre sí.
- Algunas puertas se ubican en lugares amplios donde los jets pueden fácilmente acomodarse, otras no.
- Ciertos aviones deberán ser remolcados hacia las puertas habilitadas para recibirlos.

- Si se le niega el despegue, las políticas de operación usualmente establecen que los pasajeros permanezcan en el terminal antes que en el avión.
- Si el abordaje es pospuesto, un avión podrá mantenerse en una puerta por un periodo extendido de tiempo, de esta manera se previene que otros aviones utilicen esta misma puerta.
- Los aviones arriban y parten de acuerdo a cierto horario.
- El horario está sujeto a un tiempo significativo de aleatoriedad que puede deberse al clima o a eventualidades en otros aeropuertos.
- Durante el tiempo que un avión ocupa una puerta, los pasajeros deberán descender, el avión deberá recibir combustible y los próximos pasajeros deberán abordar.
- El tiempo de partida programado es visto como una fecha de vencimiento y el desempeño de la aerolínea es medido de acuerdo a ello.

El objetivo es asignar aviones a las puertas de salida de tal manera que la asignación sea físicamente factible y conveniente, para lo cual deberán estar disponibles en sus respectivos tiempos de arribo, optimizando a la par cierto número de procesos (minimización de trabajo para el personal aéreo, remolque de aviones y retraso de vuelos).

En este escenario, se consideran a las puertas como fuentes o recursos, mientras que la manipulación y el servicio de los aviones son las tareas. El arribo de un avión en una puerta dada representa el tiempo inicial de una tarea, y su partida representa el tiempo de conclusión.

2.4.2 Planificación en la Computación

La CPU es el lugar donde se concentra la gran mayoría de las tareas que realiza el computador; además de planificar los procesos internos –propios del funcionamiento del computador– se deben atender las peticiones de mayor prioridad que va ejecutando el usuario.

A continuación se describe un tipo de planificación de Tareas en un CPU [32].

Una de las funciones de un sistema operativo computacional multitareas es planificar el tiempo que el CPU dedica a los diversos programas a ejecutar.
Para evitar el hecho de que pequeñas tareas permanezcan en el sistema por un largo periodo de tiempo esperando por tareas más extensas y de más alta prioridad, el sistema operativo recorta dichas tareas en pequeños segmentos, y en seguida los rota en el CPU, para que así, en cualquier intervalo de espacio dado, éste invierta cierta cantidad de tiempo en cada tarea.

La planificación descrita comprende las siguientes restricciones:

- Los tiempos de procesamiento exactos no son conocidos con anticipación. Pero, la distribución de tiempos de procesamiento aleatorio puede ser conocida con antelación, incluyendo sus valores esperados y sus variantes.

- Cada tarea tiene usualmente cierto nivel de prioridad (el sistema operativo originalmente permite a operadores y usuarios especificar el nivel de prioridad o medida de cada tarea)

- Una interrupción en el procesamiento de una tarea es frecuentemente debido a la presencia de otra de mayor prioridad (preemption), la cual necesita el uso del procesador para su ejecución. Está claro que la política óptima en tal ámbito hace necesario el uso de prioridades.

El objetivo es minimizar la suma estimada de los tiempos que demoran la realización de todas las tareas planificadas.

2.4.3 Planificación en la Gestión de Proyectos

El retraso de proyectos es un problema clásico; en una época en que las empresas intentan reducir el tiempo de desarrollo de sus productos para llegar al mercado lo antes posible, los problemas de gestión de proyecto y soporte técnico originan retrasos críticos.

A menudo la gestión de proyecto se ejecuta manualmente, con ayuda de metodologías tales como CPM (método de la ruta crítica) o PERT (evaluación del programa y evaluación técnica), ambas basadas en la búsqueda de una ruta a través de una malla de actividades.

Aunque las metodologías CPM y PERT son técnicas estándar de gestión de proyectos, se presenta el inconveniente de: sólo indicar una ruta crítica teórica (un retraso en una actividad de la ruta crítica origina el retraso de todo el proyecto).

Cuando la planificación se realiza usando CPM y PERT, éstos dan mayor prioridad a la complejidad del proceso y a las interrelaciones entre las diferentes etapas, que, hacia los retardos que pudiera ocurrir en la obtención de materias primas, o en problemas en soporte técnicos. Cuando aparecen estos problemas, se tiende a alterar la producción para evitar pérdidas.

Estos problemas los podría prever, con un sistema que mantenga stocks, información de inventarios de proveedores en base de datos, disponibilidad y performance de máquinas. [31].

2.4.4 Planificación en la Industrial

En la industria existen distintas sub-áreas donde es necesario una correcta planificación [27]:

- Planificación en las necesidades de materias (Logística).
- Planificación de capacidades y cargas (Transporte).
- Análisis de costos de producción (Contabilidad).
- Control de actividades en la planta de producción (Producción).

Una correcta planificación en estas áreas conlleva a optimizar el uso de los recursos de la fábrica mediante un diseño eficiente del programa y secuencia de fabricación; y, con el objetivo de maximizar la productividad, minimizar el inventario y los costos de operación.

La presente tesis de investigación se centra en la problemática de producción, específicamente en la *planificación industrial*, por tal razón se detallará el proceso de planificación dentro de dicho marco, para mostrar las mejoras existentes en las principales sub- áreas.

Planificación de procesos

La presente es el área de planificación industrial más importante, en vista que de ella depende la debida asignación, tanto del personal como de máquinas en busca de una óptima ejecución de la producción. Un error en la planificación conlleva a:

- La demora en realizar tareas, sea porque la máquina esté aún ocupada con otra tarea, a sabiendas de que esta última tarea pudo haber sido ejecutada en otra máquina que en esos momentos se encontrase desocupada.
- Esperar la ejecución de la tarea precedente (en caso existiera) pudiendo haber sido ésta ejecutada con anticipación y junto con otras, de esta forma se evitaría máquinas desocupadas.
- Sobrecargar con tareas a una máquina en particular, dejando a otras desocupadas, produciendo un exceso de tiempo muerto.
- Generar un sobre almacenamiento de stock que conlleva a frenar las máquinas hasta agotar el exceso de producción (generando tiempos muertos) o, caso contrario, no producir lo suficiente en un periodo de alza en la demanda y, así, no satisfacer las demandas del mercado

La Investigación operativa ha aportado soluciones para una correcta *planificación de procesos,* especialmente mediante estrategias de Teoría de colas y programación matemática (programación lineal); sin embargo, como dichas soluciones pertenecen al tipo de soluciones exactas, heredan las desventajas mencionadas en el capitulo anterior.

La Inteligencia artificial aporta sus técnicas heurísticas y meta heurísticas para la optimización de los procesos así como la automatización de los mismos, ya que el modelo productivo encaja perfectamente en el modelo de planificación *Scheduling* [1]:

- Tarea: operaciones, acciones y actividades en la línea de producción.
- Máquina: maquinarias y herramientas involucradas en el proceso.
- Objetivo: minimización del tiempo de trabajo, de costos, reducción de tiempos muertos e incluso combinando los anteriores.

Programación de Autómatas y robots

Hoy en día la tendencia en las fábricas de producción es el reemplazar la mano de obra por la presencia de robots y autómatas. La razón para este cambio es:

- El ahorro en costos asociados al sueldo y a seguros según sea el caso de la tarea a realizar.
- La rapidez y coordinación entre autómatas.
- La precisión que aportan los sistemas electrónicos.

Programar a un robots consiste en dotarlo de algoritmos numéricos, programación en paralelo, por tal razón el uso de meta heurísticas del tipo GRASP [25,26] es importante, debido a que éstas permiten el mayor aprovechamiento y cuidado de los mismos, al no exigir resultados exactos sin dejar de lado la precisión que ello demanda.

Entre las tareas que realizan los autómatas se incluyen: la manipulación, soldadura, extracción, pulido de las piezas, entre otras; por tal razón el conocimiento de la estructura interna y externa y de la geometría del objetivo a manipular es importante. En la actualidad software del tipo CAD y CAM son los más usados para obtener la información de la estructura del objeto [1,27].

Planificación en los stocks y mantenimiento de nivel de inventarios

Cuando existe un exceso de producción ante una baja demanda o ineficiente planificación de procesos –como se mencionó anteriormente– se genera un exceso de stock. Una correcta planificación del mismo permitirá entonces [27]:

- Alcanzar la máxima eficiencia en el espacio así como en el uso de instalaciones para el almacenamiento y/o manejo de materiales.
- La racionalización del flujo de producción.
- La reducción del capital inmovilizado en materias primas, productos finales y en el curso de fabricación.

Para la corrección de estos defectos se utilizan técnicas referidas a la Investigación Operativa (programación matemática) [24], meta heurísticas, así como Sistemas del tipo JIT (Just in time) cuyo objetivo principal es adecuar el nivel de producción al nivel de demanda, reduciendo tiempos de servicio y desperdicios, pero cuya aplicación exige una gran coordinación con los proveedores y supone cumplir plazos y niveles de calidad [27].

Industria del envasado

Al establecer que la elaboración del envase es independiente del producto en cuanto tal, la planificación de las tareas que la compone no dependerá de la elaboración del contenido, por tal razón ambas se pueden planificar en tiempos distintos o incluso optando por una tercerización de las mismas.

Considerando una empresa que produce bolsas de papel para cemento, carbón, comida de perro y demás [32]; desde la llegada de la materia prima (rollos de papel), el proceso de producción consta de tres etapas principales: impresión del logo, pegado de los lados de las

bolsas y el coser uno o ambos extremos. Cada una de estas tareas implica dificultades para su correcta ejecución:

- Desbalance en la cantidad de máquinas para una u otra tarea.
- Variación de las velocidades empleadas, siendo no necesariamente idénticas entre sí.
- Cantidad de colores capaces de imprimir o tamaño de bolsa a manejar.
- Fecha comprometida para la entrega del pedido, entre otras.

Para el caso de los envases de lata se considera desde la llegada de láminas y sus posteriores procesos a saber: cortado, soldado, etiquetado.
Para el embotellado, se estima desde la llegada de moldes de plástico y sus procesos consecutivos: modelamiento, horneado, etiquetado [1].

Se concluye que la industria del envasado, es una de las áreas más importantes dentro de la producción, y que una planificación de sus tareas implicadas es fundamental para la fábrica.

Industria del ensamblado

Dentro de la industria del ensamblado, los robots son parte fundamental, debido a que son éstos los que prácticamente representan la totalidad de mano de obra empleada [34].

La planificación consiste en una correcta sincronización y desplazamiento entre robots con el fin de minimizar tanto el tiempo como los defectos del ensamblaje, así como aumentar la producción y evitar el consumo del combustible durante el periodo de tiempo inoperante [29]
Este tipo de industria consta del ensamblado tanto de piezas grandes (componentes de volquetes, máquinas pesadas y demás), como de pequeñas (componentes de computadoras y artefactos**,** entre otros) [30].

2.5 Variantes particulares del Flow shop scheduling

Las variantes del FSS se derivan de la combinación de los componentes del scheduling, tareas (dependientes o independientes) y máquinas (homogéneas y heterogéneas).

2.5.1 Máquinas idénticas y tareas independientes

Tal y como se menciona en el capítulo anterior, el término referido *maquinas idénticas* no indica precisamente que éstas se demoren el mismo tiempo en realizar un trabajo determinado, sino que existe entre ellas una diferencia menor, la cual puede ser tomada como "tiempos iguales" al realizar un trabajo. De esta manera se podrá *escoger* dentro de todas las máquinas la que demore menos o, la que lleve un menor tiempo *acumulado* de trabajo (con menos sobrecarga de trabajo).

Las tareas para esta variante no dependen de la ejecución de otras, por lo que existe plena libertad de escoger cualquiera de ellas, siendo factible la elección de la tarea cuyo tiempo de ejecución sea menor o, aquella que no ha sido asignada en muchas máquinas.

Esta variante resulta ser interesante en vista de que se podrían proponer dos procesos, uno para escoger la tarea que tiene el menor tiempo acumulado de ejecución –dentro de todas – y otro con el propósito de elegir la máquina que tenga el menor tiempo acumulado de trabajo; tomando un enfoque más heurístico que voraz, lo cual se complica cuando se trata de "*n*" tareas y "*m*" máquinas.
Ésta es la variante a desarrollar en el presente trabajo.

2.5.2 Máquinas idénticas y tareas dependientes

Similarmente, en esta variante se podrá escoger la máquina que demore menos tiempo en realizar una tarea, sin embargo, para este caso, las tareas dependen de la ejecución de otras, haciendo necesaria la ejecución de la tarea de mayor precedencia. Por consiguiente, el número de máquinas con la capacidad de resolver dicha tarea se hace menor, siendo innecesario un proceso de selección.

2.5.3 Máquinas diferentes y tareas independientes

En la presente variante se elige la tarea que demore menos tiempo y luego se ejecuta la tarea en la máquina que toma menor tiempo para ello. Por tanto, no existen muchas alternativas ni tampoco procesos de selección.

2.5.4 Máquinas diferentes y tareas dependientes

Es posiblemente la variante más fácil de desarrollar debido a que no tiene opción de elegir: se deberá ejecutar la tarea que tenga mayor precedencia y la máquina que realice dicha tarea en el menor tiempo.

2.6 Métodos existentes para resolver el problema del Scheduling

Existen 2 métodos para resolver el problema del scheduling: métodos exactos y aproximados.

- Los métodos exactos procuran obtener una solución exacta.
- Los métodos aproximados, en cambio, pretenden obtener la solución más cercana al óptimo, mediante la aplicación de técnicas heurísticas y meta heurísticas.

2.6.1 Métodos exactos

Estos métodos no resuelven las variantes del scheduling mencionadas anteriormente (Job scheduling, Flow scheduling, Open flow scheduling y demás); mas bien, enfocan sus fuerzas en variantes que combinan niveles de demanda, órdenes de venta, entre otras, o en

planteamientos tradicionales del scheduling (sin tomar en cuenta las propiedades de las máquinas: homogéneas, heterogéneas, y tareas: dependientes o independientes).

Entre los grupos más representativos de métodos exactos se tienen aquellos que usan técnicas de investigación de operaciones, mientras que otros –que son algoritmos– aplican técnicas de inteligencia artificial.

2.6.1.1 Métodos basados en Investigación de operaciones

Las soluciones pertenecientes a este grupo [33] se centran en variar –aumentar y/o disminuir– la cantidad de máquinas y tareas hasta que el problema no se torne en uno del tipo NP-difícil:

Siendo las tareas homogéneas y las máquinas iguales:

- "n" trabajos asignados a una máquina; este problema es solucionado mediante un algoritmo, derivado de un teorema, bajo el cual se ordenan ascendentemente los tiempos de las máquinas, extrayendo de dicha ordenación el tiempo más corto.
- "n" trabajos asignados a 2 máquinas; esta variante es solucionada por el algoritmo de Jonson (S.M. Johnson 1954).
- "n" trabajos asignados a 3 máquinas; problema también resuelto por el algoritmo de Johnson, siempre y cuando la siguiente condición se cumpla:

  - Que la tarea que toma menor tiempo en ejecutarse, máquina #1 ó máquina #3, sea mayor o igual a la tarea que demora más tiempo, asignada ala máquina #2. Vale decir que el segundo proceso está completamente dominado por el primer o tercer proceso.

  Si la condición anterior no se cumple, el problema puede ser resuelto por otros algoritmos como Ignall y Schrage (Edward Ignall and Linus Schrage 1965).

- A partir de la variante; "n" trabajos para tres máquinas, el problema se transforma en uno del tipo NP-dificil.

- Dos trabajos en "m" máquinas; desarrollado por Akers y Friedman en 1955, mejorado posteriormente por Hardgrave y Nemhauser en 1963. Es esencialmente una solución gráfica en donde los tiempos son graficados en dos ejes coordinados (Eje "X" trabajo1 y Eje "Y" trabajo 2). Variantes superiores, igualmente se tornan en NP-difíciles.

El otro grupo de métodos está conformado por los diseñados y desarrollados a medida, dentro de los principales se mencionarán: Tree Traversals , GPS y Graphplan.

2.6.1.2 Tree Traversals

Las primeras soluciones para el problema del scheduling fueron búsquedas del área-situación sea backward (hacia atrás), forward (hacia adelante) o compuesto backward/forward en árboles (estructuras de datos). Una búsqueda era la aproximación más directa hacia una solución, pues trataba todas las posibles soluciones hasta que ésta produjera la deseada. En

la *búsqueda forward* el método empieza desde la *situación inicial,* usando a: los operadores para crear un árbol que *presente la situación-espacio* como nodo (regulador, cruce) y a los **operadores**, transformando una *situación* en otra en sus ramas (dominio, ramificación)

El árbol se ve expandido continuamente hasta que el solucionador encuentre un nodo que sea la *situación meta.* Lo opuesto sucede en la búsqueda backward; el solucionador inicia desde la *situación meta* e intenta transformarla hacia la situación inicial. Mientras tanto, en la búsqueda híbrida backward/forward, el solucionador emplea ambos métodos al mismo tiempo, en busca de una situación en donde ambas técnicas se encuentren [28]

2.6.1.3 GPS

GPS [28] usa sentencias en cálculos de predicado de primer orden en la resolución y análisis de la relación entre el estado actual y el estado meta para la solución de problemas.
El análisis means-ends (MEA) detecta la diferencia entre la situación actual y la situación meta, y una vez encontrado el operador que pueda reducirla, éste es identificado y aplicado. Si el operador no es directamente aplicable (apropiado) para la situación actual, entonces el sub-problema de la creación de la situación apta para su aplicación, deberá ser resuelto primero. GPS continúa de esta forma hasta reducir el problema inicial a un conjunto de subproblemas elementales que puedan ser solucionados por la directa aplicación de los operadores.

2.6.1.4 Graphplan

Es uno de los Métodos existentes más sencillos cuyo objetivo es la descomposición de un problema en forma de *árbol*, que además de organizar su recorrido con el propósito de determinar un orden adecuado (respuesta), las fases que implican los procesos de programación de máquinas, son a saber, diferenciadas:

Expansión: implica la extensión del plan en su representación gráfica hasta lograr condiciones para la existencia de un nivel.

Extracción de la solución: ejecuta un proceso de encadenamiento hacia atrás en busca de la programación que se siguió para la generación de nodos. De no conseguirlo, se genera un nuevo proceso de expansión, hasta que el lote de tareas se complete del todo, por lo cual se considera a ésta, una búsqueda heurística [1,28].

2.6.2 Métodos heurísticos y meta heurísticas planteados por variantes

Considerando al scheduling como un conjunto de N tareas y de M máquinas, cuyo objetivo principal es la ***minimización del tiempo de procesamiento o makespan.***

Bajo esta clase de problemas, las heurísticas buscan asumir una estrategia de ordenamiento de tareas y/o máquinas en listas, que precede al proceso de asignación. Para evaluar la calidad de solución adquirida, se establece comparación con un límite inferior de duración (GRASP, o criterio voraz) teniendo en cuenta que el objetivo sea la minimización del tiempo, antes que otros objetivos existentes.

Debido a la calidad de soluciones halladas, hoy en día la mayoría de soluciones encontradas se centran en un criterio voraz para plantear su metodología, muy a pesar de que a partir de este criterio heurístico se puedan generar algoritmos meta heurísticos (llámese GRASP, heurísticas Tabú, algoritmos genéticos).

Considérese el siguiente esquema que organiza los métodos heurísticos y meta heurísticos existentes, los mismos que se avocan a resolver las diversas variantes del problema del scheduling. Se presentan cuatro áreas principales: Job Scheduling, Task Scheduling, Flow Shop Scheduling y sistemas en tiempo real.

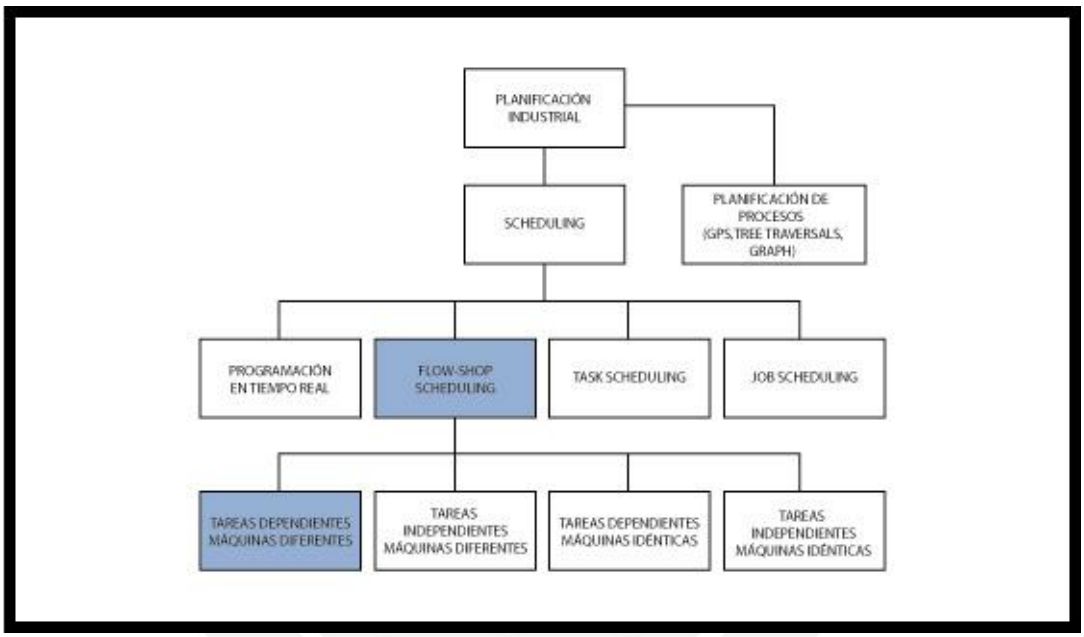


Figura 9 Métodos heurísticos y meta heurísticos para resolver las variantes del scheduling

Asimismo, se ejemplifica con trabajos relevantes a cada variante, –según su relación directa con la planificación de trabajos en líneas de producción–.

2.6.2.1 Métodos heurísticos para resolver el problema del flow-shop scheduling

- Aplicando algoritmos ***GRASP***:

- Feo, Sarathy, McGahan [34] para máquinas simples (Single Machine scheduling) consistía en asignar un costo a la ejecución de cada trabajo. Así el costo s_{ij} era el de ejecutar inmediatamente la tarea j después de la tarea i; luego agregaba el tiempo de ejecución d_j para cada trabajo y el concepto de penalidad por retrasos en la línea p_j. De esta manera**,** se pretendía aplicar las técnicas GRASP para minimizar la suma de las penalidades reduciendo así el tiempo de procesamiento de lote:

$$\sum_{j=1}^{n} s_{[j][j+1]} + \sum_{j=1}^{n} p_{[j]} D_{[j]}$$

- Ríos y Bard [35] plantearon una GRASP con la finalidad de hallar una secuencia de N tareas en un ambiente de flow shop de M máquinas con tiempos de preparación dependientes de la secuencia, que minimice el makespan. Para comparar el nuevo modelo prepararon heurísticas: basadas en el trabajo de Nawaz [67] en éste, se esboza la idea de construir una secuencia factible de ejecución. En cada interacción del algoritmo, existe una secuencia parcial S. Se selecciona una tarea h de la lista de tareas P la cual aún no ha sido programada.

La secuencia parcial S y la tarea h definen una función ávida única: $\varphi(j):\{0,1,\ldots,|S|\}\to\Re$, donde $\varphi(j)$ es el tiempo de terminación de todas las tareas en la nueva secuencia S' resultante de insertar la tarea h justo después de la j-ésima tarea en S, |S| denota la cardinalidad de S y $\Re$ es el conjunto de los números reales. Los resultados fueron favorables para el nuevo GRASP.

- J.P. Pendones, A.Corominas, R.Pastor [55]: proponen una heurística para encontrar el makespan mínimo, en una línea de producción del tipo FSSP, compuesto por m máquinas (celdas robotizadas) las cuales son suministradas por un robot industria y m-1 buffers ubicados entre las celdas. El robot realiza el transporte de las piezas entre máquina y los buffers, siendo las piezas de distinto tamaño. La heurística propuesta consiste en dos partes: la primera, determinar una secuencia de tareas a realizar, ello mediante una meta heuristica GRASP; la segunda, buscar una secuencia de movimientos del robot, mediante procedimientos basados en la heurística Branch and bound .

- El mismo problema con piezas distintas para tres y cuatro máquinas es resuelto mediante Heurísticas por Kamoun [68].

- Problemas de piezas iguales sin buffers para el caso de minimización del tiempo de ciclo Ct, son resueltos por Crama y van de Klundert [58] y Sethi et al. [69].

- El problema del makespan con buffers finitos es resuelto de forma exacta en Hitomi y Yoshimura [70], donde se asumen tiempos de carga y descarga dependientes de las piezas.

- Usando heurísticas de ***Búsqueda Tabú***:

- Acero y Torres [36]: ambos trabajaron sobre el marco de líneas de producción flexible. Representaron una solución factible mediante un vector de dimensiones 2 x N x M componentes (siendo N el número de trabajos y M el número de etapas) Las primeras N componentes del vector (primera fila) representan el número de la máquina por el cual el primer trabajo ha de pasar en cada etapa.

Así por ejemplo: 1, 3, 2, 3, 2, 1, 2. indica que: el trabajo 1 se procesará en la máquina n° 1 para la primera etapa, para la segunda etapa (el trabajo 1) pasará por la máquina n° 3, para la tercera etapa pasará por la máquina n° 2, y así sucesivamente, repitiéndose para cada uno de

los trabajos. Los sucesivos N x M componentes representarán –bajo los mismos parámetros de ordenamiento– el orden bajo el cual se procesarán los trabajos en cada etapa. Por ejemplo: la secuencia 1,6, 51, 10... indica que en la primera etapa se ejecutará el trabajo 1, luego el trabajo 6, enseguida el 51, y así sucesivamente.

- Usando ***Algoritmos Voraces***:

-Chand y Scheneeberger [37]: trabajos para líneas con máquinas simples y trabajos sin demoras.

- Otros Algoritmos:

- F. Chin, L.Tsai [10]: presentan una heurística hecha a medida, para solucionar una variante del flow shop scheduling llamada J-maximal flow shop scheduling. En dicha variante, la J-ésima tarea en cada trabajo posee un tiempo de ejecución más prolongado que las demás; el algoritmo aplica técnicas con el propósito de planificar mejor la tarea en mención y aprovechar así los recursos en su ejecución, evitando la exclusividad de máquinas.

- M.A. Salazar, R.Rios [56]: presentan una heurística para encontrar una secuencia de n tareas, m máquinas en un ambiente flow shop para minimizar el número de tareas que se entrega con retraso. La solución presentada está basada en el algoritmo de Moore 1968 (genera secuencias óptimas para n tareas en una máquina). Dicho algoritmo se evalúa y se compara computacionalmente con un método que genera secuencias tareas-máquinas sin tomar en cuenta la estructura del problema bajo 2 escenarios distintos: fecha de entrega estricta y no estricta.

2.6.2.2 Métodos heurísticos para resolver el problema del Task scheduling

- Aplicación de ***Algoritmos Voraces****:*

- Campello, Maculan [38]: se aplican Algoritmos Voraces en máquinas idénticas. Su planteamiento parte de una definición del problema como uno de programación discreta, que al ser de tipo NP-difícil, es posible.

- Tupia [1]: aplicación de Algoritmos Voraces. Se presenta el caso de las máquinas diferentes y tareas independientes, considerando tiempos distintos de ejecución para cada máquina (adaptado del modelo Campillo y Maculan), surge el concepto- matriz de T_{ij}, tiempo que tarda la tarea i-ésima en ser ejecutada por la máquina j-exima.

- Aplicación de ***Algoritmos GRASP****:*

- Tupia [1]: Frente al caso de máquinas diferentes y tareas independientes, se amplió el criterio voraz del algoritmo anterior aplicando las fases convencionales de la técnica GRASP, lo que

consiguió mejorar aproximadamente al 10% los resultados del algoritmo voraz para instancias incluso mayores a las 12500 variables, vale decir, 250 tareas por 50 máquinas.

- Usando ***Algoritmos Genéticos****:*

- O.Etiler,B.Toklu,M.Atak [52]: desarrollaron una heurística (Algoritmos genéticos) nueva para resolver el problema del FSSP. Dicha heurística genera un juego de parámetros diferentes para los operadores genéticos, lo cual produce el menor makespan de cada generación – realizable para cada población – y que posteriormente se hereda para la siguiente generación.

- Usando ***Búsqueda Tabú***:

- Wilder, Hert presentan una heurística llamada SPIRIT compuesta por 2 fases, la primera obtiene una solución inicial al FSSP con un método muy similar al que resuelve el inconveniente denominado Traveling salesman problem, y posteriormente la afina aplicando técnicas de búsqueda tabú

- Otros Algoritmos:

- Voon-Yee Vee and Wen-Jing Hsu Heurística [17]: diseñaron un algoritmo para solucionar el problema del Task Scheduling empleando las potencias del Click para las tareas paralelas (Click es un lenguaje de programación multitarea, que a su vez es una extensión del Lenguaje C, con constructores para control paralelo). No obstante, existen relaciones de precedencia que no pueden ser expresados puntualmente en el Clic; sin embargo, ésta limitación no es impedimento para que el Click sea aprovechado en el task scheduling con mayor performance que otros lenguajes multitarea.

2.6.2.3 Métodos heurísticos para resolver el problema del Job Scheduling

- Aplicación del método **Ramificación y Acotación**:

- Bard y Feo [41]: se presenta el conocido método de búsqueda de grafos dirigidos de ramificación y acotación que lleva el mismo nombre, con éste se recorre un árbol cuyas intersecciones (nodos) son secuencias de operaciones en líneas de manufacturas. Siendo los entornos prácticamente invariables, así como libres de penalidades**,** ya sea por retrasos o por alternación de una máquina a otra. De ello se sigue, la aplicación directa de un método de barrido de árboles en la selección de instrumentos y trayectorias.

- Aplicación de Algoritmos **GRASP**:

- Binato, Hery y Resende [42]: su aplicación incorpora dos nuevos conceptos en el desarrollo de una GRASP convencional para el JSP. Un procedimiento de intensificación estratégica para generar soluciones candidatas (nueva distribución probabilística) y una técnica POP (proximote optimality principle) ambas dentro de la fase de construcción del GRASP

- Aplicación de **Algoritmos Genéticos**:

- Goncalves, De Magalahes y Resende [43]: dada la representación de cromosomas, bajo medidas aleatorias; se pretende comparar y asemejar el comportamiento de un entorno de trabajo variable, al de una línea de producción flexible. En tanto, la investigación de otro teórico Davis [46] se basa en la división de las tareas en cromosomas muy pequeños, generando así las poblaciones más variadas posibles.

- Aplicación de la **Búsqueda Tabú**:

- Taillard [44]: se aplica la Búsqueda Tabú, considerando el movimiento de una candidata a operación i-ésima –programada en una máquina, hacia otra– como vecindad para la aplicación de listas Tabú. De ahí que se lograra descomponer todo el proceso de programación de un lote de operaciones, en una serie de pasos**,** considerando a la Lista Tabú como matriz (formada por operaciones y máquinas respectivas de ejecución).

- Chambers y Barnes [45] consideran las denominadas *rutas* flexibles, como rutas que puede operar más de una clase de operaciones, lo cual amplia la definición JSP dada hasta el momento.

- Subramani [48] aplica por su parte, los conceptos de JSP sobre las técnicas de Grid (Grid Computing) en la programación de tareas de súper computadoras (estrategias de programación de varios trabajos recurrentes y simultáneos vistos en la atención de los procesadores).

- Otros Algoritmos:

- Naveen Garg,Jain,Swamy [51]: presentan un algoritmo de aproximación aleatorio para el Flow Shop Scheduling para cuando el número de máquinas es fija (estable). El algoritmo se basa en el redondeo de la solución de una formulación al problema del Flow Shop Scheduling, añadiéndole ciertas restricciones adicionales, las cuales hacen del redondeo un esquema posible. El makespan devuelto por dicho algoritmo está ubicado dentro de un rango cercano al tiempo del óptimo makespan y posee un término aditivo, –muestra un trueque entre los factores aditivos y multiplicativos–. El factor aditivo es mejor que el factor aplicado en el algoritmo de Sebastián Janov 1986, y el factor multiplicativo es mejor que aquel algoritmo de D.B. Shmoys 1994.

- M. I. Alfonso, Federico Barber [57]: muchos problemas de JSSP fueron resueltos con técnicas CSP *(Constraint Satisfaction Problem, 1*977) combinándolo con una heurística para intentar reducir el espacio de búsqueda y ordenar las instancias de variables y valores en sus respectivos dominios. Con la adición de la heurística se persigue limitar la complejidad experimental del problema y número de *bactrackings* necesarios para obtener una solución.

2.6.2.4 Métodos heurísticos para resolver el problema de la programación en tiempo real

- **Planificación basada en prioridades**

  En este tipo de planificación las prioridades son asignadas por el mismo algoritmo de planificación. La tarea con mayor prioridad se ejecuta en cualquier instante.

- **Planificación round robin con y sin prioridades**

  En este tipo de planificación [50] el usuario asigna prioridades a las tareas (fuera de línea). Las tareas se organizan por prioridad y cada una posee un tiempo asignado, denominado *quantum.*

- **Algoritmo Dual Priority y sus adaptaciones**

  Este algoritmo [53] fue creado para asignar tareas dinámicas en entornos multiprocesador con memoria compartida.

- **Planificación basada en el reloj (basada en el tiempo)**

  Previo a la aplicación de este plan [49], se requieren los tiempos de inicio y de cómputo de cada tarea. El plan de ejecución se calcula fuera de línea y ésta se plasma en una tabla. Este plan no es concurrente.

2.7 Algunas aplicaciones existentes

En esta sección se analizarán las principales aplicaciones existentes en el mercado, se nombrará el método usado y el problema que persiguen resolver:

2.7.1 GARI

Creado por Descote [71]. 1981: Es un sistema básico de conocimiento para la planificación de procesos. Su dominio está restringido a la industria del cortado de metales. GARI es implementado en el lenguaje MACLISP, operando bajo el sistema de MULTICS, y cuyo conocimiento se halla representado por las reglas de producción.
GARI enfatiza la resolución del conflicto porque los desarrolladores creen que el conocimiento manufacturero es descrito mejor como varios y diversos ejemplos sugeridos de mayor peso antes que un pequeño set de restricciones a ser satisfechas. (en GARI, los conflictos son resueltos seleccionando el ejemplo sugerido de mayor peso).
Múltiples esquemas pueden ser generados regresando al punto de conflicto y reintroduciendo un ejemplo sugerido que ha sido rechazado, para luego proseguir con la planificación. Dado que la planificación es dependiente de los pesos, diferentes medidas pueden producir diversos esquemas.

2.7.2 SIPP

Creado por Nau and Chan [72], 1985: Es un sistema de conocimiento básico para la planificación de procesos generativos de las partes tratadas a máquina. El sistema SIPP usa una buena primera estrategia basada en Branch and Bound y produce menor costo en los planes de proceso, apoyado en criterios específicos del usuario. El sistema está programado en PROLOG.

2.7.3 TOM

Creado por Matsushima [73], 1988: otro sistema de reglamento básico es TOM, el cual usa un mecanismo de encadenamiento hacia atrás con la finalidad de generar una secuencia de máquina. El conflicto entre las reglas es resuelto aplicando métodos heurísticos, tales como tomar la primera regla encontrada, la última y la más utilizada. El programa está escrito en PASCAL.

2.7.4 ESHEL

Es un sistema de regla básica para la automática generación de procesos profundos de delineado en el dominio de formación. Las reglas están organizadas en una estructura jerárquica. La activación de tales reglas es controlada por un “Manager rule” que escanea las reglas-prueba de todos los candidatos y activa aquellas cuyo dominio contiene el problema. El sistema está programado en PROLOG en un entorno UNIX [28].

2.7.5 EXCAP

Creado por Davis [74], 1984: es un sistema experto para los planes de procesos generadores para el tratamiento a máquina de los componentes de turno (de rotación). Es un sistema de regla básica y usa el mecanismo de encadenamiento hacia atrás. Desarrolla un “árbol” de posibles secuencias de operaciones. Cada operación disponible es una meta hipotética en la red de inferencia y EXCAP busca aprobarla o desaprobarla. Puede generar múltiples secuencias y tiene la habilidad de proveer explicaciones. El sistema está programado en PASCAL.

2.7.6 CUTTECH

Creado por Barkocy [75], 1984: este sistema incorpora técnicas de inteligencia artificial. En él, las reglas son almacenadas en tablas de decisión y algoritmos, luego son aplicadas en orden descendente de importancia con el propósito de ubicar una lista de herramientas desde la más a la menos preferida. El énfasis de este sistema reside en la implementación y es de gran naturaleza práctica.

2.7.7 TURBO-CAAP

Creado por Wang [78], 1988: es un sistema de conocimiento base para el proceso de planificación. Este sistema combina ambos: el frame-based y esquemas de representación de sistemas de producción-base. El problema de solución es estructurado en diferentes estratos (capas) y la arquitectura de capas es implementada en un microcomputador bajo el lenguaje de programación PROLOG.

2.7.8 TOLTEC

Creado por Tsatoulis y Kashyap [76], 1988: es un sistema de planificación de procesos cuyo aprendizaje se toma de la experiencia. Éste aplica el concepto de estructuras de memoria dinámica desarrollada para la comprensión del lenguaje natural. Para almacenar el conocimiento, este sistema usa diversas estructuras de organización de memoria en diversos niveles del detalle conceptual. Dichas estructuras se encuentran de la siguiente forma: desde la más alta a la más baja en abstracción: meta-POP (paquetes de organización de memoria), MOP, Cuadros, Scripts.

La Planificación en TOLTEC procede de una solución básica (esqueleto) hacia otra más detallada, mediante una descomposición jerárquica de la solución. Las restricciones y memorias de fallos previos son utilizadas para guiar al planificador en la selección de apropiadas soluciones, demostrables en la memoria dinámica. TOLTEC emplea su TOPs (paquetes de organización temática) para aprender de sus errores pasados. TOLTEC puede o bien ser guiada por el usuario –en la selección de soluciones alternadas– o puede automáticamente mejorar su performance evitando su solución y su profundo conocimiento (conflictos de restricción) En todos estos casos, TOLTEC reorganiza su memoria dinámica para evitar la repetición de los mismos errores, o para una mejor correspondencia con las preferencias de solución del usuario. El sistema de planificación de procesos ha sido integrado con un módulo CAD [28].

2.7.9 ISIS II

Fue desarrollado por la Universidad de Carrogie Mellon, Westing House y la Fuerza Aérea Norteamericana, para planta de ensamblaje de turbinas a vapor de helicópteros que siguen una Planificación Flow Shop Scheduling. El sistema se basa en redes neuronales y el método empleado es el de una búsqueda heurística. El conocimiento radica en las restricciones y en la experiencia del planificador (scheduler). El conocimiento del experto, es empleado para *relajar* las restricciones y las soluciones límites (fronteras) reduciendo así el espectro de soluciones posibles, haciéndolo más efectivo. Con ello, se obtuvieron resultados radicales, mejorando la velocidad de producción en un 66% prácticamente [28].

2.7.10 OPGEN

Fue desarrollado por la empresa Hazetline Comp. Está diseñado para la organización tanto del diseño como del ensamblaje de tarjetas de circuitos integrados para computadoras. El método usado es un algoritmo híbrido entre encadenamiento progresivo y regresivo. Por lo expuesto, OPGEN está desarrollado para organizar:

- La línea de producción.
- La disposición de tarjetas.
- La instalación del integrado.
- La impresión de pastillas.

Antes de optar por usar OPGEN, una empresa de estas características se demoraba veinte horas en disponer las listas de tareas entre:

- Dibujar los tableros donde colocar los integrados.
- Uso de aparatos e insumos.
- Disponer las áreas de trabajo (operarios y máquinas).

Una vez implementado OPGEN, este tiempo se redujo a noventa segundos junto con otra solución de contingencia que demoraba cuatrocientos treinta y cuatro segundos [28,31].

2.7.11 BEN ARICH

Describe una investigación experimental en la ruta de *jobs* en una producción automatizada y en las instalaciones de ensamblaje, usando un sistema de conocimiento base. El objetivo del sistema es utilizar toda la data disponible en una celda informatizada de manufacturación, crear un buen mecanismo de control para supervisar el sistema, y generar soluciones de tiempo real a problemas que surgen durante el *run time* del sistema. El sistema consiste en dos tipos de conocimiento: el conocimiento de producción como reglas y el conocimiento de procedimiento (procesal). Una data base es usada para almacenar las condiciones del sistema y consta de componentes estáticos y dinámicos. El conflicto entre las reglas seleccionadas es resuelto mediante el ordenamiento de las mismas. El sistema está escrito en PROLOG bajo un sistema UNIX y fue probado en un entorno simulado [28].

2.7.12 FADES

Creado por Fisher [77], 1984: es un centro de planificación y de diseño de sistema. El conocimiento es representado en forma de reglas implementadas en procedimientos (procesos) lógicos y predicados de primer orden. El conocimiento base es parte de la estación de tecnología del trabajo. Puede ser usado como un pre microprocesador para el esquema de algoritmos existentes, tales como CORELAP.

2.8 El problema del Flow shop scheduling en máquinas semejantes y tareas independientes

2.8.1 Consideraciones generales

El problema de línea de flujo FSSP consiste de un conjunto de *n* tareas que deben ser procesadas a través de un conjunto de *m* máquinas cada una. Cada tarea tiene el mismo ruteo tecnológico a través de las máquinas. Es decir, cada tarea debe procesarse primero en la máquina 1, luego en la 2, y así hasta la máquina *m*. Ninguna pasa a una operación posterior para luego devolverse [51,16].

Este problema, es considerado del tipo NP-difícil para instancias mayores iguales a 3 máquinas [59], ya que el número de combinaciones entre maquinas y tareas es excesivamente grande, haciendo intratable poder trabajarlas exactamente, por tal razón los algoritmos tradicionales quedan de lado para optar por heurísticas o meta heurísticas y así

poder encontrar soluciones aproximadas de alta calidad y con tiempos considerables en obtener su solución.

Los componentes fundamentales en un problema de Flow shop Scheduling son: máquinas, tareas, tareas previas, buffers y matriz de tiempo de ejecución

- Las Máquinas pueden ser de 2 tipos: homogéneas o heterogéneas, refiriéndose al tiempo que demora ellas en realizar una tarea. .

- Las Tareas también son de 2 clases con respecto a la precedencia que ellas presentan dentro del proceso productivo: dependientes o independientes. Si para realizar una tarea X previamente se necesita terminar de ejecutar otra tarea Y, se dice que la tarea X es *dependiente* de la tarea Y. En caso no existir dicha dependencia se dice que son tareas independientes.

  Ambas características, homogeneidad y dependencia, permiten al algoritmo que lo resuelva poder asignar mejor los recursos para equiparar los mismos y poder mantener ocupadas todos las máquinas:

- Las tareas previas son aquellas tareas adicionales que se realizan dentro de los buffers y cuyo objetivo es preparar el bien, para que el mismo esté listo y sea procesado por la siguiente máquina siguiendo el ruteo fijado.

- Estas tareas pueden ser: limpieza de la máquina siguiente, limpieza del bien antes de ser procesado, cambios de herramienta usada, calentamiento de la siguiente máquina, entre otras.

- Los tiempo que demoran estas tareas previas, pueden o no, ser añadidos al tiempo de ejecución de las tareas por la maquina asignada.

- Los Buffers son espacios limitados entre máquinas, donde se coloca temporalmente el bien, en espera de la ejecución de la tarea precedente (ahí eventualmente se realiza tareas previas).

  Se considera el primer buffer de capacidad ilimitada, pudiendo o no variar esta característica para los siguientes buffers:

- Algunos autores [60,61] han considerado *buffers* finitos entre máquinas heterogéneas, concluyendo que afecta a la performance del FSSP, disminuyéndolo rápidamente con el incremento del tamaño del buffer.

- La matriz de tiempos de ejecución es una matriz de tamaño m * n donde se muestran los tiempos de ejecución de las tareas en cada maquina donde puede ser ejecutada. Esta información es proporcionada por los jefes de planta o ingenieros industriales, los cuales son las personas que conocen el comportamiento de las maquinas y el tiempo que demanda ejecutarla en las mismas. Cuando esa información no es conocida se suele

llenarla con valores aleatorios. Esta información es imprescindible tenerla ya que de manera contraria no se podría continuar con el FSSP [55].

El objetivo de solucionar el FSSP es generalmente encontrar una secuencia donde las tareas sean ejecutadas por todas la máquinas en el menor tiempo posible, a esta se le conoce como makespan o C_{max}.

El FSSP puede tener variantes, las cuales son reflejo de la realidad –a veces meramente de investigación– que se presenta en una línea de producción.

Las tareas podrían tener tiempos fijos de entrega, o las maquinas podrían tener asignada prioridades para realizar algunas tareas específicas, ya sea por rendimiento o por antigüedad [16], o se podría fijar tareas previas manuales con tiempos de preparación dependientes antes de cada tarea [6], los buffers podrían ser finitos o infinitos [15], o sin buffers [58], las tareas se pueden dividir en otras subtareas asemejándose al criterio del JSSP [52].entre otros. Las restricciones tecnológicas pueden variar de acuerdo ala variante fijada, se nombrará las restricciones generales:

- Todos los trabajos son disponibles al inicio o sea en el tiempo Cero.
- Una máquina no puede procesar más de una tarea a la vez, ni una tarea puede ser procesada por más de una tarea en el mismo tiempo.
- Los trabajos son realizados sin interrupción.
- La matriz de tiempo de ejecución de cada trabajo sobre cada máquina en cada estación es conocida.
- La línea de producción se dispone de tal forma que los diferentes trabajos van en la misma secuencia de la línea (ninguno pasa a una operación posterior para luego devolverse).
- No aparecerá ninguna máquina adicional una vez iniciada la programación.
- Todas las máquinas están disponibles para ser asignadas a un trabajo, salvo ésta se encuentre ejecutando otra tarea

CAPÍTULO 3

3 Algoritmo propuesto

3.1 Generalidades

En cuanto al comportamiento de los elementos en la investigación, se puede afirmar que las máquinas son de tipo homogéneo y además se agrupan en *familias*. De esta manera cada máquina, que *forma parte de una familia*, se demora prácticamente el mismo tiempo en ejecutar una tarea que otra de su misma familia.

Se denomina ***familia de máquinas*** a aquellos grupos o conjuntos que realizan una misma función dentro del proceso productivo. Por ejemplo: máquinas cortadoras, remalladoras, teñidoras, enchapadoras, y demás. De esta manera una tarea, a lo largo de su recorrido dentro del proceso productivo, escoge entre las máquinas desocupadas de una familia la máquina óptima a ejecutar el proceso.

La selección de la máquina óptima y del trabajo a ejecutarse se realizarán mediante la meta heurísticas GRASP, teniendo como criterio base, el tiempo **acumulado** de trabajo de las máquinas y el tiempo **acumulado de ejecución** de la tarea en una familia.

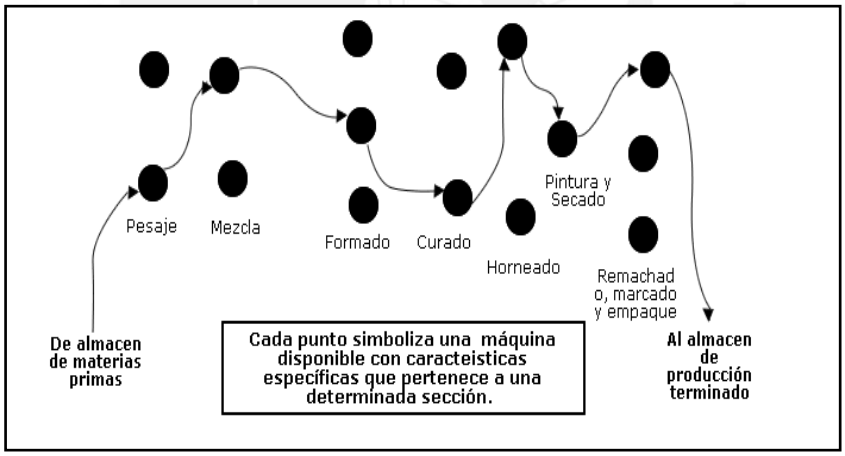


Figura 10 Disposición de las familias de máquinas dentro de una línea de producción FSS

En el gráfico anterior se muestran las máquinas dispuestas en forma vertical para visualizar mejor a la familia a la cual pertenecen.

Para mostrar los tiempos de procesamiento de las tareas en las existentes familias de máquinas, se usará una matriz T. Estos valores incluyen tiempo de manipulación de las tareas durante la ejecución.
El *blocking* [32,40] es un fenómeno que por lo general aparece en las líneas de producción del tipo Flow Shop cuando existen buffers limitados entre máquinas (por ejemplo M_i y M_{i+1}) y éstos se llenan completamente, dando como resultado el que no exista lugar en los buffers

para los trabajos que lleguen después. Esto produce que se extienda el tiempo de procesamiento de la tarea previa en la máquina M_i y así se asigne el tiempo necesario a la siguiente máquina M_{i+1} para que ejecute sus tareas pendientes evitando de esta manera que la tarea espere (fenómeno *nowait* [32,40]) u otras veces –en casos extremos– deteniendo la ejecución de las máquinas previas involucradas.

Estos tiempos de retraso generan penalidades [39,47] (que se ven reflejados en: mayor prioridad para las tareas del buffer, o en la exclusividad del procesador para la maquina afectada) condicionando así la planificación establecida.

Para el presente trabajo de investigación los buffers serán de capacidad ilimitada –como la gran mayoría de investigaciones en scheduling– evitando que se incurra en tiempos de retraso y penalidades.

La naturaleza de las máquinas usadas en esta investigación no tienen la propiedad *breakdowns* [32,40]: es decir, éstas no pueden ser interrumpidas a lo largo de la ejecución de una tarea, ya sea por cambios de componentes mecánicos durante la ejecución o por tareas de mantenimientos. De esta manera no hay un incremento al tiempo fijado para la ejecución.

En el capitulo 2, se hizo mención a los tiempos de preparación de las máquinas dentro del proceso productivo (tareas de calibración de las máquinas si así lo requieren o de limpieza de las mismas para la siguiente tarea) Estos tiempos incurridos los cuales son necesarios en muchos líneas de producción ya están incluidos en los tiempos de ejecución de la tarea en la matriz T. Dichos tiempos son obtenidos mediante una serie de mediciones previas y a través de la experiencia adquirida en el procesamiento de tareas, labores propias del jefe de planta.

Para presente proceso “GRASP Construcción”, se presentan dos constantes de relajación (α y θ) Donde α será la constante de selección de las tareas RCL (Restricted Candidate List) y θ la constante de selección de las máquinas MCL (Machine Candidate List)

Para hacer referencia a los elementos de las matrices usadas en la implementación del algoritmo, se usa la notación siguiente:
Sea P una matriz cualquiera, se tiene:

- P[0,0] representaría al primer elemento de la matriz: fila 0, columna 0.
- P[0,1] representaría al segundo elemento de la matriz: fila 0, columna 1.
- P[0] representaría toda la fila 0

Los tiempos de espera del trabajo a ser procesado (tiempos muertos) y los tiempos de recarga que puedan tener durante el proceso, se encuentran incluidos en los valores de la matriz T.

3.2 Estructuras usadas

Partiendo del supuesto: la existencia de una *instancia completa de trabajo* para la programación de máquinas-tareas, que incluye:

- Cantidad de tareas, máquinas y familias: N, M y K respectivamente.
- Matriz de tiempos de ejecución T, de las tareas en las familias de máquinas.

Se plantean la siguiente estructura de datos:

- **N**: número de tareas $J_1, J_2, ..., J_N$
- M: número de máquinas $M_1, M_2, ..., M_M$
- K: número de familias de máquinas $F_1, F_2, ..., F_K$

Matriz T: $[T_{ij}]_{NxK}$ de tiempos de ejecución, donde cada entrada representa el tiempo que se demora la tarea i-ésima en ser ejecutada en cualquier máquina de la familia j-ésima.

	F1	F2	F3	F4	F5	. . .	Fk
J1	20	25	11	32	33	...	22
J2	22	11	15	13	22	...	15
J3	8	90	52	152	12	...	223
. . .	...	...	...	...	...	...	
Jn	598	698	981	85	11	...	124

Tabla 1 Matriz T (tiempos de ejecución por familias)

Matriz G: $[G_{ij}]_{NxM}$ de números (0, 1 ó 2), donde cada entrada (i,j) representa el estado actual de la tarea i en la máquina j.

	M1	M2	M3	M4	M5	. . .	Mm
J1	2	0	1	2	...	...	2
J2	2	0	2	0	...	...	2
J3	0	0	0	1	...	...	0
. . .	...	...	...	...	...	...	...
Jn	0	2	2	2	...	...	0

Tabla 2 Matriz G (asignación tareas-máquinas)

Explicación de estados:

Estados		explicación
0	No asignada	No hubo asignación máquina-tarea.
1	ejecutándose	Actualmente la tarea está siendo ejecutada por la máquina.
2	asignada	Queda constancia que la tarea fue asignada a la máquina, durante la ejecución.

Tabla 3 Tipos de estados matriz G

Matriz R: $[R_{ij}]_{NxK}$ de números enteros positivos, donde cada entrada representa el orden (secuencia) de procesamiento de la tarea i-ésima en la familia j-ésima.

	F1	F2	F3	F4	F5	. . .	Fk
J1	4	3	2	1	5	...	9
J2	5	1	4	2	3	...	7
J3	2	1	3	4	5	...	8
. . .	...	...	...	...	...	...	...
Jn	6	9	1	2	3		10

Tabla 4 Matriz R (recorrido tareas-familias)

Matriz A: $[A_{ij}]_{Mx3}$ de números enteros (incluso cero), donde la primera columna representa la familia a la que pertenece la máquina. La segunda columna representa el estado (actual) de la maquina. La tercera columna representa el Tiempo acumulado de procesamiento que lleva la maquina hasta el momento, y la cuarta es el tiempo en que terminará la tarea que actualmente está ejecutando.

	Familia	Estado	T.acumulado	T_fin_ejecución
A1	1	0	254	25035
A2	2	1	233	21475
A3	1	1	45	11542
. . .	...	...	...	...
Am	3	0	68	215

Tabla 5 Matriz A (información máquinas)

Explicación de los valores de la columna Estados:

Estado		explicación
-1	deshabilitada	La máquina pertenece a una familia que ha sido deshabilitada.
0	disponible	Máquina libre, apta para ser asignada a alguna tarea.
1	ocupada	La máquina está ejecutando actualmente una tarea.

Tabla 6 Tipos de estados matriz A

Matriz J: $[J_{ij}]_{nx3}$ de números enteros (incluso cero), donde la primera columna representa el estado actual de la tarea y la segunda representa el Tiempo acumulado de procesamiento de la tarea (incluso tiempo de espera)

	Estado	T.ejecución
J1	0	10
J2	1	50
J3	2	22
. . .	...	....
Jn	1	41

Tabla 7 Matriz J (información Tareas)

Explicación de los valores de la columna Estado:

Estado		explicación
0	libre	Tarea que no ha sido seleccionada para pertenecer al conjunto S.
1	seleccionada	Tarea perteneciente al conjunto S y que está disponible para hacer asignada a una máquna.
2	ejecutándose	Tarea perteneciente al conjunto S y que está actualmente siendo ejecutada por una máquina.
3	finalizada	Tarea que terminó su ejecución por todas las familias de maquinas, ya no pertence al conjunto S

Tabla 8 Tipos de estados matriz J

3.3 Presentación del Algoritmo GRASP – fase de Construcción

Inicio Algoritmo **GRASP** construcción **(**N, M, K, ν , α**,** G, R, A, J **)**
1. leer archivos (a.csv, t.csv)
2. obtener N, M, K
3. Inicializar matrices (G, R, T, A, J)
4. Leer (q, α, θ, porcentaje_tm)
5. gp_tareas $\leftarrow$ M * porcentaje_tm
6. W $\leftarrow$ [N /gp_tareas] + 1
7. Para W del 1 al |W| hacer
   inicio
   7.1. TM $\leftarrow$ 0
   7.2. Carga_conjunto_S (S , J , ν ,W)
   7.3. Cambiar_estado_matriz_J (1, S, J)

7.4. Mientras ($S \neq \phi$) hacer
inicio
7.4.1. Si (es_primera_vez)
7.4.1.1. Para cada J[i] $\in$ J / i $\in$ S $\wedge$ h < M hacer
inicio
J[i][0] $\leftarrow$ 2
A[h][1] $\leftarrow$ 1
J[i][1] $\leftarrow$ J[i][1] + T [i][A[h][0]]
A[h][2] $\leftarrow$ A[h][2] + T [i][A[h][0]]
A[h][3] $\leftarrow$ TM + T [i][A[h][0]]
Asigna_orden_ejecucion (R[i][A[h][0]])
G[i][A[h]] $\leftarrow$ 1
i $\leftarrow$ siguiente_elemento_S
Fin Para

7.4.2. Caso contrario
inicio
7.4.2.1. $\beta \leftarrow$ Min (A[h][2] / (A[h][1] = 0)
$\lambda \leftarrow$ Max (A[h][2] / (A[h][1] = 0)
MCL $\leftarrow \phi$
7.4.2.2. Para cada A[h] $\in$ A / (A[h][1] = 0) hacer
Si A[h][2] $\in [\beta, \beta + \theta * (\lambda - \beta)]$ entonces $MCL \leftarrow MCL \cup \{ A[h]\}$

7.4.2.3. A* $\leftarrow$ Aleatorio $\{MCL\}$
7.4.2.4. $\beta \leftarrow$ Min (J[i][1] / i $\in$ S $\wedge$ J[i][0] = 1 $\wedge$ R[i][A*[0] = 0)
$\lambda \leftarrow$ Max (J[i][1] / i $\in$ S $\wedge$ J[i][0] = 1 $\wedge$ R[i][A*[0] = 0)
RCL $\leftarrow \phi$
7.4.2.5. Para cada (i $\in$ S $\wedge$ J[i][0] = 1 $\wedge$ R[i][A*[0] = 0) hacer
Si J[i][1] $\in [\beta, \beta + \alpha^* (\lambda - \beta)]$ entonces $RCL \leftarrow RCL \cup \{ J[i] \}$

J* $\leftarrow$ Aleatorio $\{RCL\}$
7.4.2.6. Si (J* $\neq \phi$ $\wedge$ A* $\neq \phi$) hacer
inicio
J*[0] $\leftarrow$ 2
J*[1] $\leftarrow$ J*[1] + T [J*][A*[0]]
A*[1] $\leftarrow$ 1
A*[2] $\leftarrow$ A*[2] + T [J*][A*[0]]
A*[3] $\leftarrow$ TM + T [J*][A*[0]]
Asigna_orden_ejecucion (R[J*][A*[0]])
G[J*][A*] $\leftarrow$ 1
fin

7.4.2.7. Caso contrario
ejecuta nuevamente *Mientras ($S \neq \phi$)* ***7.4*** hasta 2 oportunidades
Fin Caso contrario

7.4.3. repite_iteracion ← 0
7.4.4. **Mientras (∞)**
inicio
7.4.4.1. Para cada (A[h] ∈ A ∧ A[h][1] = 1 ∧ A[h][3] <= TM) hacer
inicio
i ← ubica_tarea_asociada (G,h)
G[i][h] ← 2
A[h][1] ← 0
7.4.4.1.1. Si ($\prod_{1}^{k} R_{i\,h} \neq 0$)
inicio
J[h][0] ← 3
Des-incluir (i,S)
fin Si
7.4.4.1.2. Caso contrario
J[h][0] ← 1

fin Para
7.4.4.2. Si ninguna _maquina_termino_su_ejecucion () ∧ repite_iteracion < 3
inicio
TM ← TM+3q
repite_iteracion ← repite_iteracion +1
ejecuta nuevamente 7. 4. 4.
fin
7.4.4.3. Si ninguna _maquina_termino_su_ejecucion () ∧ repite_ iteracion >= 3
inicio
7.4.4.3.1. FD ← verifica_familias_deshabilitadas (R)
Si (FD ≠ ϕ) hacer
inicio
deshabilitar_familia (A, FD)
ejecuta nuevamente 7. 4. 5.
fin
7.4.4.3.2. caso contrario
inicio
avanza_tiempo_mas_cercano (A, TM)
ejecuta nuevamente 7. 4. 4.
fin
fin Si
fin **Mientras (∞)**

7.4.5. TM ← TM+1q
fin *mientras (S ≠ ϕ)*
7.5. habilita_familias (A)
7.6. muestra_estructuras ()
Fin *Para W*

Fin Algoritmo **GRASP** construcción

3.4 Comentarios sobre el algoritmo GRASP propuesto - fase de Construcción

1. Leer archivos (a.csv, t.csv).

2. El valor N se obtiene de la cantidad de filas del *archivo t.cvs*.
   El valor M se obtiene de la cantidad de filas del *archivo a.csv*, y el valor K se obtiene de la cantidad de columnas del archivo *t.cvs.*

3. Las matrices G, R, T, A, J se llenan con los datos provenientes de los archivos a.csv y t.csv. Los campos restantes de dichas matrices se inicializan con cero.

4. Se solicita el ingreso de valores para *q* = quantum, *porcentaje_tm* = porcentaje máquina tarea y los respectivos coeficientes de relajamiento para listas de candidatas GRASP tanto de tareas y maquinas.

5. *gp_tareas* indica el número de tareas a ejecutarse por cada grupo.

6. W indica el número de *grupos de tareas* en el que serán divididas TODAS las tareas.

7. Inicio del bucle con **W** iteraciones

   7.1. Al comenzar cada iteración, el tiempo TM se inicializa en cero.

   7.2. La función *Carga_conjunto_S (S , J ,gp_tareas, W)* escoge *las gp_tareas* primeras tareas disponibles (aquellas que no han sido ejecutadas) de la matriz J –solo sus identificadores- y las copia al conjunto S. De esta manera el conjunto S tiene los identificadores de las tareas que serán ejecutadas y NO todas las tareas de la matriz J.

   7.3. Las tareas que fueron seleccionadas en el paso anterior son cambiadas a **estado igual a 1** (*seleccionada*) en la matriz J.

   7.4. Se inicia un bucle hasta que el conjunto S esté vació

      7.4.1. Si es la primera iteración para el grupo de tareas:
      Se asigna **todas** las tareas del conjunto S, a las máquinas; que por ser primera vez están todas disponibles. Sea J* la tarea escogida y A* la máquina escogida se tendrá:

         7.4.1.1. Para cada tarea del conjunto S hacer:
         Actualizar a estado 2 (ejecutada) la tarea J*.
         Asigno estado 1 (ocupada) a A*.
         Añado el tiempo T[J*][familia de A*] al campo tiempo de ejecución de J*.
         Añado el tiempo T[J*][familia de A*] al campo tiempo Acumulado de A*.
         Asigno en la matriz G [J*][A*] el valor de 1 (ejecutándose) que indica que actualmente se está ejecutando.

7.4.2. Caso contrario: (no es la primera iteración)

7.4.2.1. De las maquinas disponibles (estado 0); escojo las máquinas que tengan el menor y mayor valor del campo *T_acumulado,* y le asigno como identificador β y λ respectivamente.

7.4.2.2. Las máquinas disponible y cuyo valor *T_acumulado* están en el rango de valores [β **,** β +θ *** (** λ **-** β **)]** serán añadidas al rango RCL.

7.4.2.3. Se escogen aleatoriamente del rango RCL, la máquina A*.

7.4.2.4. Similarmente se escoge la tarea J*, que cumpla con la condición de estar en estado uno (seleccionado), pertenecer al conjunto S y que no haya sido ejecutada por la familia de la máquina A*. Del conjunto que cumplen esta condición, se escoge la tarea cuyo valor *T_ejecicion* sea el menor y mayor de todos y se le asigna β y λ respectivamente.

7.4.2.5. Se forma el rango de tarea cuyo valor *T_ejecucion* pertenezcan al rango [β **,** β +θ * (λ **-** β)] y se escoge aleatoriamente de dicho intervalo, la tarea J*.

7.4.2.6. Si J* y A* existen:
Asigno estado 2 (ejecutada) a J*.
Añado el tiempo T[J*][familia de A*] al campo tiempo de ejecución de J*.
Asigno estado 1 (ocupada) a A*.
Añado el tiempo T[J*][familia de A*] al campo tiempo Acumulado de A*.
Asigno el orden de ejecución de la tarea J' respecto a las familias existentes.
Asigno en la matriz G [J*][A*] el valor de 1 (ejecutándose)

7.4.2.7. Caso contrario, si no hay tarea disponibles para la máquina escogida:
Se escoge otra máquina, del rango RCL, esperando que esta vez si exista alguna tarea disponible –ya que la selección es aleatoria–. Esto se realiza en 2 oportunidades.

7.4.3. La variable *repite_iteracion* indica las veces que se ha intentado verificar que una tarea haya terminado. En este instante se asigna con valor cero, antes de entrar al bucle.

7.4.4. **Mientras (**∞**)** Indica que se iniciará un bucle infinito.

7.4.4.1. Se inicia una verificación para TODAS las máquinas que actualmente se encuentren procesando (estado 1) y cuyo *T_fin_ejecucion* sea mayor igual al tiempo TM. Si hay máquinas que cumplan con esta condición:

(sea A' una máquina que cumple la condición anterior)

Se ubica la tarea (J') que actualmente está siendo ejecutada por la máquina A'. Esta información se saca de la matriz G, sabiendo que en este momento tiene valor 1 (estado: *ejecutándose*).
Como ya se terminó de ejecutar J' en A', se asigna en la matriz G el estado 2 (ejecutada).
Se libera la máquina A', asignándole, estado 0 en la matriz A.

7.4.4.1.1. Si la tarea que ha terminado de ser ejecutada J'. ya ha pasado por todas la familias a lo largo se su recorrido, entonces:

Se asigna a J' el estado 3 (Finalizada) en la matriz J.
Se elimina J' del conjunto S,

7.4.4.1.2. Si no hay máquinas que hayan pasado por TODAS las familias, entonces:

Se asigna a J' el estado 1 (seleccionada) en la matriz J. para que puedan ser escogidas por otras maquinas pertenecientes a las familias que le falten por recorrer.

7.4.4.2. Si ninguna máquina ha terminado su ejecución en la presente iteración y el valor de la constante *repite_iteracion* es menor a 3, entonces:

Se agrega 3 unidades de tiempo al tiempo TM con el fin de acelerar trabajos que actualmente están siendo ejecutados.
Se aumenta en 1 el valor de la variable *repite_iteracion.*
Regresar a 7.4.4, con los nuevos valores de *TM* y *repite_iteracion*

7.4.4.3. Si después de avanzar 9 unidades de tiempo (3 iteraciones y cada una de 3 unidades de tiempo) y ninguna máquina ha terminado en ejecutarse, es probable que todas las maquinas de una familia, no tengan tareas pendientes y que al realizar la selección mediante la meta heurística GRASP se escoja precisamente una máquina de dicha familia.

7.4.4.3.1. Se confirma el paso anterior mediante la función *verifica_familias_deshabilitadas(R)* que arroja la existencia o no de dichas familia. Si existen familias que cumplan la condición:

Se deshabilita la familia; asignándole el valor de -1 (estado deshabilitada) a las máquinas de dicha familia en la matriz A.
Avanzar a 7.4.5, para escoger una nueva máquina y asignarle una tarea.

7.4.4.3.2. caso contrario (Si no hay familias deshabilitadas)

Indicará que hay tareas que tienen un tiempo de ejecución muy prolongado y habrá que calcular la diferencia de tiempo entre su *T_fin_ejecucion* y el tiempo actual TM, para realizar un desplazamiento de tiempo y sumarle esa diferencia de tiempo al valor de TM +1.
Regresar a 7.4.4

Fin **Mientras (∞)**

7.4.5. Se añade en una unidad al tiempo TM.
fin *mientras* ($S \neq \phi$)

7.5. Como todas las tareas seleccionadas han sido ejecutadas completamente, (el conjunto S esta vacío), se habilita a todas las tareas nuevamente para el inicio del siguiente grupo de tareas a ejecutarse. Se asigna el valor de cero (estado disponibles) a todas las maquinas que tenían estado -1 (estado inhabilitadas).

7.6. Se lista las matrices G, R, A, J para mostrar el resultado después de cada finalización de ejecución de grupos de tareas.

Fin *Para W.*

Fin Algoritmo **GRASP** construcción.

4 Implementación y experimentos numéricos

4.1 Desarrollo de un ejemplo

4.1.1 Datos de entrada

La entrada de datos al presente algoritmo se realizará mediante archivos de texto cuyos valores son separados por comas (archivos con extensión csv) los mismos que son generados usando MS Excel.

Archivo a.csv
Archivo donde se indica la familia a la que pertenecen las máquinas. La numeración izquierda Excel (filas) indica los identificadores de las **maquinas**. Para este ejemplo existen 40 de ellas.

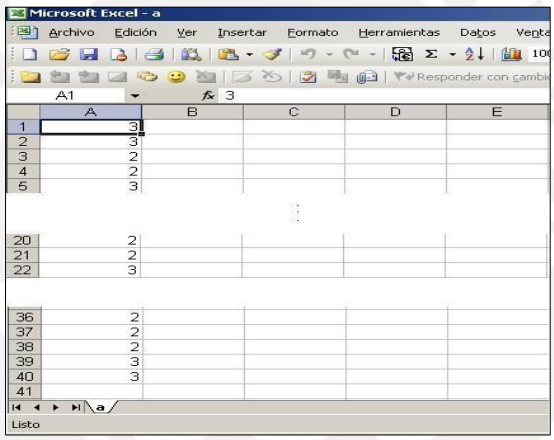
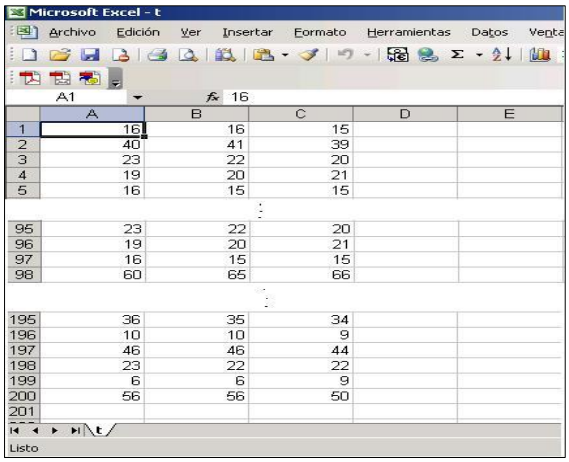


Figura 11 Extracto archivo a.csv

Archivo t.csv
Archivo donde se muestran los tiempos de ejecución que toman las tareas en las distintas familias de máquinas. La numeración de las filas del Excel indica los identificadores de las **tareas**, y las columnas son las **familias** existentes. Para este ejemplo, existen 200 tareas y 3 familias:



	A	B	C	D	E
1	16	16	15		
2	40	41	39		
3	23	22	20		
4	19	20	21		
5	16	15	15		
95	23	22	20		
96	19	20	21		
97	16	15	15		
98	60	65	66		
195	36	35	34		
196	10	10	9		
197	46	46	44		
198	23	22	22		
199	6	6	9		
200	56	56	50		
201					

Figura 12 Extracto archivo t.csv

Otras variables

α: Constante de relajamiento para el GRASP tareas: 0.05
θ: Constante de relajamiento para el GRASP máquinas: 0.05
porcentaje_tm: Es el porcentaje de tareas respecto a las máquinas que integraran cada iteración dentro del algoritmo propuesto. Para el presente ejemplo el valor es 4, vale decir que cada iteración (grupo) estará compuesto de 160 tareas para las 40 máquinas.

4.1.2 Ejecución

Variable	Valor
N (número Tareas)	200
M (número Máquinas)	40
K (número Familias)	3

CANTIDAD DE TAREAS POR GRUPO: 160
CANTIDAD DE GRUPOS: 2

Matriz T

	F1	F2	F3
J1	16	16	15
J2	40	41	39
J3	23	22	20
J4	19	20	21
J5	16	15	15
J6	60	65	66
J7	25	25	25
J8	32	33	34
J9	36	35	34
J10	10	10	9
J11	46	46	44
J12	23	22	22
J13	6	6	9
J14	56	56	50
J15	11	12	12

J146	36	35	34
J147	25	25	25
J148	56	56	50
J149	16	16	15
J150	40	41	39
J151	23	22	20
J152	19	20	21
J153	16	15	15
J154	60	65	66
J155	16	16	15
J156	40	41	39
J157	23	22	20
J158	19	20	21
J159	16	15	15
J160	60	65	66
J161	25	25	25
J162	32	33	34
J163	36	35	34

J183	36	35	34
J184	10	10	9
J185	46	46	44
J186	23	22	22
J187	13	14	15
J188	9	10	11
J189	3	2	5
J190	12	11	12
J191	16	15	15
J192	60	65	66
J193	25	25	25
J194	32	33	34
J195	36	35	34
J196	10	10	9
J197	46	46	44
J198	23	22	22
J199	6	6	9
J200	56	56	50

Matriz G

	M1	M2	M3	M4	M5	M6
J1	0	0	0	0	0	0
J2	0	0	0	0	0	0
J3	0	0	0	0	0	0
J4	0	0	0	0	0	0

M35	M36	M37	M38	M39	M40
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

J197	0	0	0	0	0	0
J198	0	0	0	0	0	0
J199	0	0	0	0	0	0
J200	0	0	0	0	0	0

0	0	0	0	0	0	
0	0	0	0	0	0	
0	0	0	0	0	0	
0	0	0	0	0	0	

Matriz R

	F1	F2	F3
J1	0	0	0
J2	0	0	0
J3	0	0	0
J4	0	0	0
J5	0	0	0
J6	0	0	0

J136	0	0	0
J137	0	0	0
J138	0	0	0
J139	0	0	0
J140	0	0	0
J141	0	0	0
J142	0	0	0
J143	0	0	0
J144	0	0	0

J192	0	0	0
J193	0	0	0
J194	0	0	0
J195	0	0	0
J196	0	0	0
J197	0	0	0
J198	0	0	0
J199	0	0	0
J200	0	0	0

Matriz A

	Familia	Estado	T.acumulado	T_fin_ejecucion
A1	3	0	0	0
A2	3	0	0	0
A3	2	0	0	0
A4	2	0	0	0
A5	3	0	0	0
A6	3	0	0	0

A33	3	0	0	0
A34	2	0	0	0
A35	2	0	0	0
A36	2	0	0	0
A37	2	0	0	0
A38	2	0	0	0
A39	3	0	0	0
A40	3	0	0	0

Matriz J

	Estado	T.ejecucion
J1	0	0
J2	0	0
J3	0	0
J4	0	0
J5	0	0
J6	0	0

J136	0	0
J137	0	0
J138	0	0
J139	0	0
J140	0	0
J141	0	0
J142	0	0
J143	0	0
J144	0	0

J192	0	0
J193	0	0
J194	0	0
J195	0	0
J196	0	0
J197	0	0
J198	0	0
J199	0	0
J200	0	0

Tabla 9 Matrices T, G, R, A, J al iniciar la ejecución

tiempo=160:
TERMINADAS -> máquina=10 tarea=45
TERMINADAS -> máquina=29 tarea=44
---
tiempo=161:
máquinas libres:

1	2	3	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	21	22	23	24	25	26	27	28	29	30	31	33	34	35	36	37	38	40

máquina escogida= 8 Familia 1

tareas libres:

1	2	3	4	5	6	7	9	11	12	13	14	15	17	18	19	20	21	22	23	24	25	26	27	28	30	32	33	34	35	36	37	38	39	40	41	42	43	46

tarea escogida= 138

TERMINADAS -> máquina=39 tarea=85

Tabla 10 Reporte parcial durante la ejecución del algoritmo

Matriz G

	M1	M2	M3	M4	M5	M6	M35	M36	M37	M38	M39	M40
J1	2	0	0	0	0	0	0	0	0	0	0	0
J2	0	2	0	0	0	0	0	0	0	0	0	0
J3	0	0	2	0	0	0	0	0	0	0	2	0
J4	0	0	0	2	0	0	0	0	0	0	0	0
J197	0	0	0	0	0	0	0	0	2	0	0	0
J198	0	0	0	0	0	0	0	0	0	2	0	0
J199	0	0	0	0	0	0	0	2	0	0	2	0
J200	0	0	0	0	0	0	2	0	0	0	0	2

Matriz R

	F1	F2	F3
J1	3	2	1
J2	3	2	1
J3	3	1	2
J4	3	1	2
J5	3	2	1
J6	3	2	1
J36	3	1	2
J37	3	1	2
J38	3	1	2
J39	3	2	1
J40	3	2	1
J41	3	2	1
J42	3	2	1
J43	1	3	2
J44	1	3	2
J192	3	1	2
J193	3	2	1
J194	3	1	2
J195	3	1	2
J196	3	1	2
J197	3	1	2
J198	3	1	2
J199	3	2	1
J200	3	2	1

Matriz A

	Familia	Estado	T.acumulado	T_fin_ejecucion
A1	3	0	246	25
A2	3	0	274	34
A3	2	0	255	35
A4	2	0	258	97
A5	3	0	290	44
A6	3	0	250	132
A33	3	0	279	25
A34	2	0	243	33
A35	2	0	295	183
A36	2	0	240	82
A37	2	0	285	46
A38	2	0	296	202
A39	3	0	280	255
A40	3	0	243	50

Matriz J

	Estado	T.ejecucion
J1	3	47
J2	3	120
J3	3	65
J4	3	60
J5	3	46
J6	3	191
J36	3	67
J37	3	21
J38	3	162
J39	3	47
J40	3	120
J41	3	65
J42	3	60
J43	3	46
J44	3	191
J192	3	191
J193	3	75
J194	3	99
J195	3	105
J196	3	29
J197	3	136
J198	3	67
J199	3	21
J200	3	162

Tabla 11 Matrices G, R, A, J al finalizar la ejecución

Explicación tabla 10

Durante la ejecución en el tiempo 161 existen 36 maquinas disponibles, las cuales ingresarán al proceso de selección GRASP, teniendo en cuenta el valor de α (constante de relajación para las maquinas) y el tiempo acumulado de ejecución que lleva actualmente cada máquina. Resultando seleccionada la máquina 8.
En dicho tiempo, las tareas disponibles que faltan ser ejecutadas por la familia 1 (familia de la máquina 8) entrarán al proceso de selección GRASP, teniendo en cuenta el valor del parámetro de relajamiento de tareas θ y el tiempo de ejecución que lleva cada tarea, resultando escogida la tarea 13. Por tanto, se asigna la tarea 13 a la máquina 8.
En ese mismo tiempo la máquina 39 terminó de ejecutar la tarea 85.
Si la tarea 85 NO ha transitado por TODAS las familias de maquinas, entonces, pasará a formar parte en la siguiente selección GRASP, caso contrario será asignada con el estado 3 (finalizada) saliendo completamente del proceso.
Similarmente, si la familia de la máquina 39 aún NO ha ejecutado TODAS las tareas que le corresponden, continuarán siendo asignadas para procesar tareas, de lo contrario, se le asignará el estado -1 (deshabilitada) a todas las maquinas de dicha familia.

4.1.3 Reporte Final

Asignación máquina tareas

FAMILIA 1 :

M8	J8	J12	J13	J15	J17	J18	J33	J34	J42	J54	J56	J57	J59	J62	J65	J71	J75	J78	J84	J92	J97	J100	J107	J109	J116	J124	J145	J14
M10	J3	J4	J7	J10	J14	J24	J27	J28	J35	J37	J38	J41	J51	J53	J66	J69	J70	J77	J80	J94	J98	J111	J117	J118	J132	J133	J134	J13
M16	J1	J9	J11	J16	J20	J25	J26	J43	J44	J45	J49	J58	J64	J67	J76	J83	J87	J89	J90	J96	J101	J102	J103	J106	J108	J115	J129	J13
M29	J5	J23	J29	J30	J32	J39	J40	J50	J52	J63	J72	J74	J95	J105	J110	J113	J114	J122	J123	J126	J127	J128	J131	J139	J140	J152	J154	J16
M31	J2	J6	J19	J21	J22	J31	J36	J46	J47	J48	J55	J60	J61	J68	J73	J79	J81	J82	J85	J86	J88	J91	J93	J99	J104	J112	J119	J12

J146	J150	J151	J153	J155	J165	J166	J168	J169	J192	J195	
J136	J137	J138	J143	J149	J156	J158	J161	J164	J170	J171	J172	J173	J181	J187	J194	J196	J197	J199
J130	J141	J142	J144	J163	J174	J176	J185	J188	J193	J198								
J160	J162	J175	J178	J180	J189	J190	J200											
J120	J121	J125	J135	J147	J148	J157	J159	J167	J177	J179	J182	J183	J184	J186	J191			

FAMILIA 2 :

M3	J3	J31	J41	J58	J66	J79	J85	J88	J100	J102	J112	J156	J157	J163
M4	J4	J52	J56	J96	J124	J135	J148	J159	J164	J182	J199			
M9	J9	J28	J64	J82	J92	J129	J138	J169	J177	J188				
M11	J11	J47	J103	J122	J126	J130	J166	J171	J200					
M13	J7	J13	J48	J69	J73	J90	J123	J160	J167	J173	J191			
M14	J14	J54	J63	J70	J71	J83	J107	J174						
M15	J15	J23	J50	J97	J98	J128	J140	J149	J161	J175				
M18	J18	J46	J55	J81	J84	J105	J113	J119	J120	J121	J168	J178	J186	
M20	J20	J29	J45	J77	J145	J146	J150	J154	J180	J193				
M21	J21	J62	J76	J89	J95	J104	J142	J144	J147	J162	J172	J181	J184	J187	J189
M25	J2	J25	J42	J67	J80	J111	J116	J133	J134	J136	J143	J183	J185		
M30	J30	J39	J44	J57	J65	J110	J137	J190							
M32	J1	J12	J32	J59	J87	J108	J114	J125	J131	J192					
M34	J8	J17	J19	J34	J49	J75	J94	J127	J141	J155	J158	J165	J194		
M35	J5	J6	J26	J33	J35	J68	J101	J106	J109	J195					
M36	J10	J24	J36	J40	J43	J51	J61	J93	J117	J132	J170	J176	J196		
M37	J37	J53	J86	J91	J115	J118	J139	J151	J153	J197					
M38	J16	J22	J27	J38	J60	J72	J74	J78	J99	J152	J179	J198			

FAMILIA 3 :

M1	J1	J18	J21	J58	J59	J65	J73	J80	J95	J104	J133	J136	J144	J161				
M2	J2	J11	J32	J62	J91	J125	J135	J158	J162	J197								
M5	J5	J16	J55	J57	J60	J76	J110	J129	J130	J146	J152	J165						
M6	J4	J6	J30	J63	J124	J154	J166											
M7	J7	J14	J44	J69	J84	J96	J167	J175										
M12	J12	J15	J45	J53	J61	J100	J127	J134	J139	J140	J159	J169	J170	J172	J181	J190	J196	
M17	J17	J42	J89	J99	J101	J102	J120	J138	J153	J163	J177							
M19	J10	J19	J43	J56	J90	J93	J98	J111	J114	J156	J179							
M22	J22	J41	J54	J66	J81	J83	J103	J112	J121	J131	J150	J173	J182	J194				
M23	J3	J9	J23	J72	J74	J160	J183											
M24	J24	J38	J68	J87	J92	J118	J145	J184	J192									
M26	J13	J26	J29	J31	J47	J49	J67	J105	J108	J155	J186	J195						
M27	J20	J27	J50	J75	J82	J94	J119	J123	J137	J141	J142	J147	J187					
M28	J8	J28	J35	J71	J107	J113	J143	J149	J168	J171	J180	J188	J189	J198				
M33	J33	J34	J46	J51	J52	J64	J88	J97	J106	J115	J151	J164	J174	J191	J193			
M39	J25	J39	J70	J78	J79	J85	J86	J109	J122	J126	J128	J157	J176	J178	J185	J199		
M40	J36	J37	J40	J48	J77	J116	J117	J132	J148	J200								

Tiempo de Ejecución de cada Máquina

M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24	M25	M26	M27
245	288	257	222	283	258	260	963	281	912	299	257	244	254	261	928	278	285	260	253	256	278	273	297	270	282	250

M28	M29	M30	M31	M32	M33	M34	M35	M36	M37	M38	M39	M40
319	946	251	906	252	253	280	274	220	258	248	274	267

Mayor Tiempo de Ejecución

M8
963

Tabla 12 Reporte final algoritmo propuesto

4.2 Replanteamiento del concepto makespan.

Como se mencionó en el capitulo anterior, el objetivo más buscado en una planificación, es reducir el tiempo del makespan.

En la línea de producción actual, el makespan, no es el 100% del tiempo acumulado que toma en ejecutar todos los trabajos propuestos. Existen perturbaciones para el makespan como la existencia de familias de máquinas, lo cual se debe a que la familia **no** acompaña a la ejecución de la tarea de principio a fin sino, que la familia realiza su trabajo puntual sobre la tarea y se retira momentáneamente de la ejecución, para que la tarea pase igualmente por las demás familias. Esto impide conocer el tiempo exacto en el que el lote completo de tareas fue ejecutado

Otra perturbación del makespan para la presente línea de producción es la *independencia* en la ejecución de las tareas, es decir, la tarea no siempre pasa por la máquina A, luego por la máquina B y así sucesivamente siguiendo un orden establecido; mas bien pasará por las tres familias en cualquier orden, y dentro de cada familia podrá optar por cualquier máquina disponible siempre y cuando ésta pertenezca al rango de máquinas con el menor tiempo de ejecución (aquellas que no han trabajado mucho).

Como el presente algoritmo muestra conceptos nuevos en la literatura del FSSP, el makespan es reformulado y se convierte en el mayor tiempo de ejecución de todas las máquinas, criterio que se acerca más al concepto del makespan puro.

La calidad del algoritmo se mostrará cuando éste sea comparado con el algoritmo voraz, en las mismas condiciones.

4.3 Especificaciones técnicas

Para la fase de implementación del presente algoritmo, se uso la siguiente configuración

Computadora personal Pentium IV, 1.8 GHz, 512 MB RAM
Sistema Operativo Windows 2000
Servidor de aplicaciones Web Apache versión 2.0.58 para Windows
Lenguaje de programación PHP versión: 5.1.4
Macromedia Dreanweaver MX 2004 versión 7.0.1
Microsoft Office Excel 2003

Las pruebas respectivas se realizaron en un servidor Web real y actualmente operativo
http://macareo.pucp.edu.pe/cramirez/
Sistema Operativo Linux versión 2.6.9-42.0.2
Servidor de aplicaciones Web Apache versión 2.0 para Linux
Lenguaje de programación PHP versión: 4. 3. 9.

5 Conclusiones y trabajos finales

Para el ejemplo los resultados del algoritmo GRASP, propuesto en la presente tesis, superan al resultado del algoritmo voraz, para las mismas instancias (cantidad de tarea y máquinas) con valores α = 0.05 y θ = [0.2 – 0.3]. Dando como resultado una mejora mayor al 3%. Haciendo válido el resultado del algoritmo

Algoritmo Voráz

	α = 0
θ = 0	962

Algoritmo GRASP propuesto

	α = 0.05	α = 0.1	α = 0.2	α = 0.3
θ = 0.05	**962**	964	968	966
θ = 0.1	967	962	966	971
θ = 0.2	962	963	965	**941**
θ = 0.3	**959**	965	**959**	967

Tabla 13 Comparación Algoritmo Voraz vs. GRASP propuesto

El ejemplo se corrió bajo las siguientes instancias: 200 tareas, 40 máquinas distribuidas en 3 familias. Esta configuración se acerca más a la de una línea de producción pequeña-mediana, haciendo que el algoritmo sea perfectamente aplicado a las industrias peruanas. Esto no excluye a las industrias grandes, ya que se podría dividir la gran cantidad de tareas de dicha línea afinando el parámetro *porcentaje_tm* y haciendo que el proceso de selección GRASP tenga más máquinas y tareas como valores de entrada.

- El concepto de familias de máquinas no ha aparecido en la literatura sobre Flow shop scheduling, por lo que su uso inspira un carácter innovador al algoritmo.

- En la implementación se usó tecnología Web, por las siguientes razones:

  ➢ La gran mayoría de las empresas manufactureras, tienen en distintos lugares geográficos sus plantas de producción ya sea por la cercanía de materia prima, costo de la mano de obra, seguridad, leyes que los respaldan, entre otras razones. La implementación en Web, permite acceder al algoritmo desde distintos lugares del mundo, siendo posible customizar el mismo con solo ingresar los valores de entrada (archivos con extensión csv y variables locales) para su actual realidad.

  ➢ Bajo costo de mantenimiento: las mejoras, correcciones y actualizaciones se realizarán sólo en el servidor de aplicaciones y no en cada máquina que tenga instalado el algoritmo. Evitando el costo de mantenimiento de personal capacitado en dichos lugares.

  ➢ No se necesita tener computadoras con una potente configuración de hardware, donde se desee usar el algoritmo, bastará ejecutar el algoritmo en máquinas con un mínimo de hardware, siempre y cuando se cuente con acceso a Internet.

- En la implementación se usaron tecnologías estables y maduras en el ambiente Web, y de código abierto (lenguaje PHP y servidor de aplicaciones Apache); evitando el costo en licencias y de personal capacitado que le dé mantenimiento.

Trabajos futuros que se recomiendan realizar para el presente algoritmo son:

- Confrontar el algoritmo con otras meta heurísticas que han tenido éxito en tareas de optimización combinatoria; es el caso de los algoritmos genéticos, colonia de hormigas, búsquedas tabú y demás. Todas éstas desarrolladas bajo la mismas condiciones.
- Realizar la fase GRASP mejoría del algoritmo para afinar la distribución tarea-máquina
- Añadir el algoritmo propuesto como parte de un sistema de apoyo a la planificación (CAP Computer Arded Planning) modificando sus inputs y outputs para que puedan interrelacionarse con otras aplicaciones pertenecientes al CAP.

Referencias Bibliográficas

[1] Tupia M . Un algoritmo GRASP para resolver el problema de la programación de tareas dependientes en máquinas diferentes. En las memorias de la Conferencia Latino Americana de Informática CLEI (30, 2004, Perú), Editores Solar M; Fernández-Baca D; Cuadros-Vargas E. p.129-139, 2004.

[2] Garey M; Jonson D. Local Search in Combinatorial Optimization. Editorial John Wiley & Sons, Inglaterra, 1997.

[3] Feo T; Resende M. Greedy Randomized Adaptive Search Procedure. Journal of Global Optimization. 6, 109-133, 1995.

[4] Martí R; Moreno-Vega J. Métodos Multiarranque. Revista Iberoamericana de Inteligencia Artificial, 19, 49-60, 2003.

[5] Melián B; Moreno Peres J; Moreno Vega J. Metaheuristicas: Una visión global. Revista Iberoamericana de Inteligencia Artificial. 19, 7-28, 2003.

[6] Rios Mercado R; Bard J. Heurística para secuenciamiento de tareas en Líneas de flujo. Ciencia UNAL - Facultad de Ingeniería Mecánica y Eléctrica - Universidad Autónoma de nuevo León. 3(4): 420-427, 2000.

[7] Marti R; Moreno J; et al. Metaheurísticas en Optimización Combinatoria. Procedente de la conferencia: 25 años de Matemáticas en la Universidad de La Laguna, p. 443-451, 1996.

[8] Silver E; Vidal R; De Werra D. A tutorial of Heuristic Methods. European Journal of Operational Research, 5(1): 62-74, 1980.

[9] Silver E. An overview of heuristic solution method. Journal of the Operational Research Society. 55 (9): 936-956, 2004.

[10] Francis Y; Chin T; Long-lieh T. On J-maximal and J-minimal flow shop scheduling. Journal of the ACM. 28 (3): 462 – 476, 1981.

[11] Yamada T; Nakano R. Genetic Algorithms for Job-Shop Scheduling Problems. Proceeding from seminar of Modern Heuristic for Decision Support. p. 67-81 Londres – Inglaterra, 1997.

[12] Lesh N; Leonardo B; Lopes L. Human-Guided Search for Jobshop Scheduling. Mitsubishi Electric Research Laboratories TR2002-43, Enero 2003.

[13] Baptiste P; Le Pape C; Nuijten W. Constraint-Based Optimization and Approximation for Job-Shop Scheduling. From documentation ILOG S.A. France.

[14] Alfonso M; Barber F. Combinación de procesos de clausura y CSP para la resolución de problemas de Scheduling. Revista Iberoamericana de Inteligencia Artificial 9, 20-26, 2000.

[15] Weng M. Scheduling Flow-shops with limited buffer spaces. Proceedings of the 2000 Winter Simulation Conference. 2, 1359-1363. Florida USA, 2000.

[16] Acero R; Torres J. Aplicación de una heurística de búsqueda tabú en un problema de programación de tareas en línea flexible de manufactura. Revista de la Sociedad de Estadística e Investigación Operativa, Bogotá-Colombia, 5 (2): 283–297, 1997.

[17] Voon-Yee V; Wen-Jing H. Applying Cilk in Provably Efficient Task Scheduling. The Computer Journal, 42 (8): 699-712, 1999.

[18] Simmen O; Sousa L. Comunication contention in task scheduling. The Journal of Supercomputing archive. 27 (2): 177 - 194, 2004.

[19] Taylor S. Process Flow Scheduling (recurso en línea). Disponible http://uwacadweb.uwyo.edu/sgt/process.htm

[20] Brassar G; Bratley P. Fundamental of algoritmics. Editorial Prentice Hall, New Jersey - USA, 1996.

[21] Faulín J; Ángel J. Introducción a la Investigación Operativa (recurso en línea). Disponible http://www.cyta.com.ar/ta0405/v4n5a1.htm

[22] Hillier F. Investigación de operaciones. Editorial McGraw-Hill, 2002.

[23] Krishna C. Real-time systems. Editorial McGraw-Hill, 1997.

[24] Castillo E; Conejo A; et al Formulación y Resolución de Modelos de Programación Matemática en Ingeniería y Ciencia. Universidad de Castilla-La Mancha, España, 2002.

[25] Mompin J. Inteligencia Artificial: conceptos, técnicas y aplicaciones. Marcombo, Barcelona-España, 1987.

[26] Pollard N.S. IEEE Grasp Parallel Algorithms for Synthesis of Whole-Hand Grasps. Proceeding of International Conference on Robotics and Automation (1999, USA), Albuquerque-USA. Disponible http://www.cs.brown.edu/people/~nsp/grasp.html, 1999.

[27] Capuz S. Introducción al proyecto de Producción Ingeniería concurrente para el diseño de producto. Editorial Alfaomega - Universidad Politecnica de Valencia, 1999.

[28] Kumara S; Kashyap R. Artificial Intelligence Manufacturing theory and Practice Editorial Industrial Engineering and Management - Institute of Industrial Engineers 1999.

[29] Bard J; Feo T; Holland S. Facility-wide planing and scheduling of wiring board assembly journal of Operations Research, 43 (2): 219-230, 1995.

[30] Shah M. Knowledge-based dynamic scheduling in a steel plant. Proceeding of Conference on Artificial Intelligence Applications CAIA (6, 1990, USA), editores Damian R; Silverman J, California-USA, p. 108-113, 1990.

[31] Rauch-Hindin W. Aplicaciones de la inteligencia artificial en la actividad empresarial, la ciencia y la industria: fundamentos - aplicaciones. Editorial Díaz de Santos, Madrid - España, 1989.

[32] Pinedo M. Scheduling, theory, algoritms and systems (second edition). Editorial Prentice hall 2002.

[33] Prawda. J. Métodos y Modelos de Investigación de operaciones vol. 2 (Modelos estocásticos). Editorial Limusa, Mexico, 1995.

[34] ROBÓTICA. Instituto Tecnológico de Estudios Superiores de Monterrey - Laboratorio de Ingeniería Mecánica e Ingeniería Mecatrónica. Disponible http://dim.tol.itesm.mx/labs/lim/robots.pdf.

[35] Ríos R; Bard J. Heurísticas para el secuenciamiento de tareas en líneas de flujo (recurso en línea). Universidad Autónoma de Nuevo León (México) - Universidad de Texas (USA) Disponible http://osos.fime.uanl.mx/~roger/ papers, 2000.

[36] Acero R; Torres J. Aplicación de una heurística de búsqueda tabú en un problema de programación de tareas en línea flexible de manufactura. Revista de la Sociedad de Estadística e Investigación Operativa, Bogotá-Colombia, 5(2): 283–297, 1997.

[37] Chand S; Schneeberger H. Single machine scheduling to minimize earliness subject to no- tardy jobs. European Journal of Operational Research, 34, 221-230, 1988.

[38] Campello R; Maculan N. Algoritmos e Heurísticas: desenvolvimento e avaliaçao de performance, Apolo Nacional, Brasil, 1993.

[39] Holstein D. Una Metaheurística Co-evolutiva para el Problema del Viajante de Comercio, Informe Final del Trabajo de Grado. Disponible http://www.densis.fee.unicamp.br/~moscato/papers/HolsteinThesis.pdf.

[40] Leung J. Handbook of Scheduling algorithms, models and performance analysis. Editorial CHAPMAN & Hall - CRC Computer and Information science series. 2004.

[41] Bard J; Feo T. Operations sequencing in discrete parts manufacturing. Journal of Management Science, 35, 249-255, 1989.

[42] Binato S; Hery D; Resende M. A GRASP for Job shop Scheduling (recurso en línea), INFORMS Spring CSTS Meeting, Carmel, California, USA 1998. Disponible http://public.research.att.com/~mgcr/doc/gjss.pdf

[43] Gonçalves J; Magalhäes J; Resende M. A Hibrid Genetic algorith for the Job Shop Scheduling. AT&T Labs Research, Technical Report No. TD-5EAL6J, 2002.

[44] Taillard E. Parallel Taboo Search Technique for the Job shop Scheduling Problem, Journal on Computing Science, 6, 108-117, 1994.

[45] Chambers J; Barnes W. Taboo Search for the Flexible-Routing Job Shop Problem. Department of Computer Sciences, University of Texas-USA, Technical Report No. TAY 2, 124, 1997.

[46] Davis L. Job shop scheduling with genetic algorithms**.** Proceeding of International Conference on Genetic Algorithms and their Applications (1, 1985, USA) editor Davis L, Morgan – Kaufmann USA, p. 136-140, 1985.

[47] Feo T; Sarathy T; McGahan J. A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. Computers and Operations Research, 23, 881-895, 1996.

[48] Subramani V; Kettimuthu R; et. al. Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests (recurso en línea). Department of Computer and Information Science of Ohio State University, USA. Disponible http:// www.gridforum.org, 2003.

[49] Kleinrock L. Quering Systems, John Wiley Editions, USA, p. 10, 1974.

[50] Yongpei Guan Y; Wen-Qiang Xiao; Cheung R.; et. al. A multiprocessor task scheduling model for berth allocation heuristic and worst-case analysis, Operations Research Letters, No. 30, 343 – 350, Elsevier Science, 2002.

[51] Naveen G; Sachin J; Chaitanya S. A Randomized Algorithm for Flow Shop Scheduling Proceedings from 19th Conference, Chennai, India, December 1999.

[52] Etiler O; Toklu B; Atak M. A genetic algorithm for flow shop scheduling problem journal of the operational research society 55, 830-835 , 2004.

[53] Banús J; Moncusí A; Labarta J. The last-call scheduling algorithm for periodic and soft aperiodic tasks in real-time systems. Reporte del Departament d'Enginyeria Informàtica Universitat Rovira i Virgili, Tarragona – España, 2000.

[54] Widmer M; Hertz A. A new heuristic for the flow shop sequencing problem. Europe Journal Operacional Resource 41, 186-193.

[55] Pendones J; Corominas A; Pastor R. Procedimiento heurístico para minimizar el Cmax en celdas robotizadas con buffers finitos y piezas distintas. Procedente de 27 Congreso Nacional de Estadística e Investigación Operativa. Lleida - España, p. 3796-3802, 2003.

[56] Salazar M; Rios Mercado R. Minimización heurística del número de tareas tardías la secuenciar líneas de flujo. Revistas Ingenierías (Revista de la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Nuevo León) Mexico, 7 (23): 52-57, 2004.

[57] Alfonso M; Barber F. Combinación de procesos de clausura y CSP para la resolución de problemas de scheduling, Revista Iberoamericana de Inteligencia Artificial, Murcia - España, 9, 20-26.

[58] Crama Y; Van de Klundert J. Cyclic scheduling of identical parts in a robotics cell. Operations Research 45 (6): 952-968, 1997.

[59] Garey M; Johnson D; Sethi R. The complexity of flowshop and jobshop scheduling Mathematics of Operations Research. 13, 330-348, 1976.

[60] Altiok, T. Production lines with phase-type operation and repair times and finite buffers. International. Journal of Production research. 23, 489-498. 1985.

[61] Bolat, A. Sequencing jobs for an automated manufacturing module with buffer. European Journal of Operational Research, 96, 622-635. 1997.

[62] Applegate, D; Cook W. A computational study of the job shop scheduling problem. ORSA Journal on Computing, 3, 149-156. 1991.

[63] Balas, E. Machine sequencing via disjunctive graphs: An implicit enumeration approach. Operations Research, 17, 941-957. 1969.

[**64] Carlier, J; Pinson E.** An algorithm for solving the job-shop problem. Management Science, 35, 164-176. 1989.

[65] Ho, J; Chang. Y. A new heuristic for the n job, m-machine flow shop problem. European Journal of Operational Research, 52, 194-206. 1995.

[66] Johnson D; Garey M; Sethi R.. The complexity of flow-shop and job-shop scheduling Mathematics of Operations Research, 1, 117-129, 1976.

[67] Nawaz M; Enscore E; Ham I.. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Revista Omega, 11(1): 91-95, 1983.

[68] Kamoun H; Hall N. Scheduling in robotic cells: heuristics and cell design. Journa Operations Research. 47, 821-835. 1999.

[69] Sethi S; Sriskandarajah C; Sorger G. Sequencing of part and robot move in a robotics cell. The international Journal of flexible manufacturing systems. 4, 331-358. 1992.

[70] Hitomi K; Yoshimura M. Operations Scheduling for work transportation by industrial robots in automated manufacturing systems. Elsevier Science Publishers B. V 3, 131-139. Amsterdam, The Netherlands, 1986.

[71] Descotte Y; Latonbe J. GARI: A problem solver that plans how to machine mechanical parts Proceedings of the 7th International Joint Conference on Artificial Intelligence. Vancouver - Canada p.766-772, 1981.

[72] Nau D. SIPP Reference Manual Technical report. Computer Science Department University of Maryland. 1985.

[73] Matsushima, k; Sata T. The Integration of CAD and CAM by Aplications of Artificial Intelligence Techniques. Annals of CIRP 31, 329-332 1982.

[74] Davis B; Darbyshire Ll. The use of expert system in process planning. Annals of CIRP 33 303-306, 1984.

[75] Barkocy B; Zdeblick W. Aknowledge based system for machine operation planing. Proceedings of Autofact. 6:2, 11-1-25, 1984.

[76] Tsatsoulis C; Kashyap R. A system for knowledge - based process planing. International journal of Aplications of IA to Engineering, 3, 61-71, 1988.

[77] Fisher E. "FADES: knowledge based factility Design" Ph.D thesis Purdue University, west La Fayette IN. 1984.

[78] Wang H; Wysk R. An Expert system for Machining date Selection. Computers and Industrial Engineering. Pergamon Press. Elmsford, NY, 1988.