3. Scheduling

Los problemas de Planificación abarcan una variedad de problemas de Optimización Combinatoria en campos tales como operaciones de producción y despacho en la industria manufacturera, sistemas distribuidos y paralelos, logística y tráfico. [GJ79].

Según [Pin95] y [Bea97] las aplicaciones más corrientes en problemas de Planificación pueden dividirse en: **Routing, Scheduling y Packing**.

En **Routing**, uno de los problemas más conocidos es el del viajante de comercio o TSP (Travelling Salesman Problem), donde un viajante de comercio debe visitar cierto número de ciudades y retornar, de manera tal de visitar cada ciudad una sola vez y minimizar alguna variable de interés (distancia, costo, etc.).

El problema de **Packing** más simple es el conocido como Zero-One-Knapsack Problem. Aquí dada una mochila de cierta cantidad y un conjunto de ítems con cierto tamaño y un valor asociado, deben acomodarse dentro de la mochila de manera tal de maximizar el valor transportado. Varios problemas de la vida real están asociados a este tipo de problema, por ejemplo, la asignación de canales de comunicación a usuarios que pagan una tarifa diferente por el servicio.

Específicamente, los problemas de **Scheduling** consisten en la asignación de tareas a recursos limitados donde ciertos objetivos deben optimizarse (incluyendo multicriterios) y varias restricciones deben cumplirse. La combinación de recursos, tareas, objetivos y restricciones produce un incremento exponencial del espacio del problema. Perteneciendo a la clase de problemas NP-completos.

3.1. Problemas de Scheduling

En general, un *Scheduling* consiste en la asignación de tareas, a través del tiempo, cuando la disponibilidad de recursos es limitada, donde ciertos objetivos deben ser optimizados y varias restricciones deben ser satisfechas. Los problemas de *Scheduling* se aplican en las organizaciones y en la industria y en consecuencia tienen un fuerte impacto económico y social (ver ¯gura 3.1). El estudio de estos problemas data aproximadamente de 1950 donde investigadores de Ingeniería industrial, investigación operativa, y administradores desarrollaron nuevos enfoques y algoritmos que tienen como objetivo principal la reducción de los costos de producción en la industria [Leu04].

Además, se cuenta con un conjunto de datos asociados a un problema en particular, estos datos se describen a continuación:

- **Tiempo de Procesamiento** (Processing Time) p_{ij}. Representa el tiempo de procesamiento del Job j en la maquina i. El subíndice i se omite si el tiempo de procesamiento del job j no depende de la máquina o si el job j va a ser procesado en una máquina dada.

- **Fecha de disponibilidad** (Ready Date) r_j . Es el tiempo en que el job arriba al sistema, es decir, el momento en el cual el job está preparado para comenzar su procesamiento.
- **Fecha de Entrega** (Due Date) d_j. Representa la fecha de finalización o bien la fecha prometida de entrega del job al cliente.
- **Peso** (Weight) w_j. Es básicamente un factor de prioridad, denotando la importancia del job j relativa a los otros jobs en el sistema. Por ejemplo, este peso puede representar el costo actual de mantener el job en el sistema.

Figura 3.1: Scheduling de procesos en fábricas.

Un problema de *Scheduling* puede ser descrito a través de la tripleta α, β, γ El campo α describe el ambiente de máquina, el campo β provee detalles de las características y restricciones de procesamiento; el campo γ contiene el objetivo a ser optimizado.

Los ambientes de máquinas definen distintos tipos de problemas que pueden incluirse dentro de la categoría de *Scheduling*. A continuación, se detallan algunos posibles ambientes de máquina que pueden ser especificados en el campo α:

- **Máquina Única** (1) es el caso más simple de todos los posibles ambientes de máquina, los Jobs se procesan en un orden prefijado en una única máquina.
- **Máquinas Idénticas en Paralelo** (Pm) existen m máquinas idénticas en paralelo y n Jobs, cada Job j requiere una ¶única operación y puede procesarse en cualquiera de las m maquinas.
- **Máquinas en Paralelo con Distinta Velocidad** (Q_m) existen m máquinas en paralelo con distintas velocidades de procesamiento y n Jobs. La velocidad de la maquina i se denota como vi. El tiempo p_{ij} del Job j en la maquina i es, asumiendo que el Job es procesado solo en la maquina i, igual a p_j / v_i.
- **Máquinas no Relacionadas en Paralelo** (R_m) es una generalización del anterior donde existen m máquinas en paralelo y n Jobs. La máquina i procesa el Job j a

la velocidad vij. El tiempo pij del Job j en la maquina i, asumiendo que el Job se procesa únicamente en la máquina i, es igual a pi / vij.

- **Flow Shop (F_m)** existen m máquinas en serie y cada Job debe ser procesado en cada una de las m máquinas. Todos los Jobs tienen el mismo *routing*, es decir, primero tienen que procesarse en la máquina 1, luego deben procesarse en la máquina 2 y así sucesivamente. Luego de haberse completado en una máquina un Job se pone en cola de la próxima, la disciplina de cada cola es FIFO.
- **Flexible Flow Shop** (FF_s) es una generalización del Flow Shop y de un ambiente de máquinas en paralelo. En lugar de m máquinas en serie existen s etapas en serie (S1, S2,..,Ss), cada una con un número de máquinas en paralelo. Cada Job debe ser procesado en cada una de las s etapas y todos los Jobs tienen el mismo *routing*. Cada etapa funciona como un banco de máquinas paralelas. En cada etapa el Job j requiere sólo una de las máquinas y cualquier máquina es capaz de procesar cualquier Job.
- **Open Shop** (O_m) existen m máquinas y n Jobs. Cada Job debe procesarse en cada una de las m máquinas, sin embargo, algunos de estos tiempos pueden ser cero. No existen restricciones respecto del *routing* de cada Job a través del ambiente de máquinas.
- **Job Shop** (J_m) existen m máquinas y n Jobs. Cada Job tiene una ruta predeterminada que incluye a todas o a ciertas máquinas. Entre los Jobs Shops la diferencia esencial radica en permitir o no la recirculación de un Job en determinadas máquinas (es decir un Job puede visitar a una máquina dada más de una vez).

El campo β puede no existir o puede tener más de una componente. Algunas posibles restricciones pueden ser:

- **Tiempo de disponibilidad** (r_j), si este símbolo está presente en el campo, el Job j no debe empezar su procesamiento antes de su tiempo de disponibilidad rj ; si esta restricción no estuviera presente, significa que el Job j puede comenzar en cualquier momento.
- **Tiempos de setup dependientes de la secuencia** (s_{jk}) representan el tiempo de setup dependiente de la secuencia entre los Jobs j y k determina el tiempo de setup para el Job k si el Job k esta primero en la secuencia y sj0 el tiempo de limpieza después del Job j si el Job j es el último en la secuencia (s_{0k} y s_{j0} pueden ser cero). Si el tiempo de setup entre los Jobs j y k dependen de la máquina, entonces se añade el subíndice i, de la siguiente manera: s_{ijk}. Si no aparece ningún valor s_{jk} en el campo, se asume que todos los tiempos de preparación son cero o independientes de la secuencia.
- **Preemptions (prmp)** Preemptions implica que no es necesario mantener un Job en una máquina hasta que se complete. El *Schedule* permite interrupciones de procesamiento de un Job (*preempt*) en cualquier momento y colocar a un Job diferente en la máquina. La cantidad de procesamiento que el Job ha recibido antes de la interrupción no se pierde. Cuando el Job es colocado nuevamente en la máquina (o en otra máquina en el caso de máquinas en paralelo) únicamente necesita la máquina por el tiempo que le quedó remanente.

- **Restricciones de Precedencia (prec)**. Las restricciones de precedencia pueden aparecer en los ambientes de máquina única o máquinas en paralelo, requiriendo que uno o más Jobs sean completados antes de que otro comience su procesamiento.
- **Breakdowns (brkdwn)**. Breakdowns de máquinas significa que las máquinas no están continuamente disponibles. Los periodos en que las máquinas no están disponibles se asumen como Jos (por ejemplo, debido a cambios o programa de mantenimiento). Si hay un ¶umero de máquinas idénticas en paralelo, el número de máquinas disponibles en cualquier momento es una función del tiempo, es decir, mt.
- **Permutaciones (prmu)**. Una restricción que puede aparecer en un ambiente de Flow shop es que las colas enfrente de cada máquina operen de acuerdo con una disciplina FIFO. Esto implica que el orden (o permutación) en el cual los Jobs pasan a través de la primera máquina se mantiene en todo el sistema.
- **Blocking (Block)**. El fenómeno de blocking puede ocurrir en un ambiente de flow shop. Si un flow shop tiene un almacenamiento limitado entre dos máquinas sucesivas, puede pasar que cuando el almacenamiento está lleno la máquina no puede liberar un Job finalizado. Este fenómeno es conocido como **blocking**: el trabajo finalizado debe permanecer en la máquina, bloqueándola e imposibilitándola para procesar otra tarea.
- **No-wait (nwt)**. El requerimiento de no-wait es otro fenómeno que puede ocurrir en °ow shops. No se permite a los Jobs esperar entre dos máquinas sucesivas. Esto implica que el tiempo de comienzo del Job en la primera máquina debe ser demorado para asegurar que el Job puede atravesar todo el flow shop sin tener que esperar en ninguna máquina.
- Recirculaci¶on (recrc). La recirculación puede ocurrir en job shops, cuando un job puede visitar una máquina más de una vez.

Cualquier otra entrada que aparezca en el campo β es auto explicativa. Por ejemplo, pj=p implica que todos los tiempos de procesamiento son iguales, y dj=d implica que todas las fechas de entrega son iguales.

La Función Objetivo especifica el objetivo a ser optimizado en el problema y corresponde al campo γ en la especificación del problema. Esta es generalmente una función de los tiempos de completación de las tareas, la cual depende del schedule.

El tiempo de finalización de la operación del job j en la máquina i se denota por C_{ij} . El tiempo que el job j existe en el sistema es Cj (es el tiempo de finalización en la última máquina en la cual el job requirió procesamiento, es decir el tiempo en que completa su procesamiento).

En el caso de que la función objetivo dependa de las fechas de entrega, se puede tener otras funciones tales como Lateness, Tardiness y Earliness. A continuación se describe cada una de ellas:

- **Lateness Lj**: es la diferencia (positiva o negativa) entre el tiempo de ¯nalizaci¶on y el de entrega previsto (due date). Lj = Cj - dj. Donde dj es el tiempo en que la tarea debería finalizarse (due date).

- **Tardiness Tj**: Se corresponde con la Lateness de una tarea si ésta es positiva y cero en otro caso. Para esta función sólo se penalizan las tareas tardías, Tj = max 0, Lj .
- **Earliness Ej**: Se corresponde con la Lateness de una tarea si ¶esta es negativa y cero en otro caso. Para esta función sólo se penalizan las tareas tempranas, Ej = max 0, -Lj .

A continuación, se detallan algunas funciones objetivas a ser minimizadas:

- **Makespan**: Es el máximo tiempo de finalización y equivale al tiempo de finalización de la última tarea en abandonar el sistema. Si contamos con n tareas entonces se define como: Cmax = max j (Cj).
- **Maximun Lateness (Lmax)**. El máximo lateness, **Lmax**, se define como max L1,..,Ln Mide la peor violaci¶on a las fechas de entrega.
- **Total Weighted Completion Time (wj Cj)**. La suma de los tiempos de finalización ponderados de n Jobs da una idea de los costos totales de mantenimiento, o inventario incurridos por el Schedule.
- **Total Weighted Tardiness (wj Tj)**. Es una función de costo más general que Total weighted completion time.
- **Weighted Number of Tardy Jobs (wj Uj)**. Es una función objetivo que mide e número de tareas tardías, en la práctica que puede ser una medida registrada fácilmente. Weighted Flowtime (Weighted Completion Time): Fwt = wj Fj Donde Fj (Flowtime) es el tiempo en que el job j permanece en el sistema. Fj = Cj – rj
-

Ejemplos de las notaciones:

- **1, sjk, Cmax**. Es un ambiente de máquina ¶única con n Jobs sujetos a tiempos de setup dependientes de la secuencia, donde la función objetivo es de minimización del makespan. Este problema es también conocido como TSP, el problema del viajante de comercio.
- **Fm j pij =pj j wj Cj**. Un problema de Fow shop con m maquinases decir, m máquinas en serie con el tiempo de procesamiento del Job j en las m máquinas idéntica e igual a pj. El objetivo es encontrar un orden en el cual los n Jobs pasen a través del sistema de forma tal que se minimice la suma de los tiempos de finalización ponderados.
- **Jm , ,j Cmax**. Es un Job shop con m maquinas. No hay recirculación, por lo tanto, un Job visita cada máquina a lo sumo una vez. El objetivo es minimizar el makespan.

Muchos algoritmos eficientes han sido desarrollados para encontrar soluciones óptimas, aunque para tamaños pequeños de este tipo de problema. Por ejemplo, se pueden mencionar los trabajos de Jackson [Jac55], Johnson [Joh54] y Smith (1956). Con el advenimiento de la teoría de complejidad Cook [Coo71], muchas investigaciones sobre dicha temática se han desarrollado debido a la inherente dificultad para resolver esta clase de problemas de forma exacta. Muchos de los problemas de scheduling son computacionalmente complejos y el tiempo requerido para calcular una solución óptima se incrementa con el tamaño del problema [MP93] y [Pin95]. Además, se ha

demostrado, por cierto, que muchos problemas de Scheduling pertenecen a la clase de NP-Hard ([Bru04] y [JKL78]).

3.2. Flow Shop Scheduling Problem FSSP

La formulación más simple del problema de FSSP se compone de dos elementos principales: (1) un grupo de m máquinas y (2) un conjunto de n trabajos que se deben procesar en este grupo de máquinas. Cada uno de los n puestos de trabajo tiene el mismo orden para su secuencia de proceso. Cada trabajo se puede procesar en una y sólo una maquina a la vez (lo que significa que no hay división de trabajos) y cada máquina puede procesar solo un trabajo a la vez. Cada trabajo se procesa solo una vez en cada máquina. Las operaciones no son apropiables y los tiempos de preparación de las operaciones son independientes de las secuencias y, por tanto, pueden incluirse en el tiempo de procesamiento. Todo problema de scheduling consiste en especificar el orden y el momento de la tramitación de los puestos de trabajo en las maquinas, con uno o varios objetivos respetando los supuestos antes mencionados. En el problema de FSSP, el objetivo makespan puede definirse como el tiempo de finalización a la que todos los puestos de trabajo procesamiento completo o de manera equivalente como tiempo máximo de realización de los trabajos (ver figura 3.2).

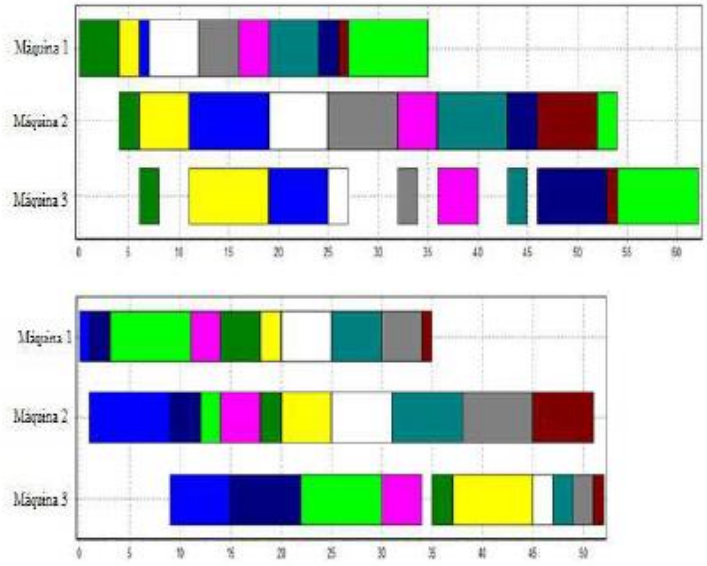

Figura 3.2: Función MakeSpam - Flow Shop Scheduling Problem

La mayor parte de la investigación al problema de FSSP (Flow Shop Scheduling Problem) se ha centrado en el desarrollo de soluciones basadas en permutaciones, ¡donde el espacio de búsqueda se reduce a n!, a diferencia de una representación binaria.

En el problema de FSSP Zero-Buffer donde las maquinas en las líneas de procesamiento tienen una memoria (buffers) de capacidad cero. Un trabajo que acaba de terminar en la máquina r no puede avanzar a la máquina s si esta máquina está aún

procesando el trabajo i de su predecesor en la secuencia de trabajos. El trabajo debe permanecer en la máquina de r, por lo tanto negar temporalmente máquina de trabajo r_i del sucesor en la secuencia de trabajo hasta el momento en que el trabajo que pueda avanzar a la maquina s.

Abadi y Sriskandarajah (1995) describen el problema de FSSP con bloqueo, aquel donde la secuencia de tareas no tiene bu®er intermedia, por lo tanto, un trabajo no puede dejar una maquina hasta la próxima maquina conectada esta libre, el trabajo (y la maquina también) se dice que está bloqueado. Aldowaisan y Allahverdi (1998) [AA98] describieron el caso en el que una vez que un trabajo comienza su procesamiento en la máquina 1 de la línea de producción debe continuar sin demora de procesamiento en cada una de las m máquinas en la línea de procesamiento. Más recientemente (2003), propusieron dos heur¶³sticas basadas en recocido simulado y un AG para FSSP con bloqueo para minimizar el makespan. El FSSP sin espera fue completamente descrito por Hall y Sriskandarajah (1996). La revisión hecha por Cheng et al. (2000) para el FSSP que incluyen tiempos de preparación es también muy adecuado para los lectores interesados. Allahverdi et al. (1999) han dividido a los problemas de Flow Shop (que implican consideraciones de configuración) en cuatro categorías: secuencia independiente sin ¶ánimo de lotes set-ups, dependientes de la secuencia no batch set-ups, secuencia por lotes independientes set-ups, y dependen de la secuencia de proceso por lotes set-ups.

Los problemas del FSSP híbridos han llamado mucho la atención de los investigadores en los últimos años. Zhixing et al. (2002) describen el FSSP híbrido sin espera de la siguiente manera. Dada una lista de centros de procesos, cada una con un número fijo de máquinas paralelas (de tal manera que al menos una fase contiene varias máquinas), hay n tareas para ser procesadas con el mismo orden de procesamiento ¯jo en los centros de la máquina, y en cada proceso se debe realizar en un máximo de una máquina en cada centro, y sin ningún tipo de interrupción en o entre las máquinas durante el proceso.

3.3. Formulación del Problema

El problema de secuenciamiento de Flow Shop (FSSP), provee de una forma conveniente de modelar los procesos de manufactura serial. El FSSP puede formularse de la siguiente manera: un conjunto de n tareas (1; 2; ..; n), con n > 1 tienen que ser procesadas en m maquinas (1; 2; ..;n) en el orden dado por el ¶³ndice de la máquina.

Cada tarea consiste en m operaciones y cada operación requiere de una maquina diferente. El tiempo de procesamiento de cada tarea i en la máquina j es fijo y está denotado por $\mathbf{t_{ij}}$ (i = 1; .. ; n; j = 1; .. ;m). La operación de cada tarea en cada máquina requiere de un período de tiempo sin interrupciones. Cada tarea puede ser procesada en una máquina en un momento y cada máquina puede procesar solamente una tarea a la vez. El objetivo es encontrar la secuencia óptima (permutación de todas las tareas) que tenga la suma mínima de tiempo de complementación de las tareas.

Dado C_{ji}; k) el tiempo de complementación de la tarea ji en la máquina k, y una permutación de tareas denotadas como (j_1; j_2; .. ; j_n). Entonces el tiempo de completación para un problema de FSSP con n Jobs tareas y m máquinas se calcula como:

La función objetivo para FSSP corresponde a la minimización del makespan dada en la ecuación Eq. 3.2.

$$
\begin{aligned}
C(j_1, 1) &= t_{j_1 1}, \\
C(j_1, k) &= C(j_1, k-1) + t_{j_1 k}, \quad k = 2, \ldots, m, \\
C(j_i, 1) &= C(j_{i-1}, 1) + t_{j_i 1}, \quad i = 2, \ldots, n, \\
C(j_i, k) &= \text{máx}\{C(j_{i-1}, k), C(j_i, k-1)\} + t_{j_i k}, \\
& \qquad i = 2, \ldots, n, \quad k = 2, \ldots, m. \qquad (3.1)
\end{aligned}
$$

El *makespan* se calcula como:

$$
C_{\text{máx}} = C(j_n, m). \qquad (3.2)
$$

3.4. Métodos exactos, heurísticas y metaheurísticas aplicados al FSSP

Uno de los primeros de los enfoques propuestos fue el de Programación Dinámica basados en los trabajos de Held y Karp para los problemas de FSSP con instancias de tamaño pequeño [HK62]. Otro enfoque muy desarrollado en la literatura es el de Branch and Bound [BLK78], [IS65], [Lom65], [DM94]). Además pueden verse variaciones de este enfoque como [Bak75], [DJ64] et al incluso modelos MILP para problemas SDST (sequence-dependent set-up times) y SSIST (separable, sequence-independent set-up times) [TS01].

Con el objetivo de superar optimalidad local y mejorar las soluciones factibles iniciales propuestas por las heurísticas antes mencionadas, se desarrolló una fuerte utilización de algoritmos denominados meta-heurísticas o heurísticas modernas. Muchos de estos algoritmos se han implementado para encontrar soluciones para FSSP. Dentro de los principales enfoques se incluyen: Recocido Simulado (Simulated Annealing-SA), Algoritmos Genéticos (Genetic Algorithms-GA) , Búsqueda Tabú (Tabu Search-TS), Enfoques Algoritmos Greedy, Búsqueda de profundidad variable, Hill Climbing y los métodos basados en la Colonia de Hormigas (Ant Colony Optimization-ACO), entre otros. Además, fueron desarrollados algoritmos híbridos, que combinan algunos de éstos.

El algoritmo de Recocido Simulado (Simulated Annealing - SA) fue introducido por Kirkpatrick et al. (1983) [KGV83] el cual simula el proceso de recocido de solidos y el proceso de resolución de problemas de optimización combinatoria. Originalmente este método fue desarrollado como un modelo de simulación para un proceso f¶³sico de recocido de la materia condensada (Metrópolis et al. 1953) [MRR+53]. Para mejorar los mecanismos de generación de nuevas soluciones se han considerado diferentes esquemas de perturbación de permutaciones. Sridhar y Rajendran (1993) hicieron uso de tres esquemas de perturbación (de intercambio adyacente, esquema de inserción y esquema de intercambio aleatorio). Osman y Potts (1989) [OP89] adoptaron intercambio de vecinos. Ogbu y Smith (1990) [OS90] eligieron la inserción y el intercambio de pares como el esquema de perturbación. Liu (1999) [Liu99] trabajo en tamaño de la vecindad y de su efecto en el Recocido Simulado para un FSSP. Peng Tian y otros [TMZ99] utilizaron seis esquemas de perturbación: (1) Intercambiando dos tareas adyacentes. (2) Intercambiando dos tareas. (3) Mover una solo tarea. (4) Mover

una tarea ulterior. (5) Inversión de una tarea ulterior. (6) Inversión y / o mover una subsecuencia de tareas.

Los Algoritmos Genéticos (GAs) fueron introducidos por Holland [Hol75] y están basados en el modelo de la evolución biológica. Holland, tenía como objetivo: mejorar la comprensión de los procesos de adaptación natural para el diseño de sistemas inteligentes artificiales con propiedades similares a los sistemas naturales. Goldberg [Gol88] describió un GA como un algoritmo de búsqueda aplicado a problemas de optimización. Por esta razón, metaheur¶³sticas como los algoritmos genéticos son ampliamente utilizados para encontrar buenas soluciones en tiempos razonables a los problemas NP-completos [CCF10], [JES09a].

La metaheur¶³stica de Búsqueda Tabú (Tabu Search - TS) fue introducida por Glover (1986)[Glo86], y luego ampliamente difundida por autores tales como Hansen (1986), Glover et al. (1993) y Glover y Laguna (1997). Este enfoque metaheur¶³stico, que ha sido aplicado para resolver diferentes problemas de optimización combinatoria, comienza con una solución inicial y luego se aplica un mecanismo de movimiento para buscar en la vecindad de la solución actual para elegir la más apropiada. Un vecino (un movimiento correspondiente de la solución actual) es admisible si no es tabú, o si un criterio de aspiración (tales como permitir que todos los movimientos que conducen a un vecino con una mejor función objetivo que se encontró hasta ahora) se cumple. Varios enfoques basados TS se han propuesto por diferentes investigadores para el problema de Flowshop, incluyendo Taillard (1990) [Tai90], Reeves (1993) [Ree93], Mocellin (1995) [Moc95], Nowicki y Smutnicki (1996) [NS96], y el famoso algoritmo ESPIRITU propuesto por Widmer y Hertz (1989) [WH89].

Los Sistemas de Colonia de Hormigas (Ant Colony System - ACS) propuesto por Dorigo y Gambardella (1997) [DG96] es una de las metaheurísticas más recientes y esperanzadoras para problemas de optimización combinatoria. La optimización basada en colonias de hormigas (Ant Colony Optimization - ACO) simula los hábitos de forrajeo colectivos de hormigas, de salir por la comida y llevarla de vuelta al nido. Las hormigas reales son capaces de encontrar el camino más corto desde una fuente de alimento a su nido sin usar señales visuales, ya que tienen una visión deficiente. Se comunican información relativa a las fuentes de alimentos a través de una esencia aromática. Esta sustancia química depositada por las hormigas en su viaje se llama feromona. A mayor cantidad de feromona en el camino da una hormiga, una estimulación más fuerte y por lo tanto una mayor probabilidad de seguirla. En esencia, se mueven al azar, pero cuando se encuentran con un rastro de feromonas, deciden (que depende de la cantidad de feromona en el camino potencial) si debe o no seguir, y si lo hacen, depositan su propia feromona en el rastro, lo que refuerza el camino. Por otra parte, con el tiempo, la feromona se evapora y como las hormigas que toman el camino más corto volverán a nido primero con los alimentos, el camino más corto tiene la mayor cantidad de feromona. En consecuencia, el camino más corto se convierte en una alternativa más atractiva para otras hormigas. ACO se ha aplicado para resolver diferentes tipos de problemas de optimizaci¶on combinatoria, incluyendo el problema del viajante de comercio (Travel Salesman Problem - TSP) [DG96], el Problema de Asignación Cuadrática (Quadratic Assignment Problem QAP) (Gambardella et al. 1999 [GTD99]), Job Shop Problem (Colorni y otros 1994) [CDMT94], VRP (Vehicle Routing Problem) (Bullnheimer y Hartl 1997) [BHS97], el problema de coloreado de grafos (Graph Colouring Problems-GCP) (Costa y Hertz 1997) [DA97], el problema del particionado (Kuntz et al. 1994)[KS94]

y la red de telecomunicaciones (Schoorderwoerd et al. 1997) [SHBR96]. Para la aplicación de ACO en el problema de Flowshop, T'kindt y otros (2003) [TGB03] propusieron un algoritmo ACO para minimizar el tiempo total finalización. Rajendran y Ziegler (2005) [RZ05] consideran el problema de la programación en permutación de Flowshop con el objetivo de minimizar el makespan, seguido por la consideración de la reducción al mínimo del tiempo de flujo total de puestos de trabajo.

3.5. Resumen

En la literatura de scheduling, los términos como regla de secuenciación, regla de despacho, regla de prioridad, o heurística se utilizan a menudo como sinónimos. Sin embargo, Gere [WSG66] ha hecho un intento de distinguir entre estos términos y considera que las normas de prioridad simplemente como a una técnica por la cual se le asigna un número (o valor) a cada trabajo en espera, de acuerdo con cierto método y el trabajo que posea un m¶³nimo de \valor" es seleccionado. Gere, define una heurística simplemente como una \regla de oro", mientras que una regla de programación puede constar de la combinación de una o más reglas de prioridad y o una o más heurísticas.

Los requisitos computacionales de métodos exactos, tales como la Programación Dinámica y Branch-and-Bound, entre otros, para resolver instancias grandes del FSSP, son ineficaces. En consecuencia, la investigación se enfocó en el desarrollo de heurísticas que si bien no resultaban ser métodos exactos proporcionaban buenos enfoques aproximados tales como el algoritmo de Palmer (1965) [Pal65], el algoritmo de CDS (1970) [CDS70], el algoritmo de Gupta (1971) [Gup71], el algoritmo NEH (1983) [NEH83], entre otros.

Framinan et al [FGL04] proporcionaron una revisión y una clasificación de las heurísticas más usadas para el FSSP y Hejazi y Sagha¯an [HS05b] presentaron un estudio exhaustivo de los principales resultados en este problema durante el período 1954-2004.