Eficiencia Energética y Robustez en Problemas de Scheduling

Joan Escamilla Fuster

Tesis depositada en cumplimiento de los requerimientos para obtener el título de Doctor por la Universitat Politècnica de València

Director: Dr. Miguel A. Salido Gregorio

Marzo 2016

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

A mi familia presente y futura.

ANTOINE DE SAINT-EXUPÉRY

A goal without a plan is just a wish.

ALLEN SAUNDERS

Life is what happens to you while you're busy making other plans.

ABRAHAM LINCOLN

Give me six hours to chop down a tree and I will spend the first four sharpening the axe.

Agradecimientos

En primer lugar, me gustaría dar las gracias a mi director Miguel Ángel Salido por toda su dedicación y ayuda durante el desarrollo de mi investigación. Por sus correcciones y consejos durante la realización de todos los artículos y de esta tesis.

También me gustaría dar las gracias a todos los compañeros del laboratorio con los cuales he compartido tantos buenos momentos como con Laura C., Laura I., Valentina, Flabio, César. En especial, a Mario con el cual también compartí una experiencia inolvidable en China, y como no nombrar al resto de integrantes de GH: Pastor, Jaume, Flores y Miquel. Con los que he compartido tanto durante estos últimos años como comidas, breaks, apuestas, discusiones, fiestas, confesiones, tesismómetro, chistes malos, bromas, etc. pero sobretodo risas y muchos recuerdos.

Por los momentos vividos en mi estancia en Nanjing me gustaría dar las gracias a Zheng Kun, Zelei Sang, Junxue, Dai Min, Haitao Zhang y al resto de compañeros del laboratorio de Nanjing y como no al profesor Dunbing Tang. Fueron muchas las experiencias vividas durante el tiempo que pasamos en China y también fueron muy enriquecedoras las estancias que ellos realizaron aquí en Valencia. Aprender su cultura y forma de vida me pareció una gran experiencia. Además, tuve la oportunidad de poder enseñarles algunas costumbres y la forma de vivir en España. En Camarena aún me siguen preguntando por lo chinos.

Me gustaría también dar las gracias a toda mi familia y amigos que aunque en muchas ocasiones no sabían muy bien lo que estaba haciendo, no han dudado en apoyarme y darme ánimos. Aunque también han estado realizando las típicas preguntas como: ¿Y tú allí qué haces?, ¿tú para cuando acabas eso? o ¿y luego qué vas a hacer?

Y como no, me gustaría agradecerle a mi mujer todo el apoyo recibido y los consejos que me ha dado tanto en los momentos en que las cosas salían como cuando no. Su ánimo y ayuda han sido muy importantes para la realización de este trabajo.

Abstract

Many industrial problems can be modelled as a scheduling problem where some resources are assigned to tasks so as to minimize the completion time, to reduce the use of resources, idle time, etc. There are several scheduling problems which try to represent different kind of situations that can appear in real world problems. Job Shop Scheduling Problem (JSP) is the most used problem. In JSP there are different jobs, every job has different tasks and these tasks have to be executed by different machines. JSP can be extended to other problems in order to simulate more real problems. In this work we have used the problem job shop with operators $JSO(n, p)$ where each task must also be assisted by one operator from a limited set of them. Additionally, we have extended the classical JSP to a job-shop scheduling problem where machines can consume different amounts of energy to process tasks at different rates (JSMS). In JSMS operation has to be executed by a machine that has the possibility to work at different speeds. Industrial processes involve a large number of task scheduling problems. The two problems extensions proposed allow to represent a large number of combinatorial problems in industrial processes.

Scheduling problems consider optimization indicators such as processing time, quality and cost. However, governments and companies are also interested in energy-consumption due to the rising demand and price of fuel, the reduction in energy commodity reserves and growing concern about global warming. In this thesis, we have developed new metaheuristic search techniques to model and solve the JSMS problem. A comparative study was carried out to analyse the behaviour of our algorithm against a well-known solver: IBM ILOG CPLEX CP Optimizer.

Robustness is a common feature in real life problems. A system persists if it remains running and maintains his main features despite continuous perturbations, changes or incidences. We have developed a technique to solve the $JSO(n, p)$ problem with the aim of obtaining optimized and robust solutions.

We have developed a dual model to relate optimality criteria with energy consumption and robustness/stability in the JSMS problem. This model is committed to protect dynamic tasks against further incidences in order to obtain robust and energy-aware solutions.

The proposed dual model has been evaluated with a memetic algorithm to compare the behaviour against the original model.

In the JSMS problem there are a relationship between Energy-efficiency, Robustness and Makespan. Therefore, the relationship between these three objectives is studied. Analytical formulas are proposed to analyse the relationship between these objectives. The results show the trade-off between makespan and robustness, and the direct relationship between robustness and energy-efficiency. To reduce the makespan and to process the tasks faster, energy consumption has to be increased. When the energy consumption is low it is because the machines are not working at highest speed. So, if an incidence appears, the speed of these machines can be increased in order to recover the time lost by the incidence. Hence robustness is directly related with energy consumption. Additionally, robustness is also directly related with makespan because, when makespan increases, there are more gaps in the solution, these incidences can be absorbed by these natural buffers.

The combination of robustness and stability gives the proposal an added value due to since an incidence cannot be directly absorbed by the disrupted task and it can be repaired by involving only a small number of tasks. Thus, the original solution can be recovered to maintain feasible the rest of the original schedule. If recovery is not possible, rescheduling must be applied. In this work we propose two different techniques to manage rescheduling over the JSMS problem. Given an incidence, a match-up technique is committed to determine the time point of the schedule when the original solution is recovered and a rescheduled solution is obtained. Then, a memetic algorithm is proposed to search for an energy efficient solution in the established rescheduling zone. An extensive study was carried out to analyse the behaviour of the proposed techniques. To this end, some incidences were simulated over some well-known benchmarks. The proposed match-up technique maintained a good performance and many instances were recovered in an efficient way. Finally, most of the rescheduling solutions were improved to save more energy consumption.

This work represents a breakthrough in the state of the art of scheduling problems and in particular the problem where energy consumption can be controlled by the rate of the machines.

Resumen

Muchos de los problemas industriales se pueden modelar como un problema de scheduling donde algunos recursos son asignados a tareas a fin de minimizar el tiempo de finalización, para reducir el uso de los recursos, el tiempo de inactividad, etc. Existen varios tipos de problemas de scheduling que intentan representar diferentes situaciones que pueden aparecer en los problemas del mundo real. Job-Shop scheduling (JSP) es el problema más utilizado. En JSP hay diferentes trabajos, cada trabajo tiene diferentes tareas y estas tareas tienen que ser ejecutadas por diferentes máquinas. JSP puede ser extendido a otros problemas con el fin de simular una mayor cantidad de problemas reales. En este trabajo se ha utilizado el problema job shop scheduling con operadores $JSO(n, p)$, donde cada tarea también debe ser asistida por un operador de un conjunto limitado de ellos. Además, hemos ampliado el clásico problema JSP a un problema donde las máquinas pueden consumir diferentes cantidades de energía al procesar tareas a diferentes velocidades (JSMS). En JSMS las operaciones tiene que ser ejecutadas por una máquina que tiene la posibilidad de trabajar a diferentes velocidades. Los procesos industriales implican un gran número de problemas de scheduling de tareas. Los dos problemas extendidos pueden representar un gran número de problemas combinatorios en los procesos industriales.

Los problemas de scheduling consideran indicadores de optimización tales como: el procesamiento de tiempo, la calidad y el coste. Sin embargo, hoy en día los gobiernos y los empresarios están interesados también en el control del consumo de energía debido al aumento de la demanda y del precio de los combustibles, la reducción de las reservas de materias primas energéticas y la creciente preocupación por el calentamiento global. En esta tesis, hemos desarrollado nuevas técnicas de búsqueda metaheurística para modelar y resolver el problema JSMS. Además, se lleva a cabo un estudio comparativo para analizar el comportamiento de nuestro algoritmo contra un resolutor conocido: IBM ILOG CPLEX CP Optimizer.

La robustez es una característica común en los problemas de la vida real. Un sistema persiste si permanece en funcionamiento y mantiene sus principales características a pesar de las perturbaciones continuas, cambios o incidencias. Hemos desarrollado una técnica

para resolver el problema $JSO(n, p)$ con el objetivo de obtener soluciones robustas y optimizadas.

Hemos desarrollado un modelo dual para relacionar los criterios de optimalidad con el consumo de energía y la robustez/estabilidad en el problema JSMS. Este modelo se ha desarrollado para proteger a las tareas dinámicas contra incidencias, con el fin de obtener soluciones sólidas y que tengan en cuenta el consumo de la energía. El modelo dual propuesto ha sido evaluado con un algoritmo memético para comparar el comportamiento frente al modelo original.

En el problema JSMS hay una relación entre la eficiencia energética, la robustez y el makespan. Por lo tanto, se estudia la relación entre estos tres objetivos. Se desarrollan fórmulas analíticas para representar la relación estimada entre estos objetivos. Los resultados muestran el equilibrio entre makespan y robustez, y la relación directa entre la robustez y eficiencia energética. Para reducir el makespan, el consumo de energía tiene que ser aumentado para poder procesar las tareas más rápido. Cuando el consumo de energía es bajo, debido a que las máquinas no están trabajando a la velocidad más alta, si una incidencia aparece, la velocidad de estas máquinas puede ser aumentada con el fin de recuperar el tiempo perdido por la incidencia. Por lo tanto la robustez está directamente relacionada con el consumo de energía. Además, la robustez también está directamente relacionada con el makespan porque, cuando el makespan aumenta hay más huecos en la solución, que en caso de surgir incidencias, estas pueden ser absorbidas por estos buffers naturales.

La combinación de robustez y estabilidad da un valor añadido debido a que si una incidencia no puede ser absorbida directamente por la tarea interrumpida, esta puede ser reparada mediante la participación un pequeño número de tareas. Por lo tanto, la solución original se puede recuperar para mantener viable el schedule original. Cuando la recuperación no es posible, se debe aplicar rescheduling. En este trabajo se proponen dos técnicas diferentes para gestionar el rescheduling sobre el problema JSMS. Así que, dada una incidencia, una técnica match-up es usada para determinar el punto de tiempo donde el schedule recupera la solución original y se obtiene una solución replanificada. A continuación, se propone un algoritmo memético para buscar una solución eficiente energéticamente en la zona de replanificación establecida. Un extenso estudio se ha llevado a cabo para analizar el comportamiento de las técnicas propuestas. Para ello, se han simulado incidencias sobre algunas instancias conocidas. La técnica match-up propuesta mantiene un buen rendimiento y muchos casos han sido recuperados de una manera eficiente. Por último, la mayoría de las soluciones replanificadas han sido mejoradas para ahorrar en el consumo de energía.

Este trabajo representa un avance en el estado del arte en los problemas de scheduling y, en particular, en el problema donde el consumo de energía puede ser controlado por la velocidad de las máquinas.

Resum

Molts dels problemes industrials es poden modelar com un problema de scheduling on alguns recursos són assignats a tasques a fi de minimitzar el temps de finalització, per a reduir l'ús dels recursos, el temps d'inactivitat, etc. Existeixen diversos tipus de problemes de scheduling que intenten representar diferents situacions que poden aparèixer en els problemes del món real. Job-Shop scheduling (JSP) és el problema més utilitzat. En JSP hi ha diferents treballs, cada treball té diferents tasques i aquestes tasques han de ser executades per diferents màquines. JSP pot ser estès a altres problemes amb la finalitat de simular una major quantitat de problemes reals. En aquest treball s'ha utilitzat el problema job shop scheduling amb operadors $JSO(n, p)$, on cada tasca també ha de ser assistida per un operador d'un conjunt limitat d'ells. A més, hem ampliat el clàssic problema JSP a un problema on les màquines poden consumir diferents quantitats d'energia per a processar tasques a diferents velocitats (JSMS). En JSMS les operacions ha de ser executades per una màquina que té la possibilitat de treballar a diferents velocitats. Els processos industrials impliquen un gran nombre de problemes de scheduling. Els dos problemes estesos poden representar un gran nombre de problemes combinatoris en els processos industrials.

Els problemes de scheduling consideren indicadors d'optimització tals com: el processament de temps, la qualitat i el cost. No obstant açò, avui en dia els governs i els empresaris estan interessats també amb el control del consum d'energia a causa de l'augment de la demanda i del preu dels combustibles, la reducció de les reserves de matèries primeres energètiques i la creixent preocupació per l'escalfament global. En aquesta tesi, hem desenvolupat noves tècniques de cerca metaheurística per a modelar i resoldre el problema JSMS. A més, es duu a terme un estudi comparatiu per a analitzar el comportament del nostre algorisme contra un resolutor conegut: IBM ILOG CPLEX CP Optimizer.

La robustesa és una característica comuna en els problemes de la vida real. Un sistema persisteix si continua en funcionament i manté les seues principals característiques malgrat les pertorbacions contínues, canvis o incidències. Hem desenvolupat una tècnica per a resoldre el problema $JSO(n, p)$ amb l'objectiu d'obtenir solucions robustes i optimitzades.

Hem desenvolupat un model dual per a relacionar els criteris de optimalidad amb el consum d'energia i la robustesa/estabilitat en el problema JSMS. Aquest model s'ha desenvolupat per a protegir a les tasques dinàmiques contra incidències, amb la finalitat d'obtenir solucions sòlides i que tinguen en compte el consum de l'energia. El model dual proposat ha sigut evaluat amb un algorisme memético per a comparar el comportament front un model original.

En el problema JSMS hi ha una relació entre l'eficiència energètica, la robustesa i el makespan. Per tant, s'estudia la relació entre aquests tres objectius. Es desenvolupen fórmules analítiques per a representar la relació estimada entre aquests objectius. Els resultats mostren l'equilibri entre makespan i robustesa, i la relació directa entre la robustesa i l'eficiència energètica. Per a reduir el makespan, el consum d'energia ha de ser augmentat per a poder processar les tasques més ràpid. Quan el consum d'energia és baix, a causa que les màquines no estan treballant a la velocitat més alta, si una incidència apareix, la velocitat d'aquestes màquines pot ser augmentada amb la finalitat de recuperar el temps perdut per la incidència. Per tant la robustesa està directament relacionada amb el consum d'energia. A més, la robustesa també està directament relacionada amb el makespan perquè, quan el makespan augmenta hi ha més buits en la solució, que en cas de sorgir incidències, aquestes poden ser absorbides per els buffers naturals.

La combinació de robustesa i estabilitat dóna un valor afegit a causa de que si una incidència no pot ser absorbida directament per la tasca interrompuda, aquesta pot ser reparada mitjançant la participació d'un xicotet nombre de tasques. Per tant, la solució original es pot recuperar per a mantenir viable el schedule original. Quan la recuperació no és possible, s'ha d'aplicar rescheduling. En aquest treball es proposen dues tècniques diferents per a gestionar el rescheduling sobre el problema JSMS. Així que, donada una incidència, una tècnica match-up és usada per a determinar el punt de temps on el schedule recupera la solució original i s'obté una solució replanificada. A continuació, es proposa un algorisme memético per a cercar una solució eficient energèticament en la zona de replanificación establida. Un extens estudi s'ha dut a terme per a analitzar el comportament de les tècniques propostes. Per això, s'han simulat incidències sobre algunes instàncies conegudes. La tècnica match-up proposta manté un bon rendiment i molts casos han sigut recuperats d'una manera eficient. Finalment, la majoria de les solucions replanificadas han sigut millorades per a estalviar en el consum d'energia.

Aquest treball representa un avanç en l'estat de l'art en els problemes de scheduling i, en particular, en el problema on el consum d'energia pot ser controlat per la velocitat de les màquines.

Acrónimos

AI Inteligencia artificial (Artificial Intelligence)

ATC Coste del Retraso Aparente (Apparent Tardiness Cost)

B&B Ramificación y poda (Branch and Bound)

BJ Salto Hacia Atrás (Backjumping)

BP British Petroleum

BT Vuelta atrás (Backtracking)

BTU Unidad Térmica Británica (British Thermal Unit)

COVERT Coste en el Tiempo (COst oVER Time)

CP Camino Crítico (Critical Path)

CP Optimizer Herramienta IBM ILOG CPLEX CP Optimizer

CSOP Problema de Satisfacción y Optimización de Restricciones (Constraint Satisfaction Optimization Problem)

CSP Problema de Satisfacción de Restricciones (Constraint Satisfaction Problem)

EDD Instante de Finalización más Temprano (Earliest Due Date)

ERD Instante de Inicio más Temprano (Earliest Receipt Date)

FC Comprobacion hacia delante (Forward-checking)

GA Algoritmo Genético (Genetic Algorithm)

GA+LS Algoritmo Memético (Memetic Algorithm)

GHG Gas de Efecto Invernadero (Greenhouse Gas)

GRASP Procedimiento de Búsqueda Voraz Adaptativo Probabilista (Greedy Randomized Adaptive Search Procedure)

GT Prueba y Error (Generate-and-Test)

IP Programacion entera (Integer Program)

JOX Cruce Basado en el orden de trabajos (Job-based Order Crossover)

JML Job con más carga (Job with More Load)

JMT Job con más tareas (Job with More Tasks)

JSMS Problema Job-Shop Scheduling con Máquinas a distintas Velocidades (Job-shop Scheduling problem with Machines at different Speeds)

JSO(n, p) Problema Job-Shop Scheduling con Operadores (Job-shop Scheduling problem with Operators)

JSP Problema Job-Shop Scheduling (Job-Shop Scheduling Problem)

LFJ Trabajo menos Flexible (Least Flexible Job)

LNS Número Mayor de Sucesores (Largest Number of Successors)

LP Programación Lineal (Linear Program)

LPT Tiempo de Procesamiento más Largo (Longest Processing Time)

LR Relajación Lagrangiana (Lagrangian relaxation)

LS Búsqueda Local (Local Search)

MML Máquina con más carga (Machine with More Load)

MMT Máquina con más tareas (Machine with More Tasks)

MPS Plan Maestro de Producción (Master Production Schedule)

MRP Sistema de Planificación de Requerimientos de Materiales (Material requirements planning)

MS Mínima Holgura (Minimum slack)

NC Nodo-Consistencia (Node-Consistency)

NWAUF Función “Normalized Weighted Additive Utility”

SA Enfriamiento Simulado (Simulated Annealing)

SIRO Servicio en Orden Aleatorio (Service In Random Order)

SPT Tiempo de procesamiento más Corto (Shortest Processing Time)

SQNO Cola más Corta en la Siguiente Operación (Shortest Queue at Next Operation)

SST Tiempo de Setup más Corto (Shortest Setup Time)

TS Búsqueda Tabú (Tabu Search)

UPV Universitat Politècnica de València

WSPT Tiempo de procesamiento Ponderado más Corto (Weighted Shortest Processing Time)

XML Lenguaje de Marcas Extensible (eXtensible Markup Language)

Índice general

Abstract IX

Resumen XI

Resum XIII

Acrónimos XV

Índice General XIX

Lista de Figuras XXII

Lista de Tablas XXIV

1 Introducción 1

1.1 Generalidades 1
1.2 El problema de scheduling 2
1.3 Motivación 4
1.4 Estructura del Trabajo 7

2 El Problema de Job Shop Scheduling 11

2.1 Introducción 11

2.2 Definiciones y Conceptos Básicos . . . 12
2.3 Métodos de resolución exactos . . . 14
2.3.1 Programación Dinámica . . . 14
2.3.2 Branch and Bound . . . 16
2.3.3 Branch and Cut . . . 18
2.3.4 Programación con restricciones . . . 19
2.4 Métodos de resolución heurísticos . . . 20
2.4.1 Desplazar el Cuello de Botella . . . 20
2.4.2 Reglas de Procesamiento Básico . . . 22
2.4.3 Reglas de Procesamiento Compuesto . . . 23
2.5 Técnicas de resolución Metaheurísticas . . . 25
2.5.1 Enfriamiento Simulado (Simulated Annealing) . . . 26
2.5.2 Búsqueda Tabú . . . 27
2.5.3 Procedimiento de Búsqueda Voraz Adaptativo Probabilista (Greedy Randomized Adaptive Search Process) . . . 28
2.5.4 Algoritmo Genético . . . 28
2.6 Técnicas de Resolución de un CSP . . . 30
2.6.1 Técnicas de consistencia . . . 31
2.6.2 Técnicas de búsqueda . . . 32
2.6.3 Técnicas híbridas . . . 33
2.6.4 Ejemplos de Aplicación . . . 35
2.7 Conclusiones . . . 36

3 El Problema de Job Shop Scheduling con Operadores 37
3.1 Introducción . . . 37
3.2 Robustez y Estabilidad en Scheduling . . . 38
3.3 Modelización del problema $JSO(n, p)$ como CSOP . . . 39
3.3.1 Fase de Modelado . . . 40
3.3.2 Fase de Resolución . . . 43
3.4 Desarrollo de Técnica de 3 Pasos . . . 44
3.4.1 Modelado y Resolución CSOP . . . 44
3.4.2 Post-Proceso (PP) . . . 45
3.4.3 Método de Distribución de Buffers . . . 47
3.4.4 Ejemplo . . . 48

3.5 Evaluación 49
3.5.1 Evaluación del Modelo CSOP 49
3.5.2 Evaluación del Modelo de Técnica de 3 Pasos. 51
3.6 Conclusiones 54

4 Eficiencia Energética en Job Shop Scheduling 57
4.1 Introducción 57
4.2 Modelos Energéticos en Scheduling 58
4.3 Modelo de Job Shop Scheduling con Consumo de Energía (JSMS) 59
4.3.1 Generación de banco de pruebas (Benchmarks) en JSMS 60
4.3.2 IBM ILOG CPLEX CP Optimizer 63
4.4 Diseño de Algoritmo Genético para JSMS. 63
4.4.1 Codificación y Decodificación del Cromosoma 64
4.4.2 Población inicial 65
4.4.3 Cruce 67
4.4.4 Mutación 69
4.4.5 Función Fitness. 70
4.4.6 Búsqueda Local 71
4.4.7 Algoritmo Memético 74
4.5 Evaluación de JSMS. 74
4.6 Conclusiones 81

5 Robustez y Estabilidad en Job Shop Scheduling 83
5.1 Introducción 83
5.2 Robustez en Job Shop Scheduling con Consumo de Energía (JSMS) 84
5.2.1 Análisis de Incidencias 85
5.2.2 Modelado dual del problema JSMS 86
5.2.3 Resolución del Problema JSMS 87
5.2.4 Adaptación de la Solución JSMS 88
5.3 Evaluación de Robustez en JSMS 88
5.4 Conclusiones 93

6 Análisis de Parámetros en JSMS: Robustez, Consumo Energético y Makespan 95

6.1 Introducción . . . 95
6.2 Makespan versus Consumo Energético . . . 96
6.3 Robustez versus Consumo Energético . . . 97
6.4 Robustez versus Makespan . . . 99
6.5 Análisis General . . . 101
6.6 Conclusiones . . . 104

7 Rescheduling en JSMS 105

7.1 Introducción . . . 105
7.2 Rescheduling y Recuperabilidad . . . 107
7.3 Diseño de Algoritmo para Rescheduling en JSMS . . . 109
7.3.1 Desarrollo de una Técnica Match-up . . . 109
7.3.2 Rescheduling . . . 112
7.4 Evaluación . . . 114
7.4.1 Incidencias y Robustez . . . 114
7.4.2 Evaluando la técnica Match-up . . . 115
7.4.3 Evaluando el algoritmo memético para rescheduling . . . 117
7.5 Conclusiones . . . 119

8 Conclusiones y Trabajos Futuros 121

8.1 Conclusions . . . 121
8.2 Publications List . . . 123

Referencias 125

Índice de figuras

1.1. Consumo total mundial de energía por sector (2009) 5

1.2. Consumo de energía mundial 1990-2040 (trillones de BTU) (Administration 2013) 6

2.1. Grafo Dirigido para Job Shop con el makespan como el objetivo 13

2.2. Flujo de los Trabajos 13

2.3. Clasificación de metaheurísticas 26

2.4. Esquema de Técnicas utilizadas en la resolución de CSPs 30

3.1. Instancia parcial de un JSP representado en XCSP. 41

3.2. Soluciones a un JSP: Una solución óptima y soluciones robustas. 48

3.3. Tiempo computacional para calcular paso 1 y paso 2 + paso 3 53

4.1. Codificación de un cromosoma para el problema JSP 64

4.2. Codificación de un cromosoma para el problema JSMS 65

4.3. Resultado del cruce entre dos padres 68

4.4. Ejemplo de dos cromosomas decodificados antes y después de aplicar la búsqueda local 72

4.5. Valores de F para CP, GA y GA+LS en las instancias Taillard de 50 y 100 trabajos. 76

4.6. Valores de F para CP, GA y GA+LS en las instancias Watson50, Watson100 y Watson200. 79

5.1. Los tres diferentes casos para la asignación del tiempo de buffer 87

5.2. Transformación dual de una tarea del schedule 87

6.1. Frontera pareto para la optimización del makespan y el consumo de energía. 97

6.2. Frontera pareto para la optimización de la robustez y el consumo de energía 98

6.3. Frontera pareto para la optimización de la robustez y el makespan. 100

6.4. Análisis General de las interrupciones del 40 % del máximo tiempo de procesamiento (%incid=40). 102

6.5. Interrupciones desde el 10 % hasta el 40 % del máximo tiempo de procesamiento. 103

7.1. Rescheduling por recuperación 108

7.2. Relación entre tareas por trabajo y máquina 110

7.3. Solución original, solución recuperada, solución mejorada después de una incidencia y una búsqueda en anchura 113

7.4. Incidencias absorbidas para los distintos tipos de instancias 115

7.5. Tiempo de ejecución medio de la técnica Match-up para las diferentes instancias 116

7.6. Comparativa entre MUp y Rec para las instancias 10x5 119

Índice de tablas

2.1. Resumen de las reglas de procesamiento básico que han sido estudiadas . 24

3.1. Makespan y tiempo computacional ($p_i = [1, 10]$) 50
3.2. Makespan y tiempo computacional ($p_i = [1, 50]$) 50
3.3. Makespan y tiempo computacional ($p_i = [1, 100]$) 50
3.4. Makespan, Número de buffers y Robustez 52
3.5. Porcentaje de robustez para incidencias grandes ($p_i = [1, 100]$) 54

4.1. Velocidad de las máquinas para la población inicial 67
4.2. Resultados de las instancias Agnetis <3_5_10>. 75
4.3. Resultados de las instancias Agnetis <7_10_100>. 75
4.4. Resultados de las instancias Agnetis <3_25_100>. 76
4.5. Resultados de las instancias Taillard50. 77
4.6. Resultados de las instancias Taillard100. 77
4.7. Comparación entre los valores de la F obtenidos por el CP Optimizer para los valores 1 y 4 en el parámetro cp.param.Worker. 78
4.8. Resultados de las instancias Watson50. 78
4.9. Resultados de las instancias Watson100. 80
4.10. Resultados de las instancias Watson200. 80

5.1. Datos del histórico de las incidencias 85

5.2. Estudio comparativo entre el modelo original y el modelo dual con la duración media de las incidencias 89

5.3. Comparativa entre el modelo Dual con IncMax y el modelo dual con IncMEd 91

5.4. Estudio de estabilidad 92

6.1. Makespan y consumo de energía para las instancias $< m, v_{max}, p >$ . . . 96

6.2. Consumo de energía y robustez para las instancias $< m, v_{max}, p >$ 99

6.3. Makespan y robustez para las instancias $< m, v_{max}, p >$ 101

7.1. Número de incidencias absorbidas por robustez (Rb), por punto match-up (MUp) o solución no recuperada (Mk) para diferentes instancias 116

7.2. Número de incidencias absorbidas por robustez (Rb), por punto match-up (MUp) o solución no recuperada (Mk) para diferentes instancias 117

7.3. Replanificando los resultados obtenidos por la técnica de Match-up 118

Capítulo 1

Introducción

1.1 Generalidades

Muchos procesos industriales pueden ser representados como un problema de scheduling, el cual consiste en la asignación de recursos a tareas que se intentan ejecutar en la mayor brevedad posible, utilizando el menor número de recursos, reduciendo los tiempos de espera, etc. Pero en ciertas ocasiones puede interesar que los procesos se ejecuten con seguridad y que ante cierto imprevisto o retraso la solución al problema de scheduling pueda continuar siendo válida (robustez). La robustez puede ser necesaria en problemas críticos donde un cambio en el plan original puede producir grandes costes económicos, la pérdida de tiempo, la necesidad de cancelar tareas, la pérdida de recursos, etc. Es por ello que para ciertos problemas industriales puede ser necesario tener en cuenta criterios adicionales como la robustez a la hora de planificar.

Además de obtener soluciones optimizadas y robustas, hoy en día, está apareciendo el interés por otros criterios de optimización, como es la eficiencia energética. El ahorro de energía produce un descenso en los costes de producción pero además contribuye a la reducción de los problemas medioambientales. Puesto que, problemas como el efecto invernadero, el rápido agotamiento de los recursos no renovables (por ejemplo, gas, petróleo, carbón), el descenso de la biodiversidad, etc. Son problemas que preocupan cada vez más a gobiernos y asociaciones.

1.2 El problema de scheduling

Los problemas de scheduling se pueden definir como problemas de optimización donde la finalidad es asignar recursos a ciertas tareas a lo largo del tiempo. La principal meta es la optimización de uno o más objetivos (Baker 1974). Los problemas de scheduling pueden presentarse en muchos dominios tales como:

- Sistemas industriales: Pueden aparecer problemas que deben de ser resueltos simultáneamente y de una forma optimizada, como los problemas de cadenas de montaje o sistemas guiados automáticamente (Silva, Moretti y Azevedo 2014).
- Sistemas informáticos: Para hacer que el uso de un ordenador multi-proceso sea óptimo, varias tareas deberán de ser ejecutadas en paralelo (Saifullah y col. 2013).
- Sistemas administrativos: Pueden aparecer problemas de gestión de citas, asignación general de recursos o la planificación de horarios (Ajanovski 2013).
- Sistemas de transporte: Aparecen problemas como el diseño de rutas óptimas para vehículos o la asignación de conductores (Zhong y col. 2013).

Cada uno de los distintos problemas de scheduling puede presentar ciertas peculiaridades. Pero en definitiva todos pueden ser descritos como una serie de tareas. Las cuales deben de ser asignadas a un recurso para ser ejecutadas en un intervalo de tiempo. Para representar los problemas de scheduling se usan ciertos problemas generales que se adaptan a problemas de la realidad. Existen distintos tipos de scheduling y cada uno de ellos puede presentar distintas restricciones y distintos criterios de optimización. Generalmente un problema de scheduling se compone con los siguientes elementos:

- Trabajos o Jobs
- Tareas u Operaciones
- Recursos o Máquinas
- Objetivos

El termino trabajo es usado en un problema de scheduling para identificar a una actividad o actividades que tienen que ser realizadas. Dicha actividad puede tener limitaciones de tiempo, como el instante de tiempo donde puede empezar o el tiempo máximo en el que debe ser finalizada.

Un trabajo puede estar compuesto por una o más tareas. Para la realización de cada tarea suele ser necesario el uso de uno o más recursos. Las tareas que pertenecen a un mismo trabajo pueden tener un orden de precedencia. Cada tarea puede tener un tiempo de proce-

samiento definido o puede ser variable dependiendo del recurso utilizado para ejecutarla y de sus propiedades.

En ocasiones, las tareas deben de ser ejecutadas por un recurso/máquina en concreto, pero en otras ocasiones puede haber varios recursos o modos disponibles para ejecutar una misma tarea. Habitualmente, una máquina solo puede ejecutar una tarea a la vez (máquinas simples), pero podemos encontrar problemas donde en una misma máquina se pueden ejecutar más de una tarea al mismo tiempo (máquinas paralelas). Existen muchos problemas de scheduling, de los que destacamos los tres tipos más importantes:

- Open-Shop: Un conjunto determinado de trabajos deben de ser procesados en un tiempo determinado y con unos recursos en concreto, en un orden arbitrario y el objetivo es determinar el momento en que cada trabajo se va a procesar en cada recurso (Gonzalez y Sahni 1976).
- Job-Shop: Un conjunto determinado de trabajos deben de ser procesados en un tiempo determinado y con unos recursos en concreto, cada recurso debe ser usado para ejecutar cada tarea en un orden determinado (Graham 1966).
- Flow-Shop: Es una extensión del Job-Shop, pero con la singularidad de que los recursos deben de ser utilizados en el mismo orden para todos los trabajos. El número de tareas debe de ser el mismo para todos los trabajos (Garey, Johnson y Sethi 1976).

Los objetivos a minimizar pueden ser varios. A continuación se exponen y se explican brevemente algunos de ellos (Hastings y Yeh 1990):

- Makespan: Diferencia de tiempo entre el inicio y el final de una secuencia de trabajos.
- Tardiness: Retraso en el procesamiento para cada trabajo.
- Lateness máxima: Máxima desviación del tiempo de finalización de los trabajos.
- Total flow time: Suma del tiempo de finalización de todos los trabajos.

Todos los objetivos presentados están relacionados con el tiempo, ya que es el objetivo que normalmente más se optimiza. Aunque el objetivo del tiempo no es el único, ya que en ciertas ocasiones puede resultar interesante minimizar el coste económico u optimizar los recursos usados.

El uso de los recursos suele ir asociado a un coste (Fox y Sadeh 1990), por lo tanto, si se reduce o se optimiza el uso de los recursos, se puede producir un ahorro en el coste. El

coste del recurso puede estar relacionado con el mantenimiento, la energía que utiliza, el operario encargado de manejar el recurso, etc.

1.3 Motivación

Como se ha comentado en la sección anterior, en la mayoría de los problemas de scheduling el objetivo a optimizar está relacionado con el tiempo, pero pueden aparecer otros objetivos interesantes. En el sector industrial aparecen un gran número de problemas de scheduling, por lo general los problemas de scheduling tratan los objetivos relacionados con la eficiencia de la producción (makespan, el retraso máximo, el número trabajos retrasados, etc.). Sin embargo, se puede reducir al mínimo el consumo de energía durante el proceso de producción, generando una fabricación sostenible. El consumo de energía no solo constituye una parte importante del coste de producción total, sino que también da lugar a efectos ambientales. El sector industrial es uno de los que más energía utiliza, consumiendo en algunos países cerca de la mitad de energía total. Este sector incluye a diversos tipos de industrias, como pueden ser la industria encargada del papel, los alimentos, los productos químicos, del hierro y acero, la minería, la agricultura, la construcción, etc. La energía se consume en el sector industrial para una amplia gama de propósitos, tales como procesamiento, montaje, producción de vapor, calefacción, aire acondicionado, iluminación, etc.

El consumo de energía del sector industrial también incluye productos derivados del petróleo y del gas natural, que son utilizados como materias primas para producir productos no energéticos, como fertilizantes para la agricultura y los productos petroquímicos para la producción de plásticos (EIA 2013). Datos estadísticos recogidos en 2009 por el gobierno alemán (BMWi 2009) muestran que un 47 % del consumo de energía del país fue consumido por el sector industrial, y el uso de esta electricidad produjo entre el 18 % y 20 % de las emisiones de CO_2 estatales. En la figura 1.1 se puede observar que a nivel mundial el porcentaje de consumo por el sector industrial es también muy elevado (Planeación Energética y Desarrollo Tecnológico 2009).

Los procesos industriales también son conocidos por ser las principales fuentes de gases de efecto invernadero (GHG). Estadísticas han demostrado que el GHG emitido por el uso de fuentes de energía, como la electricidad, el carbón, el petróleo y el gas, durante la fabricación de procesos industriales, es de entre el 37 % y el 50 % del GHG emitido mundialmente (Newman y col. 2012). Además, hay que tener en cuenta que las reservas de los recursos no renovables son finitas. La contabilidad de las reservas produce muchas disputas, con las estimaciones más optimistas por parte de las empresas y las más pesimistas por parte de los grupos ecologistas y los científicos académicos. Los primeros tienden a presentar como reservas todos los yacimientos conocidos más los que prevén encontrar. Los segundos ponen el acento en el coste monetario creciente de la exploración y de la extracción. Ni las mejores predicciones dan más de 100 años a recursos como el gas y

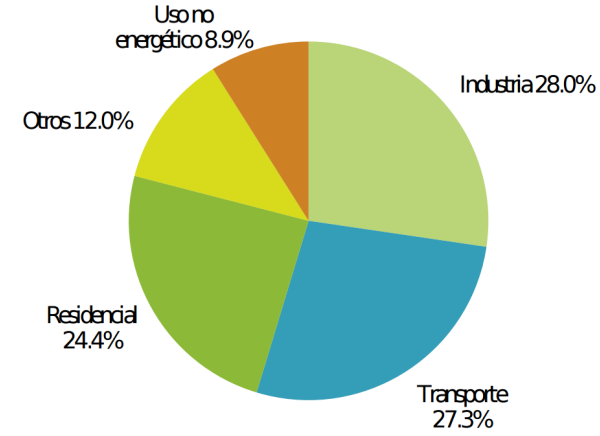


Figura 1.1: Consumo total mundial de energía por sector (2009)

el petróleo, si el consumo continúa en los niveles actuales. La compañía británica British Petroleum (BP) afirmó en un estudio reciente que las reservas de petróleo del planeta suman 1,68 billones de barriles y que con el ritmo de producción actual durarán hasta 2067 (Petroleum 2014).

Es por ello que, bajo la creciente presión derivada de la aplicación del protocolo de Kyoto y el protocolo de Copenhague, las empresas han comenzado a tomar medidas para reducir la emisión de gases de efecto invernadero en sus productos y servicios. El sector industrial tiene una influencia mucho mayor en el impacto ambiental, debido a que, como se ha indicado anteriormente, es el sector en el que se utiliza más energía. La energía que se consumió en el mundo en 2011 fue de 524 trillones de BTU[1] y la energía utilizada por las industrias fue del 51 % (Administration 2013). Así, la reducción del consumo en este sector afecta directamente al consumo total. Sin embargo, resulta difícil para las empresas tener en cuenta la aplicación de energías renovables y la reducción de emisiones cuando se toman decisiones de fabricación y funcionamiento, especialmente en los problemas de planificación y programación de la producción (Wang y col. 2011). El gran reto que se presenta es tratar de reducir energía pero sin una gran pérdida de optimalidad ya que, como se puede ver en la figura 1.2, la tendencia indica que el consumo de energía va a seguir en aumento.

[1] 1BTU = 1055,056 Julios

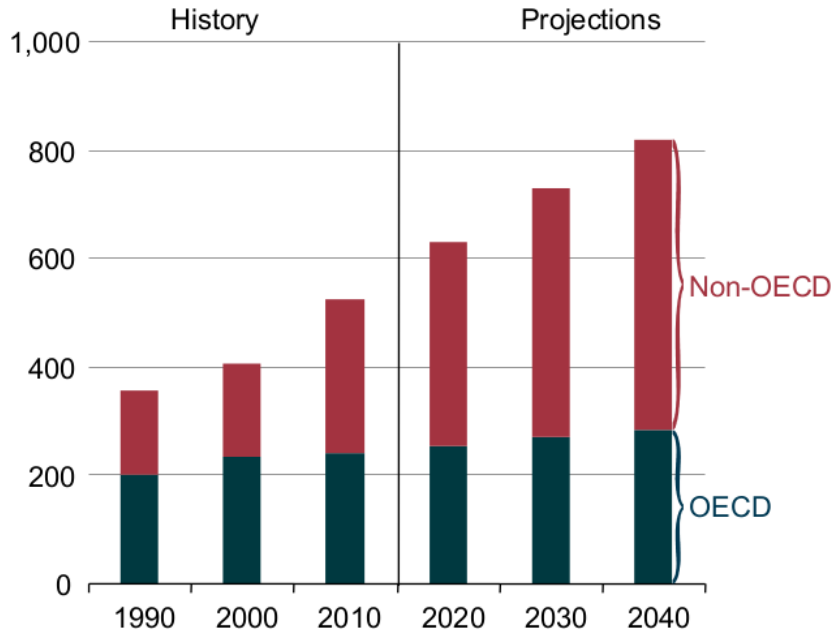


Figura 1.2: Consumo de energía mundial 1990-2040 (trillones de BTU) (Administration 2013)

En muchas ocasiones, cuando se está realizando un plan, tener que interrumpir el mismo por una razón inesperada puede ser muy crítico. Puesto que puede suponer la pérdida de mucho tiempo o la pérdida de recursos, ya que la detención de un plan puede suponer que ciertos recursos puedan no ser utilizados en el futuro. En casos extremos, una incidencia puede suponer un accidente y por tanto acarrear consecuencias como la destrucción de herramientas y recursos, la pérdida de vidas, catástrofes medioambientales, etc. Es por ello, que en ciertas ocasiones cuando se realiza un plan se debe de tener en cuenta la robustez, es decir, la capacidad de la solución para poder absorber supuestas interrupciones que puedan ocurrir. El grado de robustez no debe de ser siempre el mismo, dado que dependerá del problema y de las necesidades en cada momento.

Si se tiene en cuenta la robustez a la hora de realizar un plan normalmente conllevara la pérdida de optimalidad. En consecuencia puede suponer que se tarde más en realizar ciertas tareas por dejar espacios de seguridad o que el plan resultante sea más relajado. Aunque si se mira a largo plazo o teniendo en cuenta que pueden haber interrupciones puede resultar más rentable perder algo de optimalidad para ganar robustez, ya que ante una incidencia, los daños y el número de recursos para restablecer la solución serán menores.

En este trabajo se analiza la relación entre algunos parámetros importantes con el fin de obtener una solución multi-objetivo. De modo que se consiga un equilibrio claro entre el makespan y el consumo de energía; y entre el makespan y la robustez. Y como se puede observar, existe una estrecha relación entre el consumo de energía y la robustez. Sin embargo, esta estrecha relación no ha sido analizada en la literatura y nuevas técnicas pueden ser desarrolladas para lograr estos objetivos de forma conjunta.

En muchos problemas reales de planificación la robustez y la eficiencia energética pueden jugar un papel muy importante, y muy especialmente cuando se tienen en cuenta conjuntamente. Cuando tratamos con problemas de scheduling reales o problemas que simulan la realidad, la complejidad para obtener soluciones optimizadas de estos problemas es muy elevada y más aún cuando tenemos varios objetivos a optimizar. En estos casos, es necesario el desarrollo de técnicas de búsqueda heurísticas o metaheurísticas ya que los métodos de búsqueda exactos no son capaces de encontrar una solución optimizada a problemas complejos en un tiempo limitado. Para desarrollar técnicas heurísticas o metaheurísticas adaptadas a cada problema es necesario previamente realizar un estudio de los parámetros influyentes en la optmización de las soluciones. Es por ello que, surge la motivación de realizar un estudio entre dichos objetivos, ya que consideramos que tanto la eficiencia energética como la robustez son dos parámetros que se deben tener muy en cuenta en los problemas de scheduling. Mediante la optimización de la eficiencia energética se reduce el consumo de energía, y por lo tanto, se reduce el coste y la contaminación. Si se optimiza la robustez, la solución tendrá mejor capacidad para adaptarse a pequeños cambios que aparezcan. Cuando una solución no es capaz de adaptarse a dichos cambios, la solución no es válida, y por lo tanto se deben aplicar técnicas de recuperabilidad. Por lo que en dicho trabajo hemos desarrollado técnicas para replanificar una solución, que no puede adaptarse a cierta incidencia, minimizando los cambios y tratando de restablecer lo antes posible el plan.

1.4 Estructura del Trabajo

En esta tesis se plasma el conjunto de trabajos realizados a lo largo del doctorado. El trabajo de investigación se puede dividir en varias fases:

- Análisis del problema job-shop scheduling (JSP) y métodos para resolverlo: Se analizan las distintas técnicas utilizadas en la literatura y se desarrollan y analizan varias técnicas para obtener resultados al problema JSP y derivados.
- Extensión de benchmarks del clásico problema JSP a un problema Job-Shop Scheduling con Máquinas a distintas Velocidades (JSMS): Creación de instancias JSMS a partir de instancias JSP, por medio de la incorporación de distintas velocidades a las máquinas.

- Análisis de la eficiencia energética en problemas de scheduling: Se analiza cómo ha sido estudiada la eficiencia energética en los problemas de scheduling y se aplica al problema JSMS.
- Análisis de la robustez en problemas de scheduling: Se realiza un análisis para medir la robustez en los problemas de scheduling y se aplica al problema JSMS.
- Análisis de la Robustez, el Consumo Energético y el Makespan en el problema JSMS: Se analiza la relación entre los distintos objetivos.
- Rescheduling en JSMS: Cuando no es posible mantener la robustez en un problema se debe recurrir a replanificar la solución, por lo que en esta fase se analiza de qué forma se puede replanificar una solución minimizando los cambios y relacionando la replanificación con las características de nuestro problema JSMS y la eficiencia energética.

En el capítulo 2 se presenta el problema JSP y se explican distintos métodos que se pueden utilizar para resolverlo. Primeramente, se comentan diversos métodos exactos para resolver el problema, donde su objetivo es hallar la solución óptima al problema. Después, se presentan diversos métodos heurísticos y metaheurísticos los cuales tratan de encontrar una buena solución al problema en un tiempo limitado. Seguidamente, se presentan las características principales de un problema de Satisfacción y Optimización de Restricciones (CSOP) y se describen las técnicas y métodos para resolver estos problemas.

En el capítulo 3 presentamos una variante del problema JSP, el cual tiene la peculiaridad de que cada tarea debe ser ejecutada por un operario (JSO). Este problema es modelado y resuelto mediante un CSOP. Posteriormente, se aborda el problema añadiendo como objetivo la robustez. Para ello se desarrolla una técnica que aborda el problema en 3 pasos. En el primer paso se resuelve el clásico problema JSP como un CSOP. Posteriormente la solución obtenida se modifica para que tenga en cuenta los operadores y además maximice el número de buffers, para que la solución sea capaz de absorber pequeñas incidencias que puedan aparecer. Por último, se modifica esta solución para distribuir los buffers, y por lo tanto, maximizar el número de tareas que dispongan de un buffer.

En el capítulo 4 se extiende el problema JSP para que las máquinas puedan trabajar a distintas velocidades (JSMS) y por lo tanto tener distintos consumos de energía. Mediante la resolución de este problema se obtienen soluciones donde se tiene en cuenta el makespan y el consumo de energía. Para ello se ha diseñado un algoritmo genético y un algoritmo memético, el cual resulta de la hibridación de un algoritmo genético más una búsqueda local. Los resultados de ambos algoritmos son comparados con los obtenidos por la herramienta IBM ILOG CPLEX CP Optimizer.

En el capítulo 5 se tiene en cuenta la robustez y la estabilidad en el problema JSMS. Por ese motivo hemos diseñado un modelo con el cual se obtienen soluciones más robustas

que con el modelo habitual. Para desarrollar dicho modelo se analizan las incidencias que afectan a las soluciones del problema JSMS. Posteriormente, las tareas más propensas a sufrir una incidencia son protegidas haciendo uso del análisis previo de las incidencias. En las soluciones finales obtenidas, dichas tareas tendrán la habilidad de poder absorber gran parte de las incidencias que puedan ocurrir.

En el capítulo 6 se realiza un estudio sobre la relación entre la optimalidad, la eficiencia energética y la robustez. De manera que sabiendo la relación que existe entre ellas, mediante fórmulas matemáticas, se puedan aproximar las soluciones.

Cuando una solución no es capaz de recuperarse tras una incidencia, es necesario el rescheduling y esto es tratado en el capítulo 7. En dicho capítulo se desarrolla una técnica para replanificar el schedule cuando sea necesario, con la intención de restablecer el plan anterior lo antes posible y minimizando los cambios. Para las soluciones que no son válidas tras una incidencia se aplica una técnica llamada Match-up, mediante la aceleración de las máquinas de las tareas posteriores a la incidencia se reduce el tiempo de procesamiento de estas. De tal forma que, se pueda absorber la incidencia y determinar un punto de tiempo (punto Match-up) a partir del cual las tareas pueden ser ejecutadas como estaba establecido antes de que ocurriera la incidencia. Además, al determinar el punto Match-up, acelerando todas las tareas afectadas, se obtiene una solución válida. Esta solución puede ser mejorada posteriormente a nivel de eficiencia energética, reduciendo la velocidad de las máquinas (cuando sea posible) entre el instante de la incidencia y el punto Match-up.

Por último, en el capítulo 8 se concluyen las ideas fundamentales tratadas durante todo el trabajo de la tesis doctoral y se apuntan futuras líneas de trabajo.

Capítulo 2

El Problema de Job Shop Scheduling

2.1 Introducción

En la sección 1.2 se han presentado las características de los problemas de scheduling. Uno de los problemas más comunes de scheduling es el Job Shop Scheduling (JSP). El problema JSP está descrito formalmente en numerosos trabajos, utilizando generalmente, modelos matemáticos o redes de restricciones (Balas 1967, Dorndorf, Pesch y Phan-Huy 2000). En este capítulo se presentan las dos aproximaciones más comunes para modelar este tipo de problema, así como algunos de los métodos más usados para resolverlo.

A partir del estudio que hemos realizado, hemos clasificado los métodos para resolver el problema JSP. En primer lugar, se realiza una clasificación de los métodos de resolución, en la cual se diferencia entre métodos de optimización exacta, métodos heurísticos y métodos metaheurísticos. Los métodos de optimización exacta son empleados generalmente para resolver problemas que han sido modelados mediante programación matemática. Entre los métodos heurísticos y metaheurísticos se puede diferenciar entre aquellos cuya heurística está basada en las propiedades de un Job Shop (Desplazar Cuello de Botella y Reglas de Procesamiento), y entre aquellos que son comunes a la resolución de cualquier problema de optimización combinatoria.

Este capítulo está organizado de la siguiente forma: En la sección 2.2 se indican las características del problema Job Shop Scheduling (JSP), se proporcionan los elementos que lo definen, así como, el significado dado en la literatura especializada. Estos elementos y las restricciones que deben ser satisfechas por el mismo son modelados mediante un grafo

disyuntivo. En la Sección 2.3 se describen métodos de optimización exacta. El objetivo de los mismos consiste en hallar una única solución a un problema, que de existir, deberá ser la óptima. Por otro lado, en la sección 2.4 describimos métodos heurísticos, para los cuales el principal objetivo es hallar una solución optimizada en un periodo de tiempo dado. Para ello se utilizan heurísticas basadas en las propiedades de un Job Shop. A continuación, en la sección 2.5 se presentan algunas de las técnicas metaheurísticas típicamente usadas en el problema Job Shop scheduling. En la Sección 2.6 se describen las técnicas y métodos empleados para resolver problemas CSP, basados en las propiedades del mismo. Finalmente, presentamos las conclusiones de este capítulo.

2.2 Definiciones y Conceptos Básicos

En esta sección exponemos y definimos los conceptos básicos del problema Job Shop Scheduling, el cual es uno de los problemas típicos de scheduling, tal y como se ha presentado en la sección 1.2.

En un problema de Job Shop Scheduling existen n trabajos, cada uno formado por una secuencia ordenada de operaciones. Cada operación consiste en realizar una determinada tarea utilizando una de las M máquinas existentes en el problema. Se utiliza el concepto de máquina para representar un recurso en general. El procesamiento de la tarea j en la máquina i es referido como una operación (i, j) y su duración es p_{ij}. Para obtener un plan factible se debe de asignar un tiempo de comienzo a todas las tareas, satisfaciendo las siguientes restricciones:

- Las tareas de cada trabajo deben estar secuencialmente ejecutadas.
- Cada máquina puede ejecutar una sola tarea al mismo tiempo.
- No se permiten preferencias entre las tareas.

Al igual que en la mayoría de los problemas de scheduling (sección 1.2), el objetivo suele estar relacionado con el tiempo de procesamiento de los trabajos. Uno de los objetivos más utilizados es minimizar el tiempo en completar todas las tareas (makespan $C_{\max}$).

El grafo disyuntivo, introducido por Roy y Sussman 1964, se utiliza para representar un problema de Job Shop scheduling. Se trata de un grafo dirigido G, con un conjunto de nodos V y dos conjuntos de arcos A y E. En la figura 2.1 se muestra un ejemplo de este tipo de grafo.

Los nodos V corresponden a todas las operaciones (i, j) que deben ser llevadas a cabo en los n trabajos. Los arcos del conjunto A, llamados arcos conjuntivos, representan las rutas de los trabajos. Si el arco $(i, j) \rightarrow (h, j)$ es parte de A, entonces el trabajo j tiene que ser procesado en la máquina i antes de ser procesado en la máquina h. Esto significa

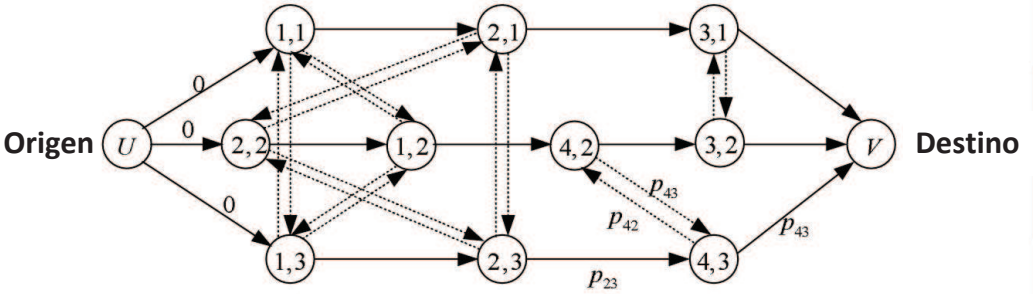


Figura 2.1: Grafo Dirigido para Job Shop con el makespan como el objetivo

que la operación (i, j) precede a la operación (h, j). Dos operaciones, (i, j), (i, k), que pertenecen a dos trabajos diferentes, j y k, y que deben ser procesadas por la misma máquina i, están conectadas una a otra por los llamados arcos disyuntivos E en direcciones opuestas.

El conjunto de arcos disyuntivos E forman m cliques de arcos dobles, un clique[1] para cada máquina. Todos los arcos, conjuntivos o disyuntivos, saliendo de un nodo tienen como etiqueta el tiempo de procesamiento de la operación representada por el nodo. Además, hay un nodo origen U y un nodo destino V, los cuales son nodos artificiales. El nodo origen U tiene n arcos conjuntivos que parten del mismo hacia las primeras operaciones de los n trabajos, y el nodo destino V tiene n arcos conjuntivos entrantes desde las últimas operaciones. Los arcos salientes del nodo U tienen etiqueta cero. En el grafo $G = (V, A, E)$ de la figura 2.1, se representa un JSP con cuatro máquinas y tres trabajos. En la figura 2.2 se indica la secuencia de operaciones que componen cada trabajo y la máquina requerida para cada una de estas operaciones.

Trabajo	Máquina			
1	1	2	3	
2	2	1	4	3
3	1	2	4	

Figura 2.2: Flujo de los Trabajos

Obtener una solución para un JSP corresponde a seleccionar uno de los arcos disyuntivos para cada par existente en el grafo, de tal forma que, el mismo resulte finalmente en un grafo dirigido acíclico. Esta selección determina la secuencia en la cual las operaciones van a ser llevadas a cabo sobre cada máquina.

[1] Un clique, en un grafo, es el menor conjunto de vértices, donde todos son adyacentes entre sí

Si D denota el subconjunto de los arcos disyuntivos seleccionados en la solución del problema y el grafo $G(D)$ es definido por el conjunto de arcos conjuntivos, entonces D corresponde a una solución si, y solo si, $G(D)$ no contiene ningún ciclo dirigido.

El makespan de una solución es determinado por el camino más largo en $G(D)$ desde el nodo origen U al nodo destino V. Este camino más largo está formado por un conjunto de operaciones, donde la primera de ellas empieza en el instante cero, y la última finaliza en el instante del makespan. Cada operación en este camino está inmediatamente seguida, ya sea por la siguiente operación sobre la misma máquina, o por la siguiente operación del mismo trabajo en otra máquina. El problema de minimizar el makespan se reduce a encontrar una selección de arcos disyuntivos que minimice la longitud del camino más largo (llamado camino crítico).

Hay varias formulaciones de programación matemática para Job Shop sin recirculación, incluyendo un número de formulaciones de programación entera. Sin embargo, la formulación usada más frecuentemente es la llamada formulación de programación disyuntiva. Esta formulación de programación disyuntiva está relacionada cercanamente con la representación del Job Shop por medio de grafos disyuntivos.

El hecho de que un problema de scheduling pueda ser formulado como un programa disyuntivo no implica que haya procedimientos de solución estándar disponibles que siempre trabajen satisfactoriamente. Además, minimizar el makespan en un Job Shop es un problema muy difícil y los procedimientos de solución están basados en procedimientos enumerativos o en procedimientos heurísticos. La complejidad de un problema JSP es NP-hard (Garey, Johnson y Sethi 1976).

2.3 Métodos de resolución exactos

En esta sección describimos algunos de los métodos empleados para resolver problemas de optimización combinatoria modelados mediante la programación matemática, obteniendo, si este tiene solución, la más óptima (Srinivasan 1971, Davis y Jones 1988).

2.3.1 Programación Dinámica

El término Programación Dinámica fue acuñado por Bellman 1957. Anteriormente el término programación no se refería a la programación en ordenadores, era utilizado para denotar planificación (tienen una etimología similar a la programación lineal, la programación mixta, etc.). La programación dinámica fue creada con el objetivo de planificar de forma óptima procesos multi-etapas. En el trabajo realizado por Dasgupta, Papadimitriou y Vazirani 2006 se explican varios ejemplos de este tipo de procesos resueltos utilizando Programación Dinámica.

Este método es un paradigma algorítmico basado en el principio de optimalidad de Bellman "*Cualquier subsecuencia de decisiones de una secuencia óptima de decisiones que resuelve un problema, también debe ser óptima respecto al subproblema que resuelve*". Por lo tanto, para resolver un problema utilizando Programación Dinámica, es necesario identificar tanto los subproblemas que lo forman, como la relación que existe entre ellos. La relación establece un orden para resolver los subproblemas, ya que la solución de uno es necesaria para resolver otro. Así, el subproblema más pequeño es aquel que no requiere de la solución de ningún otro, sin embargo, su solución sí es necesaria para el siguiente subproblema en el orden. El subproblema más grande, es aquel cuya solución requiere la solución del resto de subproblemas identificados. Así, resolviendo el último subproblema, el problema original queda resuelto.

El método de Programación Dinámica está caracterizado por tres tipos de ecuaciones, particularmente: condiciones iniciales, relación recursiva y una función del valor óptimo.

En scheduling se puede seleccionar entre programación dinámica hacia adelante y programación dinámica hacia atrás. En el trabajo realizado por Webster y Azizoglu 2001 se presenta un algoritmo de programación dinámica hacia adelante y otro hacia atrás, con el objetivo de minimizar el total weighted flowtime en el problema jobs scheduling con tiempos de preparación en máquinas paralelas idénticas. De este estudio se extrae que el algoritmo hacia atrás es mejor cuando la suma del tiempo de procesado y los tiempos de configuración son mayores que la suma de los pesos, de lo contrario la recursión hacia adelante se ve favorecida.

Formulación de Programación Dinámica hacia Adelante

Cuando la ecuación que establece la relación entre los subproblemas j y $j+1$ es definida de tal forma que, la solución óptima de j depende de la solución óptima de $j+1$, se dice que el método de Programación Dinámica se aplica hacia adelante.

Formulación de Programación Dinámica hacia Atrás

Cuando se aplica Programación Dinámica hacia atrás, el subproblema inicial representa el estado final del problema original. Por lo tanto, la ecuación que establece la relación entre los subproblemas es tal que, el subproblema j requiere de la solución hallada al subproblema $j-1$.

2.3.2 Branch and Bound

El método Branch and Bound (B&B) fue introducido por Land y Doig 1960, en un trabajo en el cual describían al nuevo método como una aproximación para hallar soluciones óptimas a problemas de optimización discretos y combinatorios.

Los problemas de optimización discretos son aquellos en los que las variables de decisión asumen valores discretos desde un conjunto específico; cuando este conjunto es el conjunto de los números enteros, se tiene un problema de programación entera. Los problemas de optimización combinatorios, por otro lado, son problemas en los cuales se debe elegir la mejor combinación factible de valores para las variables del problema, de tal forma que, la función objetivo sea evaluada al valor óptimo.

Un procedimiento B&B básico se inicia resolviendo el modelo matemático original sin las restricciones que obligan a instanciar determinadas variables únicamente con valores enteros (restricciones de integridad). El modelo sin las restricciones de integridad se denomina relajación de un problema de programación lineal LP (Linear Program) de un problema de programación entera IP (Integer Program). Denominamos $\overline{X}^0 = (< x_1, a_1 > , ..., < x_n, a_n >)$ a la asignación de las n variables del problema, mediante la cual se obtiene el valor óptimo de la función objetivo. Si la solución de la relajación LP es una solución entera, es decir, los valores asignados a las variables enteras son valores pertenecientes al conjunto de números enteros, entonces esta solución es también óptima para el problema original (problema en el cual se incluyen las restricciones de integridad). Si $\overline{X}^0$ no es entera, entonces el valor de la solución óptima de la relajación LP, $\overline{CX}^0$, sirve como un límite inferior (si el objetivo es minimizar), o un límite superior (si el objetivo es maximizar), para el valor de la solución óptima al problema de programación entero original.

Si una de las variables en $\overline{X}^0$ no es entera, por ejemplo $x_j = r$, entonces el procedimiento Branch and Bound procede como sigue. El problema de programación entera es dividido en dos subproblemas SP(1), SP(2), agregando dos restricciones mutuamente exclusivas y mutuamente exhaustivas. En el subproblema SP(1), el problema de programación entero original es modificado por agregar la restricción adicional

$$x_j \leq \lfloor r \rfloor,$$

donde $\lfloor r \rfloor$ denota el entero más grande menor que r, mientras que en el subproblema SP(2), el problema de programación entero original es modificado por agregar la restricción adicional

$$x_j \geq \lceil r \rceil,$$

donde $\lceil r \rceil$ denota el entero más pequeño mayor que r.

El procedimiento Branch and Bound ahora considera la relajación LP de uno de los subproblemas, por ejemplo SP(1), y lo resuelve. Si la solución es entera, entonces esta rama del árbol ya no tiene que ser explorada posteriormente, ya que esta solución es la óptima

para la versión de programación entera de SP(1). Si por el contrario, la solución no es entera, SP(1) tiene que ser dividido en dos subproblemas, SP(1,1) y SP(1,2), a través de la adición de restricciones mutuamente exclusivas y exhaustivas. Cada nodo que corresponde a una solución no entera se ramifica en otros dos nodos y así sucesivamente. Si una solución en un nodo es no entera, entonces este valor proporciona un límite inferior (o superior según se minimice o maximice) para todas las soluciones de sus descendientes.

Si asumimos que se está minimizando, el procedimiento Branch and Bound finaliza cuando todos los nodos del árbol, o bien tienen una solución entera, o bien tienen una solución no entera que es mayor a las soluciones enteras del resto de nodos. El nodo con la mejor solución entera provee una solución óptima al problema de programación entero original.

Con el método Branch and Bound se intenta eliminar un nodo con el objetivo de determinar un límite inferior para el valor de la función objetivo de todas las soluciones correspondientes a la descendencia de dicho nodo. Si el límite inferior obtenido es mayor al valor de la función objetivo de una solución ya obtenida previamente, entonces ese nodo y toda su descendencia es descartada de la búsqueda, por lo que el árbol es podado. Sin embargo, cuando es posible podar un árbol de búsqueda utilizando el método Branch and Bound, el número de nodos a evaluar en un problema real puede ser muy elevado. La ventaja de este método es que todos los nodos resultantes han sido evaluados, garantizando que la solución obtenida es la óptima.

El algoritmo Branch and Bound ha sido utilizado para resolver el problema JSP en el trabajo de Brucker, Jurisch y Sievers 1994, en el cual se desarrolla un algoritmo B&B rápido para resolver un problema con 10 trabajos y 10 máquinas en menos de 19 minutos. En el trabajo de Brah y Hunsucker 1991 también se aplica el algoritmo Branch and Bound para resolver el problema flow shop con múltiples procesadores. En el trabajo realizado por Brucker, Hurink y col. 1997 se han resuelto las instancias del problema open-shop del benchmark propuesto por Taillard de una forma eficiente mediante el uso de un algoritmo B&B. Dichas soluciones han sido comparadas a las soluciones obtenidas por un algoritmo heurístico de búsqueda Tabú y se han obtenido mejores soluciones con menos esfuerzo computacional. Al probar con instancias más complejas creadas por ellos mismos, la diferencia entre las dos técnicas es más grande.

Relajación Lagraniana

La Relajación Lagrangiana (LR), también conocida como Descomposición Lagrangiana, fue introducida al inicio de la década de 1970 por medio del trabajo pionero de Held y Karp 1970, Held y Karp 1971. Utilizando el problema del viajante de comercio como marco de trabajo.

LR es una técnica de descomposición dirigida por la función de optimización. Magnanti y Wong han denominado a esta técnica *Directiva del coste* (Magnanti y R.T. 1990). En

primera instancia, esta técnica reduce y simplifica el problema relajando determinadas restricciones. Muchos problemas de optimización combinatoria se consideran complicados por determinadas restricciones. Aplicar Relajación Lagrangiana a estos problemas supone identificar las restricciones que aumentan la complejidad del problema, eliminarlas como restricción del problema, y absorberlas en la función objetivo, penalizándolas con un factor denominado multiplicador de Lagrange. El siguiente objetivo consiste en hallar un límite inferior o superior, según la función objetivo, lo más cercano al valor óptimo. Para alcanzar este objetivo se procede a resolver, iterativamente, una secuencia de subproblemas modificados.

Considerando el proceso que define la Relajación Lagrangiana, Beasley 1993 indicó que la Relajación Lagrangiana requiere dirigir dos aspectos importantes; uno es un aspecto estratégico, y el otro es un aspecto táctico. En el primero incluyó la clasificación y relajación de las restricciones. En el segundo, la selección de una buena técnica para actualizar los multiplicadores de Lagrange.

Principalmente en una amplia variedad de problemas, ha habido dos técnicas que han sido aplicadas para hallar valores a los multiplicadores de Lagrange. Estas son *Optimización del Subgradiente* y *Ajuste del Multiplicador*.

En el trabajo realizado por Poojari, Mitra y Siamitros 2005 se realiza una descripción detallada de cómo se lleva a cabo la selección de restricciones que se deberán relajar, así como el método de optimización del subgradiente para hallar los valores más adecuados de los multiplicadores de Lagrange. La Relajación Lagrangiana también ha sido empleada para resolver problemas de scheduling, como en el trabajo de Irohara 2010, donde se proponen tres algoritmos de LR para resolver el problema hybrid flow-shop scheduling con buffers limitados. Para ello, los autores relajando las restricciones de capacidad de las máquinas y las restricciones de precedencia, obtienen buenos resultados, especialmente para los problemas más grandes. En el trabajo realizado por Kaskavelis y Caramanis 1998 se aplica la Relajación Lagrangiana a problemas de Job Shop Scheduling de tamaño industrial, con más de 10000 restricciones de recursos. En este trabajo se añaden dos características nuevas para mejorar el algoritmo LR, se actualiza más frecuentemente el multiplicador y se ajusta el tamaño del subgradiente basándose en el valor del multiplicador.

2.3.3 Branch and Cut

Los métodos Branch and Cut son una combinación de Branch and Bound y Planos de Corte. Son utilizados para resolver problemas de programación entera mediante la relajación de las restricciones de integridad (Branch and Bound) y mediante la acotación del espacio de búsqueda para impedir soluciones no enteras (planos de corte).

Los planos de corte son desigualdades válidas, esto es, cualquier desigualdad $\pi^T x \leq \pi_0$ que satisfaga los puntos factibles del problema de programación entero original. Al espa-

cio convexo de todas las soluciones del problema de programación entera se le denomina hiperpoliedro. Los algoritmos de Planos de Corte fueron propuestos inicialmente por Gomory 1963, pero en esa primera aproximación no eran muy eficientes porque no acotaban suficientemente el espacio de búsqueda. Por ello la convergencia a soluciones enteras era muy lenta. En la década de 1980, se produjo el desarrollo de la teoría de poliedros y la especificación de planos de corte específicos para determinados tipos de problemas, mediante los cuales los métodos de planos de corte resurgieron como técnicas para acotar el espacio de búsqueda de un problema relajado.

Los planos de corte obtenidos según la descripción realizada por Gomory 1963 y Chvátal 1973 fue denominada *Planos de Corte Gomory-Chvátal*. Estos planos de corte eran obtenidos al combinar desigualdades del problema original y explotar el requerimiento de integridad de determinadas variables. Se puede ver el desarrollo de un ejemplo en el trabajo de Mitchell 2002.

Por otro lado, existen varios trabajos, Caprara y Fischetti 1997, Jünger, Reinelt y Thienel 1995, o Applegate y col. 1994 , en los cuales se han desarrollado desigualdades válidas para determinados tipos de problemas, como por ejemplo, el problema de la mochila o el problema del viajante de comercio. En estos trabajos se ha combinado la técnica para generar planos de corte Gomory-Chvátal con la teoría de poliedros que proporciona un método para obtener desigualdades que definan una de las caras del poliedro que representa el conjunto de soluciones del problema relajado.

El algoritmo Branch and Cut también ha sido aplicado a problemas de scheduling, como en el trabajo realizado por Hoffman y Padberg 1993, donde se resuelven problemas de scheduling relacionados con la tripulación de aerolíneas mediante el uso de dicho algoritmo. El problema de scheduling de las grúas del muelle en una terminal de contenedores también han sido tratado con el algoritmo Branch and Cut (Moccia y col. 2006).

2.3.4 Programación con restricciones

La programación con restricciones es un método basado en la especificación de un conjunto de restricciones, las cuales deben ser satisfechas por cualquier solución del problema planteado. La programación con restricciones difiere de los métodos comunes de programación, ya que, no se especifican explícitamente los pasos o secuencias que se deben seguir para obtener una solución. La programación con restricciones es utilizada para la descripción y resolución de problemas combinatorios particularmente difíciles, por ejemplo, en las áreas de planificación y scheduling. Estos métodos combinan búsqueda con mecanismos de propagación de restricciones, los cuales son capaces de reducir el espacio de búsqueda. Un ejemplo de método de programación de restricciones para problemas de scheduling es el propuesto por Grimes y Hebrard 2015 donde resuelven problemas de scheduling combinando inferencia relativamente simple, con una serie de técnicas de programación con restricciones genéricas. Otro trabajo a destacar es el realizado por La-

borie 2014 en el cual presentan 2 algoritmos para propagar las restricciones de recursos en recursos discretos. En el trabajo realizado por Monette, Deville, Van Hentenryck y col. 2009 se propone un enfoque de programación con restricciones para resolver un problema Job Shop scheduling, mediante un algoritmo de filtrado y heurísticas dedicadas que reducen los límites de las variables de decisión y se mejoran las cotas inferiores del árbol de búsqueda, posteriormente con un algoritmo de programación con restricciones realizan la búsqueda local en cada solución consiguiendo buenas cotas superiores con poco tiempo.

La programación con restricciones consiste en la incrustación de restricciones en un lenguaje principal. Los primeros lenguajes principales utilizados fueron los lenguajes de programación lógica, por lo que la programación con restricciones se llamó inicialmente programación lógica de restricción. Los dos paradigmas comparten muchas características importantes, como variables lógicas y backtracking. El backtracking ha sido muy utilizado en los problemas de scheduling (Sadeh, Sycara y Xiong 1995). En dicho trabajo se presentan distintas técnicas backtracking para resolver el problema JSP.

2.4 Métodos de resolución heurísticos

En esta sección se describen algunas técnicas de propósito general que han demostrado ser útiles en los sistemas de scheduling industriales. La mayoría de estos métodos no garantizan típicamente la solución óptima; su objetivo es encontrar una solución razonablemente buena en un periodo de tiempo relativamente corto. Aunque existen muchas técnicas, se resumen las más representativas.

2.4.1 Desplazar el Cuello de Botella

Adams, Balas y Zawack 1988 propusieron la heurística Desplazar el Cuello de Botella, donde hay un conjunto de M máquinas consideradas en el Job Shop. En la descripción de una iteración de la heurística se asume que en iteraciones previas ha sido fijado un orden de utilización sobre determinadas máquinas de M, las cuales están agrupadas en el subconjunto M_0.

Una iteración resulta de la inclusión de una máquina del conjunto $M \setminus M_0$ al conjunto M_0 y de la secuencia en que serán procesadas las operaciones que requieran de la misma. Para determinar que máquina de $M \setminus M_0$ debería ser incluida, se analiza cuál de ellas es la que causa en un sentido o en otro el trastorno más severo. Para determinar esto, el grafo dirigido original G es modificado borrando todos los arcos disyuntivos de las máquinas pertenecientes a $M \setminus M_0$ y manteniendo únicamente los arcos disyuntivos relevantes a las máquinas en M_0(uno de cada par). El grafo resultante es denominado G'. Borrar todos los arcos disyuntivos de una máquina específica implica que todas las operaciones en esta máquina, las cuales originalmente deben ser realizadas una tras otra, ahora pueden ser realizadas en paralelo (como si la máquina tuviera capacidad infinita, o lo que es igual,

que cada operación tuviera su propia máquina). El grafo G' tiene uno o más caminos críticos que determinan el correspondiente makespan, a lo que se llama $C_{\max}(M_0)$.

Se considera cada una de las máquinas en $M \setminus M_0$ como un problema de una sola máquina con instantes de inicio mínimo (release date) e instantes de finalización (due date) y con la demora máxima que debe ser minimizada. El mínimo valor de $L_{\max}$ en el problema de única máquina correspondiente a la máquina i es denotado por $L_{\max}(i)$ y es una medida de lo crítico que resulta el procesamiento en la máquina i.

Después de resolver todos estos problemas de única máquina, la máquina i con la mayor demora mínima $L_{\max}(i)$ es elegida. Esta máquina se etiqueta como h, a su máxima demora como $L_{\max}(h)$ y la secuencia de operaciones a ser procesadas es dada por la solución del problema de máquina única asociada, cuyo coste fue $L_{\max}(h)$. Si los arcos disyuntivos que especifican la secuencia de las operaciones en la máquina h son insertados en el grafo G, entonces el makespan de la solución parcial actual se incrementa en al menos $L_{\max}(h)$, esto es: $C_{\max}(M_0 \cup h) \geq C_{\max} + L_{\max}(h)$.

Antes de empezar la siguiente iteración y determinar la siguiente máquina a ser programada, un paso adicional debe ser realizado dentro de la iteración actual. En este paso adicional todas las máquinas en el conjunto original M_0 son resecuenciadas una a una para ver si el makespan puede ser reducido. Esto es, una máquina l es retirada del conjunto M_0 y un grafo G'' es construido por modificar el grafo G' a través de la inclusión de los arcos disyuntivos que especifican la secuencia de operaciones en la máquina h, excluyendo arcos asociados con la máquina l. La máquina l es resecuenciada para resolver el correspondiente problema de demora máxima para única máquina con el instante de inicio y de finalización determinados por los caminos críticos en el grafo G''. Resecuenciar cada una de las máquinas en el conjunto original completa la iteración. En la siguiente iteración el procedimiento completo es repetido y otra máquina es agregada al conjunto actual $M_0 \cup h$.

La estructura de la heurística *"Desplazar el Cuello de Botella"* muestra la relación entre el concepto de cuello de botella y los conceptos más combinatorios, tales como, el camino crítico (camino más largo) y la demora máxima. Un camino crítico indica la ubicación y el timing de un cuello de botella. La máxima demora da una indicación de la cantidad en la que es incrementado el makespan cuando una determinada máquina es añadida al conjunto de máquinas ya programadas. Como se ha indicado esta heurística no garantiza que la solución generada sea la solución óptima. Por esto Pinedo y Singer 1999 proponen un método utilizando esta heurística para minimizar el retraso total ponderado.

2.4.2 Reglas de Procesamiento Básico

Una regla de procesamiento es la que asigna prioridades a todos los trabajos que están esperando ser procesados por una máquina. El esquema de prioridades puede tener en cuenta los atributos del trabajo, los atributos de la máquina, así como el instante de tiempo actual. Cuando una máquina es liberada, una regla de procesamiento selecciona, entre los trabajos esperando por dicha máquina, aquel trabajo con la mayor prioridad. En los trabajos de Lawrence 1984 y de Holthaus y Rajendran 1997 se proporciona una descripción de varias reglas para asignar prioridades.

Las reglas de procesamiento pueden ser clasificadas de varias formas. Por ejemplo, una distinción puede ser realizada entre reglas de procesamiento estáticas y reglas de procesamiento dinámicas. Las reglas de procesamiento estáticas no son dependientes del tiempo, sino que dependen de los datos relacionados al trabajo y/o a la máquina. Un ejemplo de regla dinámica es la regla de la *Mínima Holgura (MS)*, la cual ordena trabajos según la holgura restante de los mismos. La holgura restante es definida como $max(d_j - p_j - t, 0)$ donde d_j es el instante de finalización del trabajo j, p_j su tiempo de procesamiento, y t el instante de tiempo actual. Esto implica que en algún instante de tiempo el trabajo j puede tener mayor prioridad que el trabajo k y en otro instante de tiempo puede darse lo contrario.

Una segunda forma de clasificar las reglas de procesamiento es según la información en la que se basan. Una regla local únicamente considera información perteneciente, ya sea a la cola donde el trabajo está esperando o a la máquina donde el trabajo está en cola. Las reglas globales usan información relacionadas con otras máquinas. A continuación se enumeran algunas reglas de procesamiento básico.

- Regla de *Instante de Inicio más Temprano (ERD)*: selecciona al trabajo que llega primero a la máquina.
- Regla de *Instante de Finalización más Temprano (EDD)*: selecciona el trabajo cuyo instante de finalización (due date) es más próximo.
- Regla de *Mínima Holgura (MS)*: selecciona el siguiente trabajo seleccionado para usar la máquina.
- Regla de *Tiempo de Procesamiento más Largo (LPT)*: selecciona primero aquel trabajo con tiempo de procesamiento mayor en la máquina considerada.
- Regla de *Tiempo de Procesamiento Ponderado más Corto (WSPT)*: los trabajos son seleccionados en orden decreciente según el valor de w_j/p_j, siendo w_j el peso asignado al trabajo j en la función objetivo y p_j su tiempo de procesamiento en la máquina considerada.

- Regla de *Tiempo de Setup más Corto (SST)*: selecciona el trabajo con el tiempo de configuración más corto.
- Regla de *Trabajo menos Flexible (LFJ)*: esta regla es utilizada en el caso en el que existan máquinas no idénticas en paralelo. Cada trabajo j puede ser procesado por un subconjunto M_j de máquinas en paralelo. Aquel trabajo que pueda ser procesado por el menor subconjunto M_j es el siguiente trabajo seleccionado.
- Regla de *Camino Crítico (CP)*: aquel trabajo con el mayor tiempo de procesamiento total, considerando únicamente las actividades sucesivas a la actual, es el siguiente trabajo a ser seleccionado.
- Regla de *Número Mayor de Sucesores (LNS)*: selecciona como siguiente trabajo aquel que tenga el mayor número de trabajos posteriores.
- Regla de *Cola más Corta en la Siguiente Operación (SQNO)*: se selecciona el trabajo con la cola de trabajos más corta en la siguiente máquina, la longitud de la cola puede ser medida como el número de trabajos esperando en cola o la cantidad total de trabajo esperando en cola.
- Regla de *Servicio en Orden Aleatorio (SIRO)*: el siguiente trabajo es seleccionado de forma aleatoria de entre los trabajos esperando para utilizar la máquina.

En la tabla 2.1 se presenta un resumen de las reglas de procesamiento básico enumeradas previamente, junto con los objetivos y atributos considerados para cada una.

2.4.3 Reglas de Procesamiento Compuesto

Las reglas de procesamiento básico son útiles cuando se intenta encontrar una solución razonablemente buena con respecto a un único objetivo tal como el makespan, la suma de los instantes en que los trabajos han sido completados o la máxima demora.

Sin embargo, en la práctica los objetivos son mucho más complejos. Un objetivo realista puede ser una combinación de varios objetivos y puede ser también en función del tiempo o en función del conjunto de trabajos esperando ser procesados. Según Dechter 2003, ordenar los trabajos en base a uno o dos parámetros puede conducir a soluciones no aceptables. Reglas de procesamiento más elaboradas, que consideran ciertos parámetros, pueden funcionar mejor cuando se utilizan funciones objetivo más complejas.

Una regla de procesamiento compuesto es una expresión en la cual se combinan ciertas reglas de procesamiento básico, cada una con un peso determinado. La medida en la que un determinado atributo afecta a la prioridad total de un trabajo es determinada por la regla de procesamiento básico que lo utiliza y un parámetro de escala. Cada regla básica, en la regla compuesta, tiene su propio parámetro de escala que es utilizado según la

	Regla	Objetivos
Dependiente del instante de inicio y del instante de finalización	**ERD**	Minimizar la variación en el tiempo de espera
	EDD	Minimizar el máximo retraso
	MS	Minimizar el retraso total
Dependiente del tiempo de procesamiento	**LPT**	Equilibrar la carga entre máquinas paralelas
	WSPT	Minimizar de forma ponderada la suma de instantes de finalización
	CP	Minimizar el makespan. Finalizar cuanto antes el trabajo que se prevé requiera mayor tiempo de procesamiento
	LNS	Minimizar el makespan, cuando se establece una determinada sucesión entre los trabajos
Otros	**SST**	Minimizar el makespan y tiempo de espera cuando se considera el tiempo de configuración
	LFJ	Minimizar el makespan cuando una operación puede ser procesada por más de una máquina diferente
	SQNO	Minimizar el makespan cuando existe diferencia en cuanto a a las máquinas utilizadas por los trabajos
	SIRO	Facilidad de Implementación

Tabla 2.1: Resumen de las reglas de procesamiento básico que han sido estudiadas

contribución que se desee de la regla básica a la expresión del ranking total. El parámetro de escala puede ser fijado por el diseñador de la regla o puede ser variable en función del conjunto de trabajos particular a ser programados. Un ejemplo de reglas de procesamiento compuesto son las denominadas *ATC* y *COVERT*.

La regla *Coste del Retraso Aparente (ATC)* es una regla para una máquina y n trabajos (todos disponibles en el instante cero). El objetivo consiste en minimizar la suma de los retrasos de cada trabajo $\sum w_j T_j$. En Pinedo 2005, se proporcionan más detalles acerca de esta regla de procesamiento.

Por otra parte la regla *Coste en el Tiempo (COst oVER Time - COVERT)*. Es una combinación de dos reglas de procesamiento básico: *Tiempo de procesamiento Ponderado más Corto* (WSPT) y *Mínima Holgura* (MS). Cuando se prevé que un trabajo se ha retrasado (por ejemplo, porque su holgura actual es cero), su índice de prioridad se reduce a w_j/p_j (WSPT). Por otro lado, si se espera que un trabajo finalice anticipadamente, donde la holgura excede a una estimación del coste de retraso, el índice de prioridad para el trabajo se incrementa linealmente con el decremento de la holgura. COVERT usa una estimación de retrasos para el peor caso como la suma de tiempos de procesamiento multiplicado por un parámetro de búsqueda hacia adelante k, donde $(X)^+ = max(0, X)$. Así, la prioridad de un trabajo se incrementa linealmente desde cero, cuando la holgura es muy alta, a WSPT, cuando el status del trabajo se vuelve tardío. El siguiente trabajo procesado en una máquina es aquel con el índice de prioridad más alto. En Vepsalanen y Morton 1987 realizaron un estudio y pruebas de varias reglas de procesamiento tales como EDD, WSPT, COVERT, y ATC.

2.5 Técnicas de resolución Metaheurísticas

En esta sección describimos algunas de las técnicas empleadas para resolver problemas de scheduling industriales. Al igual que las técnicas heurísticas, las metaheurísticas no garantizan la solución óptima. Sin embargo, cuando por la dimensiones del problema resulta muy difícil obtener el óptimo con los métodos tradicionales, las técnicas metaheurísticas permiten obtener soluciones cercanas a la óptima en un tiempo razonable. La figura 2.3 representa una clasificación de gran parte de estas técnicas. A continuación se resumen brevemente algunas de ellas.

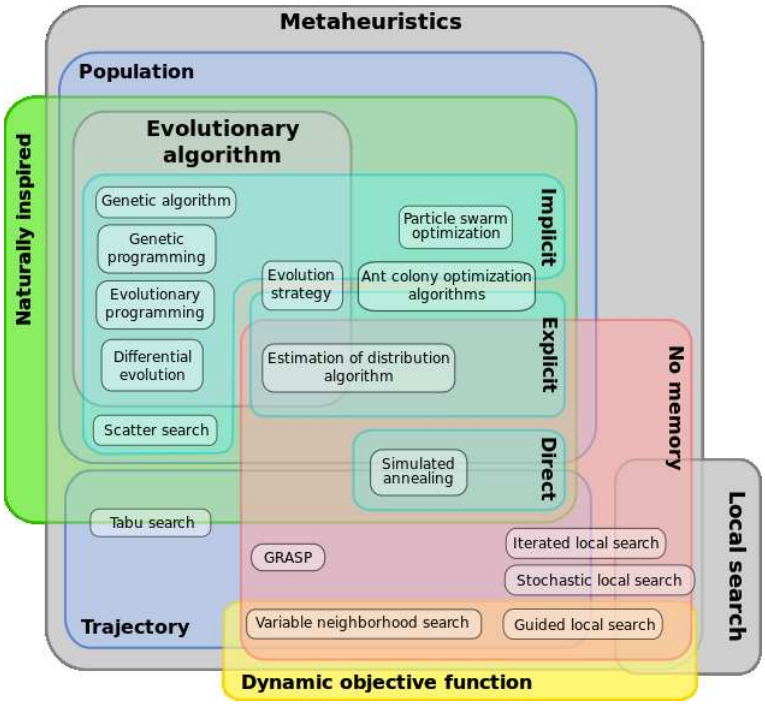


Figura 2.3: Clasificación de metaheurísticas (imagen extraída de http://en.wikipedia.org/wiki/Metaheuristic).

2.5.1 Enfriamiento Simulado (Simulated Annealing)

El enfriamiento simulado (SA) es un algoritmo de búsqueda metaheurística para problemas de optimización. El objetivo general de este algoritmo es encontrar una buena aproximación al valor óptimo de una función en un espacio de búsqueda grande. A este valor óptimo se lo denomina "óptimo global". Este método, descrito por Kirkpatrick, Gelatt y Vecchi 1983, es una adaptación del algoritmo Metropolis-Hastings, un método de Montecarlo utilizado para generar muestras de estados de un sistema termodinámico.

El nombre y el algoritmo están inspirados en el proceso de enfriamiento del acero y la cerámica, una técnica que consiste en calentar y luego enfriar lentamente el material para variar sus propiedades físicas. El calor causa que los átomos aumenten su energía y que puedan así desplazarse de sus posiciones iniciales (un mínimo local de energía). El enfriamiento lento les aporta mayores probabilidades de recristalizar en configuraciones con menor energía que la inicial (mínimo global).

La idea fundamental es admitir con una probabilidad especificada a algunos vecinos que empeoren la solución actual. De este modo, se trata de evitar que el algoritmo pueda caer en óptimos locales. El enfriamiento simulado selecciona una solución aleatoriamente entre los vecinos de la solución actual. Si la solución seleccionada es peor que la actual, la nueva solución es aceptada con una probabilidad que depende de dos parámetros: la temperatura y el incremento de energía. Pero si se elige una solución mejor que la actual, esta es aceptada siempre.

Al principio de la búsqueda, la temperatura debe tener un valor alto para que la probabilidad de que una solución pueda ser aceptada sea alta. Cuando el algoritmo avanza, el valor de la temperatura se debe de ir reduciendo. Por lo tanto, será menos probable aceptar una solución que empeore a la actual, hasta que, en la parte final del algoritmo, solo se aceptaran soluciones que mejoren a la actual. Para aplicar esta técnica a un problema de scheduling se deben definir reglas de vecindad, las más utilizadas son:

- El tiempo de procesamiento más Corto (SPT)
- El tiempo de Procesamiento más Largo (LPT)
- El instante de Finalización más Temprano (EDD)
- La regla de Mínima Holgura (MS)

Han sido muchos los trabajos donde se ha empleado esta técnica para el problema de Job Shop scheduling. Un ejemplo de esto es el artículo de Van Laarhoven, Aarts y Lenstra 1992 donde se aplica el algoritmo de enfriamiento simulado para encontrar el mínimo makespan al problema JSP. También en el trabajo realizado por Ponnambalam, Jawahar y Aravindan 1999, donde se comparan los resultados obtenidos con esta técnica y los obtenidos con algoritmos genéticos.

2.5.2 Búsqueda Tabú

La búsqueda tabú (TS) fue introducida por Glover y Laguna 1999 y está basada en un mecanismo de memorización, evitando la elección de ciertos vecinos basándose en la historia de búsqueda reciente, o en la frecuencia con la que se han realizado ciertas transformaciones para alcanzar el estado actual. El mecanismo de memorización tiene en cuenta las transformaciones realizadas para obtener las últimas soluciones, de forma que estas transformaciones se tienen en cuenta para la generación de los vecinos de la solución actual (se consideran tabú). La lista tabú debe tener un tamaño ajustado a las características del problema. En ocasiones puede ocurrir que un movimiento de la lista tabú produzca una mejora en la solución actual. Es por ello que se tienen en cuenta criterios de aspiración, que consisten en considerar excepciones a lo indicado por la lista tabú. Un criterio de

aspiración muy utilizado es admitir soluciones que son mejores que la mejor solución conocida hasta el momento.

Uno de los primeros trabajos en aplicar la búsqueda Tabú al problema de Job Shop fue el realizado por Taillard 1994. En dicho trabajo se compara su técnica sobre con resultados obtenidos por el proceso de desplazar el cuello de botella y el algoritmo de enfriamiento simulado. Los resultados obtenidos muestran la eficiencia del algoritmo.

2.5.3 Procedimiento de Búsqueda Voraz Adaptativo Probabilista (Greedy Randomized Adaptive Search Process)

Greedy Randomized Adaptive Search Process (GRASP) es un algoritmo metaheurístico aplicado a los problemas de optimización combinatoria (Feo y Resende 1995). GRASP es un procedimiento iterativo que está formado por una fase de construcción, donde se obtiene una solución, y una fase de mejoras iterativas de la misma a través de una búsqueda local. Las soluciones son generadas por la adición de elementos a una lista de elementos clasificados por una función voraz, en lugar de elegir siempre al mejor candidato se elige de entre una lista de los mejores, de donde se selecciona uno aleatoriamente. La lista puede tener un tamaño definido o, en ocasiones, el tamaño de esta lista viene dado por un porcentaje de tolerancia entre el valor del mejor elemento y el resto. Al final de cada iteración, todos los candidatos son revalorizados como consecuencia del último elemento insertado en la lista. Como criterio de parada del algoritmo se suele usar un número máximo de iteraciones sin actualizar la mejor solución.

Esta técnica ha sido empleada para resolver el problema Job Shop scheduling, como se puede observar en el trabajo de Binato y col. 2002. En dicho trabajo, incorporan al algoritmo GRASP dos conceptos nuevos: una estrategia de intensificación y una aproximación al principio de optimalidad en la fase de construcción. Mediante la aplicación de estas variantes se presenta una gran mejora respecto a resultados obtenidos por el algoritmo GRASP clásico.

2.5.4 Algoritmo Genético

Los algoritmos genéticos son métodos evolutivos adaptativos que pueden ser usados para resolver problemas de búsqueda y optimización (Goldberg 1985). Los algoritmos genéticos son muy adecuados para la resolución de problemas de scheduling porque, a diferencia de los métodos heurísticos, los algoritmos genéticos operan con una población de soluciones, en lugar de una única solución. Por ejemplo, una solución puede ser la optimización de un proceso de producción que se completará en una cantidad mínima de tiempo, mientras que en otra solución podemos estar optimizando una cantidad mínima de defectos.

A medida que aumentamos el número de objetivos que estamos tratando de lograr, también aumentamos el número de restricciones sobre el problema y, de manera similar, aumentamos la complejidad. Los algoritmos genéticos son adecuados para el tipo de problemas en el que el espacio de búsqueda es grande y el número de soluciones factibles es pequeño (Mangano 1995).

Para aplicar un algoritmo genético en un problema de scheduling, primero tenemos que representarlo como un cromosoma. Un cromosoma es una secuencia específica de tareas que representan un individuo en nuestra población, cada individuo representa una solución al problema. Para asegurarnos de que nuestro cromosoma es una solución viable, debemos asegurar que respeta las restricciones del problema. Para generar la población inicial se pueden generar combinaciones de tareas al azar, pero siempre cumpliendo las restricciones del problema. Una vez la población inicial ha sido creada, esta puede ser cruzada combinando cromosomas. Con la técnica de cruce conseguimos crear un individuo a partir de la unión de dos individuos de la población. Posteriormente, este nuevo individuo puede ser mutado, con la mutación dotaremos de aleatoriedad al algoritmo genético. La descendencia de esta combinación se selecciona basándose en una función de evaluación (fitness) que incluye uno o varios de nuestros objetivos, tales como la minimización del tiempo o la minimización de defectos. Este proceso continuará hasta que se cumpla un tiempo pre-asignado o hasta que encontremos una solución que se ajuste a ciertos criterios mínimos. En general, cada nueva generación tendrá una mejor función de evaluación media, es decir, las soluciones actuales tendrán mayor calidad que las generaciones anteriores. En problemas de scheduling hay que asegurarse de que la descendencia seleccionada es viable, ya que se podrían crear soluciones que no cumplan las restricciones del problema. Por ello, es muy importante la codificación y decodificación del cromosoma.

En la literatura se pueden encontrar numerosos trabajos donde se ha utilizado un algoritmo genético para resolver el problema de Job Shop scheduling. Un claro ejemplo es el trabajo realizado por Davis 1985. También, en el trabajo realizado por Della Croce, Tadei y Volta 1995 se presenta un algoritmo GA donde la codificación está basada en órdenes de preferencia y codificación, de esta forma se consiguen obtener siempre cromosomas factibles. En el trabajo realizado por Xu y col. 2014 se ha diseñado un algoritmo GA para la asignación de prioridades a tareas mientras se usa una heurística basada en el tiempo de fin más temprano.

2.6 Técnicas de Resolución de un CSP

Los Problemas de Satisfacción de Restricciones (CSP) son problemas matemáticos definidos como un conjunto de variables que deben satisfacer un número de restricciones. Un CSP puede ser resuelto asignando valores del dominio para cada variable del problema. El espacio de soluciones puede ser visto como un árbol de búsqueda, donde cada nivel representa una variable. Un CSP se representa por la tripleta (X, D, C) donde:

- $\mathcal{X} = x_1, ..., x_n$ es un conjunto de n variables.
- $\mathcal{D} = d_1, ..., d_n$ es un conjunto de dominios, de modo que cada variable $x_i \in X$ tiene un dominio finito de valores d_i.
- $\mathcal{C} = c_1, ..., c_m$ es un conjunto finito de restricciones que limitan los valores que las variables pueden tomar de forma simultánea.

La mayoría de los algoritmos para la resolución de CSP buscan la asignación sistemática de valores a las variables. Estos algoritmos garantizan o bien encontrar una solución o bien demostrar que el problema no tiene solución.

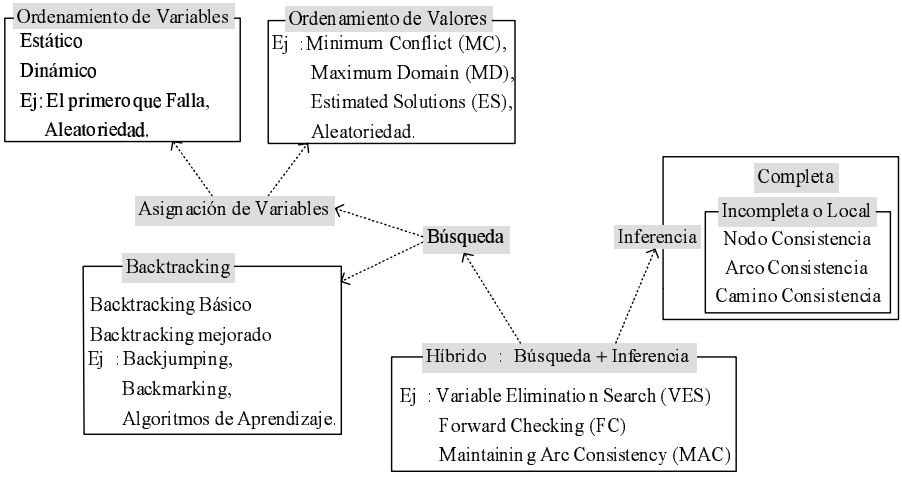


Figura 2.4: Esquema de Técnicas utilizadas en la resolución de CSPs

Los métodos para resolver CSPs se pueden clasificar en base a la relación que existe entre el proceso de búsqueda y las técnicas, que utilizando propiedades de un CSP, buscan disminuir el espacio de búsqueda (figura 2.4). La forma más común para reducir el espacio de búsqueda es la poda. La poda solo se utiliza si la rama a podar no contiene soluciones parciales (Mencía y col. 2012). La desventaja de estos algoritmos es que pueden tardar

mucho tiempo en encontrar una solución. Un CSP también puede ser resuelto por un procedimiento heurístico de búsqueda local, tales como los citados en la figura 2.4.

2.6.1 Técnicas de consistencia

Las técnicas de consistencia fueron introducidas para mejorar la eficiencia de las técnicas de búsqueda. Para obtener una solución para un CSP se debe asignar un valor del dominio a cada una de las variables, el número de posibles asignaciones de valores a las variables puede ser enorme y probablemente muy pocas serán consistentes y por lo tanto soluciones válidas. Con la eliminación de los valores redundantes según la definición del problema, el espacio de soluciones disminuye. La reducción del espacio de soluciones se puede realizar una sola vez como un pre-proceso o se puede realizar constantemente durante la búsqueda. Las inconsistencias locales son los valores individuales o la combinación de valores de las variables que no pueden participar en una solución, ya que no satisfacen alguna propiedad de consistencia (Mackworth 1977).

Por ejemplo, si un valor a de la variable x_i no es compatible con ningún valor de una variable x_j, debido a las restricciones en común entre x_i y x_j, entonces a es inconsistente y este valor se puede eliminar del dominio de la variable x_i. En los párrafos siguientes se introducen los algoritmos más conocidos y más usados para CSPs binarios.

- Un CSP es nodo-consistente si todas sus restricciones unarias se mantienen para todos los elementos del dominio. El algoritmo de nodo-consistencia sencillo (NC), que elimina los elementos redundantes mediante la comprobación de los dominios uno tras otro, tiene $O(dn)$ de complejidad de tiempo, donde d es el tamaño máximo de los dominios y n el número de variables del problema. Por lo tanto, mediante la aplicación de esta técnica de consistencia se asegura de que todos los valores de la variable satisfacen todas las restricciones unarias de esa variable.

- Un CSP es arco-consistente (Mackworth 1977) si para cada par de variables restringidas x_i, x_j, y para cada valor a en D_i, hay por lo menos un valor b en D_j, de tal forma que, la asignación (x_i, a) y (x_j, b) satisface la restricción entre x_i y x_j. Cualquier valor en el dominio D_i de la variable x_i que no sea arco-consistente puede ser borrado de D_i, ya que, no puede formar parte de ninguna solución. La arco-consistencia es una técnica muy importante para la resolución de CSPs y es la base de muchos lenguajes de programación de restricciones. Los algoritmos óptimos para hacer que un CSP sea arco-consistente requieren un tiempo $O(ed^2)$, donde e es el número de restricciones (arcos en la red de restricción) y d es el tamaño de los dominios. La arco-consistencia también se puede extender fácilmente a las restricciones no binarias.

- Un CSP es camino-consistente (Montanari 1974) si consideramos para cada par de valores a y b dos variables x_i y x_j, de tal manera que las asignaciones de (x_i, a) y

(x_j, b) satisfacen la restricción entre x_i y x_j, existiendo un valor para cada variable a lo largo de cualquier camino entre x_i y x_j tales que se cumplen todas las restricciones. Cuando un problema camino-consistente también es nodo-consistente y arco-consistente, entonces el problema se dice que es fuertemente camino-consistente.

Las técnicas de consistencia pueden ser usadas durante la fase de comprobación hacia adelante (forward checking) del algoritmo de búsqueda de la siguiente manera. Cada vez que se realiza alguna decisión de búsqueda (por ejemplo, cuando se asigna un valor a la variable), el problema se hace arco-consistente (arco-consistente se utiliza típicamente durante la búsqueda debido a que se realiza en poco tiempo y tiene poca complejidad). Si se detecta un fallo (cualquier dominio se convierte en vacío), entonces no es necesario instanciar otras variables y el retroceso (backtracking) se produce inmediatamente.

2.6.2 Técnicas de búsqueda

Los algoritmos de resolución de CSPs buscan la resolución del problema mediante la asignación de variables.La desventaja es que estos algoritmos pueden necesitar mucho tiempo. Las acciones de muchos algoritmos de búsqueda se pueden representar como un árbol de búsqueda.

Prueba y Error (Generate and Test). El método generate-and-test (GT) se origina con la intención de resolver problemas de combinatoria. En primer lugar, el algoritmo de GT genera una solución y luego se comprueba si esta solución es correcta, es decir, si la solución satisface las restricciones originales. En este paradigma, cada posible combinación de las asignaciones de variables se genera y se prueba para ver si satisface todas las restricciones de forma sistemática. La primera combinación que satisfaga todas las restricciones es una solución. El número de combinaciones consideradas por este método es el tamaño del producto cartesiano de todos los dominios de las variables.

La principal desventaja es que no es eficiente, debido a que genera muchas asignaciones de valores a las variables que pueden ser rechazadas en la fase de comprobación. Por lo tanto, se pueden generar instanciaciones contradictorias al principio del proceso de generación de una solución, y esta inconsistencia no será descubierta hasta que no se asigne un valor a todas las variables y la solución sea comprobada. Visiblemente, se puede conseguir mucha más eficiencia si la validez de la restricción se prueba tan pronto como las respectivas variables sean instanciadas.

Retroceso (Backtracking). Un algoritmo simple para resolver un CSP es la búsqueda Backtracking (BT) (Bitner y Reingold 1975). BT empieza con un conjunto inicialmente vacío de variables instanciadas consistentes y trata de ampliar el conjunto con una nueva variable y un valor para esa variable. Si tiene éxito, el proceso se repite hasta que se incluyen todas las variables. Si no tiene éxito, se considera otro

valor para la última variable añadida. Si esa variable no tiene ningún valor más en el dominio, entonces la variable es retirada del conjunto, y el algoritmo retrocede de nuevo a una variable anterior (backtracking). El algoritmo de regreso más simple se llama retroceso cronológico porque cuando no hay salida el algoritmo vuelve a la variable inmediatamente anterior en el orden de instanciación.

En el método de BT, las variables son instanciadas secuencialmente y tan pronto como todas las variables relevantes para una restricción son instanciadas, la restricción es etiquetada. Si una solución parcial viola cualquiera de las restricciones, el retroceso se realiza para la variable más recientemente instanciada que todavía tiene alternativas disponibles. Claramente, siempre que una instanciación parcial viola una restricción, backtracking es capaz de eliminar un subespacio del producto cartesiano de los dominios de las variables restantes.

2.6.3 Técnicas híbridas

En la sección 2.6.1 se han presentado algunas técnicas de consistencia que borran los valores inconsistentes de los dominios de las variables. En la sección 2.6.2 han sido presentadas algunas técnicas de búsqueda para resolver CSPs. Por lo tanto, las técnicas de consistencia se pueden usar como una etapa de pre-proceso donde las inconsistencias pueden ser detectadas y eliminadas. Adicionalmente, las técnicas de consistencia pueden ser utilizadas durante el proceso de búsqueda formando las llamadas técnicas híbridas. Algunas de estas técnicas híbridas se basan en técnicas de búsqueda hacia adelante y hacia atrás.

- **Algoritmos de búsqueda hacia atrás (Look-Back):** El algoritmo BT puede llegar en muchas ocasiones a una situación inconsistente si tras la asignación de un valor a una variable x_i se incumple una restricción. El modo de resolver dicha inconsistencia es asignar nuevos valores del dominio a la variable x_i, tras haber comprobado todos los valores se retrocedería a la variable x_{i-1} para asignar un nuevo valor y de este modo volver a asignar valores a x_i. Supongamos que un nuevo valor para x_{i-1} existe, pero que no hay ninguna combinación entre x_i y x_{i-1} que cumpla las restricciones del problema, esta circunstancia se llama situación sin salida, ya que la misma situación sin salida se alcanzará en x_i repetidamente hasta que todos los valores de x_{i-1} se hayan explorado. Los algoritmos Look-back tratan de explotar la información del problema para comportarse de manera más eficiente en situaciones sin salida. Al igual que BT, los algoritmos look-back realizan comprobaciones de coherencia hacia atrás (entre la variable actual y las variables anteriores).

  Backjumping (BJ) (Gaschig 1979) es un algoritmo similar a BT excepto que se comporta de una manera más inteligente cuando se encuentra ante una situación sin salida. Si nos encontramos la misma situación del ejemplo anterior con las variables x_i y x_{i-1}, a diferencia del comportamiento de BT que retrocedería a la

variable anterior x_{i-1}, BJ saltaría a la variable (x_j) más profunda (es decir que está más cerca de la raíz del árbol) que esté en conflicto con x_i. Cambiando la instanciación de x_j puede hacer que sea posible encontrar una instanciación consistente para la variable actual. Por lo tanto, BJ evita cualquier trabajo redundante que BT hace al tratar de reasignar variables entre x_j y la variable x_i. Conflict-directed BJ (Prosser 1993), backmarking (Gaschnig 1977) y learning (Frost y Dechter 1994) son también ejemplos de algoritmos look-back.

En el trabajo de Sadeh, Sycara y Xiong 1995 se aplican tres técnicas de backtraking (Dynamic Consistency Enforcement, Learning Ordering From Failure e Incomplete Backjumping Heuristic) sobre el problema JSP y se combinan con un esquema look-back. De este modo se consigue reducir la complejidad de la búsqueda backtracking, permitiendo la resolución de más problemas por la reducción del coste computacional, y disponer de un esquema más eficiente para resolver el problema JSP.

- **Algoritmos de búsqueda hacia adelante (Look-Ahead):** Como hemos explicado, los algoritmos look-back tratan de mejorar el rendimiento de BT con un comportamiento más inteligente cuando se encuentran una situación sin salida. Sin embargo, se siguen realizando solo comprobaciones de coherencia en variables anteriores e ignorando las futuras. Supongamos que, en la búsqueda de una solución, a la variable x_i se le da un valor que es inconsistente con todos los posibles valores para la variable x_j. En el caso de búsqueda desinformada, esto solo se detectará cuando x_j sea instanciada. Además, con BT se producirá que el árbol de búsqueda se amplíe repetidamente hasta x_j, hasta que el nivel de BT llegue hasta x_i. Ambas anomalías podrían evitarse reconociendo que el valor elegido para x_i no puede ser parte de una solución, ya que no hay ningún valor para x_j que sea compatible con ella. Los algoritmos Look-Ahead hacen esto mediante la aceptación de un valor para la variable x_i solo si, después de mirar hacia adelante, no se observa que la instanciación de x_i llevaría a una situación sin salida. Al comprobar esto, la reducción del problema también puede llevarse a cabo mediante la eliminación de los valores del dominio de las variables futuras que no son compatibles con la instanciación de x_i. Los distintos algoritmos Look-Ahead se diferencian en cuanto miran hacia adelante, es decir lo exhaustiva que será la búsqueda y por lo tanto la dimensión de valores eliminados de los dominios.

  Forward-checking (FC) (Haralick y Elliott 1980) es uno de los algoritmos Look-Ahead más comunes. Comprueba la satisfacibilidad de las instanciaciones y elimina los valores que no son compatibles con la creación de instancias de la variable actual. En cada paso, FC comprueba la asignación actual con todos los valores de las variables futuras que están restringidas con la variable actual. Todos los valores de las variables futuras que no son consistentes con la asignación actual se eliminan de sus dominios. Si un dominio de una variable futura se convierte en vacío, la

asignación de la variable actual se deshace y se le asigna un nuevo valor. Si no hay ningún valor consistente, entonces BT se lleva a cabo. Por lo tanto, FC garantiza que en cada paso la solución parcial actual es consistente con cada valor de cada variable futuro. En consecuencia, FC puede identificar situaciones sin salida y podar el espacio de búsqueda de forma más eficiente.

El algoritmo Look-Ahead ha sido utilizado para resolver muchos problemas de scheduling. En el trabajo de Sadeh 1991, este algoritmo es usado para resolver dos variantes del problema JSP. En la primera variante las operaciones se deben realizar dentro de unas ventanas de tiempo no relajables y en la segunda variante se presenta el problema factory scheduling.

2.6.4 Ejemplos de Aplicación

Son muchos los problemas que se pueden resolver mediante la aplicación de técnicas de programación basada en la especificación de restricciones y en esta sección se muestran unos ejemplos de problemas reales de scheduling resueltos con alguna de estas técnicas.

Un claro ejemplo de aplicación es el que se realiza en el trabajo Ingolotti y col. 2006 donde se resuelve el problema de planificación de horarios de trenes mediante el modelado como un CSOP. Se hace uso de técnicas meta-heurísticas que ayudan a resolver el problema mediante el uso del conocimiento específico de este. Se trata de un problema multi-objetivo donde factibilidad, eficiencia computacional y optimalidad son tenidas en cuenta. En el trabajo realizado por Khemmoudj, Porcheron y Bennaceur 2006 se resuelve el problema de los cortes en las centrales nucleares. Una central nuclear puede necesitar ser parada de repente para realizar reabastecimiento de combustible o mantenimiento, para la realización del corte se deben de cumplir ciertas restricciones de seguridad, logística de mantenimiento y operación de la planta. Es por ello que su modelado con programación de restricciones es muy oportuno, además se utilizan técnicas de búsqueda local para mejorar la búsqueda de soluciones. Otro ejemplo del uso de CSOPs es el mostrado en el trabajo de Lau y Tsang 1998 donde se trata el problema de una terminal de contenedores. En el artículo de Cambazard y col. 2005 se resuelve el problema de planificación de horarios del colegio mediante una extensión de programación de restricciones, ya que se dispone de la opción de hacer uso de relajaciones automáticas. Mediante la herramienta diseñada se consigue realizar la planificación en pocos minutos en lugar de la semana que se tardaría si se realizara la tarea a mano.

2.7 Conclusiones

En este capítulo se ha mostrado las características del problema Job-Shop scheduling y distintos métodos o técnicas para resolverlo. Se ha mostrado especial interés en la programación con restricciones, y como esta técnica ha sido utilizada en numerosas ocasiones para resolver problemas de scheduling.

El problema de Job-Shop scheduling puede representar a muchos problemas reales de planificación del sector de Manufacturing o de procesos industriales. Mediante este problema y sus derivados se pueden desarrollar y evaluar distintas técnicas. Con el uso de las extensiones del problema se puede modificar el problema para que se parezca a cierto grupo de problemas reales. En definitiva, los problemas de scheduling tratan de planificar ciertas tareas siguiendo un conjunto de restricciones y tratando de optimizar ciertos criterios ,es por ello, que modelar el problema como un CSOP puede resultar interesante.

Capítulo 3

El Problema de Job Shop Scheduling con Operadores

3.1 Introducción

El problema JSP puede ser modificado para que represente de una forma más cercana cierto tipo de problemas reales. En este caso se ha modificado el problema incorporando el recurso de los operadores. Esta extensión del problema fue propuesta por Agnetis y col. 2011 y el problema se identifica como $JSO(n, p)$ donde n representa el número de trabajos y p es el número de operadores. El número de trabajos n tiene que ser mayor que el número de operadores p, de lo contrario cada operador podría ser asignado a cada trabajo y entonces estaríamos tratando con un JSP estándar. Además de extender el problema, se ha incorporado a la función de optmización la robustez convirtiendo así el problema en multiobjetivo. En la sección 3.2 se definen los conceptos de robustez y estabilidad para problemas de scheduling. En la sección 3.3 modelamos y resolvemos el problema $JSO(n, p)$ como un CSOP para minimizar el makespan. En la sección 3.4 se desarrolla una técnica de 3 pasos que resuelve el problema $JSO(n, p)$, pero en este caso se optimiza tanto el makespan como la robustez. En la sección 3.5 se muestran y analizan los resultados obtenidos por ambas técnicas.

La extensión al problema $JSO(n, p)$ respecto al JSP produce algunos cambios en la formulación del problema:

- Una nueva restricción es añadida: Cada operación tiene que ser asistida por un operador y cada operador no puede asistir más de una operación al mismo tiempo.

- El grafo de representación G sufre algunos cambios: Al representar el problema JSP G estaba definido como (V, A, E) donde V representa los nodos, A las relaciones de precedencia y E las limitaciones de capacidad. El nuevo grafo está representado como $G = (V, A \cup E \cup I \cup O)$ (Sierra, Mencía y Varela 2011) donde los nodos siguen estando representado por V, las relaciones de precedencia por los arcos A y las limitaciones de capacidad por los arcos E. Ahora se incorporan los arcos de operadores O que incluye 3 tipos de arcos: un arco (u, v) para cada par de tareas del problema, un arco (O_i^{start}, u) y (u, O_i^{end}) para cada nodo de operador y tarea. El conjunto I representa los arcos conectando el nodo *start* a cada nodo O_i^{start} y arcos conectando cada nodo O_i^{end} al nodo *end*. Los arcos se ponderan con el tiempo de procesamiento de la tarea en el nodo fuente.

Este problema es estudiado en el trabajo de Agnetis y col. 2011 donde se establecen los casos mínimos como NP-hard. Además, se propone una serie de algoritmos exactos y aproximados para hacer frente a este problema y se evalúa en un conjunto de instancias generadas a partir de los casos mínimos citados anteriormente.

Los resultados del estudio experimental mostrados en Agnetis y col. 2011 dejan claro que las instancias de $JSO(n, p)$ con tan solo 3 trabajos, 3 máquinas, 2 operadores y 30 tareas por trabajo pueden ser difíciles de resolver de forma óptima. Además, en este capítulo se añade a la función objetivo la robustez. De tal forma, que se buscan soluciones que minimicen el makespan pero que al mismo tiempo sean capaces de absorber pequeñas incidencias que puedan aparecer en la solución. En ocasiones, se dispone de conocimiento sobre las incidencias, pero este caso no existe ninguna información, por lo tanto, la solución más robusta suele ser la solución en la que más tareas tienen la posibilidad de absorber incidencias.

3.2 Robustez y Estabilidad en Scheduling

La robustez puede estar relacionada con muchos problemas del mundo real. En la industria la robustez puede resultar rentable, porque si una perturbación aparece en un schedule este puede seguir siendo válido y por lo tanto no es necesario replanificar ni interrumpir el proceso de producción. Consideramos que un sistema es robusto si es capaz de mantener su funcionalidad a pesar de que pueden aparecer algunas incidencias. En scheduling una solución es robusta si es capaz de absorber algunas interrupciones sin retrasar las tareas sucesoras.

En los problemas del mundo real es muy común dejar algunos “gaps” o “buffers” entre las tareas consecutivas del schedule. En ocasiones estos gaps se generan para satisfacer las restricciones de precedencia. Se considera que una solución S es robusta si al aparecer una incidencia I esta solo afecta a la tarea en la cual ha aparecido la incidencia Billaut,

Moukrim y Sanlaville 2008. De modo que, la incidencia no se propaga al resto de tareas. Por lo tanto, la robustez de una solución puede ser medida de la siguiente forma:

- $R_{S,I} = 1$ Si solo la tarea afectada por I es modificada. Por lo tanto un "gap" o "buffer" asignado a esta tarea puede absorber la incidencia.
- $R_{S,I} = 0$ Si más de una tarea se ven afectadas por I. Esto significa que el "buffer" asignado a esta tarea no puede absorber la incidencia y se propaga al resto del schedule.

Por lo tanto, una solución robusta es una solución que mantiene su viabilidad ante la aparición de incidencias. Si una solución no es robusta frente a una perturbación, esta puede ser considerada como estable. Una solución es considerada estable si tras sufrir unas incidencias con solo la realización de unos pocos cambios es capaz de ser válida(Jen 2003). Así que la estabilidad puede resultar interesante para la replanificación, a fin de, recuperar la solución original por medio de la modificación de solo unas pocas tareas.

Se considera que una solución S es $s\text{-}estable$ si existe una solución S', de manera que $||S' - S|| < s$, donde $||.||$ es cierta n-dimensionalidad definida en el espacio de soluciones para evaluar la diferencia entre S y S' Barber y Salido 2014. Así que en scheduling, la diferencia entre S y S', por ejemplo $||.||$ podría ser el número de tareas que tienen que cambiar para mantener la factibilidad.

3.3 Modelización del problema $JSO(n,p)$ como CSOP

En esta sección el problema $JSO(n,p)$ es modelado como un CSP y resuelto usando un algoritmo look-ahead (Forward-checking). Mediante el uso de este algoritmo se puede obtener una solución. Sin embargo, el objetivo en este caso no sería obtener una solución válida sino una solución optimizada. Por ello, se ha desarrollado una técnica branch and bound, de modo que, si una solución es encontrada, el algoritmo continúa buscando para encontrar más soluciones que mejoren los resultados de las soluciones ya obtenidas. Cuando el objetivo no es solo obtener una solución, sino que se pretende obtener una solución optimizando ciertos criterios, estamos hablando de un Problema de Satisfacción y Optimización de Restricciones (CSOP).

3.3.1 Fase de Modelado

En esta sección se modela el problema $JSO(n, p)$ como un Problema de Satisfacción y Optimización de Restricciones (CSOP) (Beck 1994). El modelo CSOP para un $JSO(n, p)$ está caracterizado por los siguientes elementos:

- Un conjunto de variables $st_{\theta_{ij}}, ..., st_{\theta_{pq}}$ asociadas con el tiempo de comienzo de las tareas. Estas variables tomarán un valor en el dominio finito de valores $d_{ij}, ..., d_{pq}$ y serán limitadas por restricciones unarias sobre cada variable. En estos problemas, el tiempo se asume normalmente como discreto.
- Un conjunto de restricciones $c_1, ..., c_m$ de entre las variables definidas en el producto cartesiano $d_i \times ... \times d_j$ y restringidas por los dominios de las variables.
- La función objetivo es minimizar el makespan.

El problema está compuesto por cuatro tipos de restricciones:

1. Restricciones de precedencia: Las tareas θ_{ij} de cada trabajo J_i tienen que ser planificadas acorde a unas restricciones de precedencia. Por ejemplo, existe un orden entre las tareas de cada trabajo y puede ser representado por un grafo de precedencia (Sadeh y Fox 1996).
2. Restricciones de capacidad: Los recursos no pueden ser usados simultáneamente por más de una tarea. Así que, dos tareas diferentes θ_{ij} y θ_{ik} no se pueden superponer a menos que usen diferentes recursos.
3. Restricciones de Operadores: Un operador no puede asistir más de una operación al mismo tiempo.
4. No se permiten preferencias entre las tareas.

XCSP Abscon

Para el modelado del problema hemos utilizado la sintaxis XCSP (Roussel y Lecoutre 2009). XCSP usa el lenguaje Extensible Markup Language (XML). EL lenguaje XML presenta una forma simple y flexible de representar la información y además da la facilidad de poder usar funciones y estructuras definidas en Abscon Lecoutre y Tabary 2008. En la figura 3.1, se puede observar un ejemplo de la representación de una instancia con la sintaxis XCSP.

Figura 3.1: Instancia parcial de un JSP representado en XCSP.

Técnicas de filtrado para reducir el dominio de las variables

Cuando más reducido sea el dominio de las variables más pequeño será el espacio de búsqueda y por lo tanto más fácil será resolver el problema. Por lo tanto hemos desarrollado dos técnicas para reducir el dominio para el problema de JSP en concreto.

La primera técnica desarrollada calcula los valores iniciales para cada tarea siguiendo las reglas de nodo-consistencia y arco-consistencia, de modo que se calcula el intervalo máximo en el que puede ser planificada cada tarea (algoritmo 1). En el algoritmo 1 primero se calcula para cada tarea θ_{ij} el valor más pequeño de la suma de los tiempo de procesado para las tareas que deben de ser planificadas antes por las restricciones de precedencia ($cumulative_{ij}$). Es decir la suma del tiempo de procesado mínimo para las tareas del trabajo i que deben de ser planificadas antes que θ_{ij}. Además el algoritmo calcula el valor máximo para todos los dominios de todas la variables siendo este la suma de todos los tiempo de procesado ($maxTime$), este sería el valor si las tareas fueran planificadas de forma lineal. Pero debido a que las tareas del mismo trabajo que son ejecutadas más tarde

tienen que ser planificadas antes de $maxTime$ el valor máximo para el dominio de cada variable θ_{ij} puede ser reducido restando la duración de las tareas del mismo trabajo i que deben de ser planificadas más tarde θ_{ij} (teniendo en cuenta el tiempo de procesado de θ_{ij}).

Algoritmo 1: Calculo de valores iniciales para reducir el dominio

Data: J: conjunto de trabajos;
Result: Comienzos relativos para cada tarea y maxtime
$maxTime = 0$;
$cumulative_{ij} = 0,\ \forall \theta_{ij} \in \theta$;
foreach $i \in J$ **do**
    $cumulativeJob = 0$;
    foreach $\theta_{ij} \in \theta$ **do**
        $maxTime = maxTime + p_{\theta_{ij}}$;
        $cumulative_{ij} = cumulativeJob$;
        $cumulativeJob = cumulativeJob + p_{\theta_{ij}}$;
return $cumulative_{ij}, maxTime$;

Los valores $cumulative_{ij}$ y $maxTime$ obtenidos por el algoritmo 1 son usados para filtrar los dominios eliminando, los valores que no pueden ser usados para obtener soluciones factibles.

Con la segunda técnica para reducir los dominios, los valores que no pueden ser posibles, son calculados, ya que una tarea solo podrá empezar $st_{\theta_{ij}}$ por los valores representados como las sumas de las tareas que puedan ser ejecutadas delante de ella θ_{ij}.

Estos valores son calculados por el algoritmo 2. Por ejemplo si una tarea θ_{ij} es la primera de su trabajo los posibles valores que podrá tomar será 0 y el conjunto de combinaciones posibles de todas las tareas T que puedan ser ejecutadas antes que ella, en este caso T serían las tareas del resto de trabajos. Para la siguiente tarea del mismos trabajo θ_{ij+1}, los posibles valores serían los mismos que θ_{ij} más el tiempo de procesado de θ_{ij}.

En el algoritmo 2, la función *tasksCanBeBefore* devuelve la tarea que puede ser ejecutada delante de una tarea dada θ_{ij}; y, la función *combineDurTask* calcula todos los posibles valores para $st_{\theta_{ij}}$ siguiendo las restricciones de preferencia.

En consecuencia, aplicando las técnicas de filtrado, los dominios de las variables pueden ser reducidos y por lo tanto la fase de resolución puede ejecutarse de una forma más eficiente.

Algoritmo 2: Calcular valores posibles

Data: Todas las tareas
Result: Dominio nuevo para todas θ
$pValues_{jt} \leftarrow \emptyset,\ \forall \theta_{ij} \in \theta$;
foreach $i \in J$ **do**
    foreach $\theta_{ij} \in \theta$ **do**
        $tasksBefore \leftarrow$ `tasksCanBeBefore`(θ_{ij})
        $pValues_{ij} \leftarrow$ `combineDurTasks`$(tasksBefore)$
return $pValues$

3.3.2 Fase de Resolución

Una vez ha sido modelado el problema y los dominios han sido filtrados por medio de las técnicas de nodo-consistencia y arco-consistencia, este se resuelve usando una extensión del algoritmo Forward Checking (FC) (algoritmo 3). Cuando una solución ha sido encontrada, el algoritmo intenta encontrar más soluciones mediante el uso de una técnica Branch and Bound (B&B). En la ejecución de esta técnica también se han usado técnicas de filtrado, de modo que, se podan soluciones del árbol de búsqueda si las soluciones podadas no pueden mejorar a la mejor solución obtenida hasta el momento (Mencía y col. 2015). De esta manera se consigue que el algoritmo sea completo y correcto.

En lugar de aplicar Arco-Consistencia, este algoritmo realiza un procedimiento de filtrado que elimina los dominios en bloques, y por lo tanto, varios valores pueden ser eliminados al mismo tiempo. Esto se debe a que si una tarea θ_{ij} tiene que ser planificada después de una tarea θ_{kl}, el comienzo de la tarea ij ($st_{\theta_{ij}}$) nunca podrá ser menor o igual al valor de comienzo de la tarea kl ($st_{\theta_{kl}}$) más su duración. Por lo tanto, todos los valores entre $st_{\theta_{kl}}$ y el fin de la tarea θ_{kl} pueden ser eliminados del dominio de θ_{ij}. Los valores a borrar dependerán del tipo de restricción. Si la restricción es de precedencia, todos los valores anteriores al fin de la tarea θ_{kl} serán eliminados. En cambio si la restricción es de capacidad, serán eliminados los valores entre $st_{\theta_{kl}}$ (incluido) y el tiempo final de procesado.

Algunas técnicas heurísticas han sido aplicadas para mejorar la eficiencia del algoritmo. La selección de variables y la selección de valor son dos tareas críticas, y con una buena técnica heurística puede mejorar los resultados. La selección de variables es muy importante porque una buena selección puede suponer obtener más rápidamente la solución y/o mejorar la poda, en este caso hemos seleccionado en primer lugar las primeras tareas de cada trabajo. La selección de valor también es importante, en este caso se ha seleccionado los valores de las variables temporales en orden creciente y se ha escogido el valor más pequeño para cada variable temporal. Esta es la forma más correcta de seleccionar las variables porque si una tarea puede comenzar en un dominio de intervalo de tiempo, el primer valor será probablemente el valor que minimice el makespan.

Algoritmo 3: Selección de valores Forward-Checking con filtrado por bloque

while $D'_i \neq \emptyset$ **do**
  Seleccionar primer elemento $a \in D'_i$, y eliminar a de D_i
  forall the $k,\ i < k \leq |D'|$ **do**
    if *Restricción(i,k) == MáquinaOJobRestricción* **then**
      if *Restricción(i,k)== JobRestricción* **then**
        Eliminar valores if($val_k < a + dur_i$) from D'_k
      else
        Eliminar valores if($val_k + dur_k \geq a \wedge val_k < a + dur_i$) from D'_k
    else
      forall the $b \in D'_k$ **do**
        if *no* `CONSISTENT` *(*a_{i-1},$x_i := a, x_k := b$*)* **then**
          Eliminar b de D'_k
    if $D'_k = \emptyset$ **then**
      Restablecer cada $D'_k,\ i < k \leq n$ a valores antes que a fuera seleccionada
    else
      return a
return null

3.4 Desarrollo de Técnica de 3 Pasos

En la sección 3.3 se muestra como se modela y se resuelve el modelo $JSO(n, p)$ como un CSOP. Debido a la complejidad y la necesidad de obtener soluciones óptimas de forma eficiente proponemos una técnica que hemos llamado Técnica de 3 Pasos (Escamilla, Rodriguez-Molins y col. 2012).

En el primer paso, se resuelve el problema job-shop scheduling sin tener en cuenta los operadores y con el objetivo de minimizar el makespan. En el segundo paso, esta solución se modifica para tener en cuenta los operadores, modificando el problema a $JSO(n, p)$. Ahora el objetivo es obtener una solución donde se minimice el makespan y se maximice la robustez. En el tercer paso, la solución es modificada para distribuir los buffers temporales sin modificar el makespan obtenido.

3.4.1 Modelado y Resolución CSOP

En la primera parte, se modela un JSP como un CSOP. Debido a que mediante los 2 pasos siguientes la solución será modificada no es necesario encontrar la solución óptima, aunque sí que es necesario que sea una solución optimizada donde se minimice el makespan. Para obtener una solución cercana a la óptima se ha utilizado nuestro resolutor CSOP, aunque cualquier resolutor podría haber sido utilizado. Nuestro resolutor proporciona un grupo de soluciones donde cada nueva solución siempre mejora a la anterior. El algoritmo se detendrá cuando la solución óptima haya sido encontrada o cuando se alcance un tiempo máximo establecido.

Las fases de modelado y resolución son parecidas a las mostradas en las sección 3.3 pero sin tener en cuenta a los operadores. En este caso, solo las restricciones de precedencia, de capacidad y de no preferencia serán tenidas en cuenta. Para modelar el problema, el algoritmo de reducción del dominio (algoritmo 1) y el algoritmo para calcular los valores posibles (algoritmo 2) han sido usados para calcular los valores iniciales para cada tarea y para reducir el dominio. Estos algoritmos no tienen en cuenta los operadores y por lo tanto pueden ser usados en el problema JSP clásico.

En la parte de resolución se pueden utilizar las técnicas presentadas en la sección 3.3. Por lo tanto el problema es resuelto utilizando una extensión del algoritmo Forward Checking (FC) (algoritmo 3).

3.4.2 Post-Proceso (PP)

Una vez el CSOP ha sido resuelto, y tenemos una solución optimizada del problema job-shop scheduling, esta solución se utiliza para introducir los operadores y por lo tanto poder obtener una solución (CSOP+PP) para el problema $JSO(n, p)$. Para ello se busca una solución factible minimizando el makespan y maximizando el número de buffers (N_{buf}), con lo que conseguiremos garantizar un cierto nivel de robustez. Hay que tener en cuenta que una solución factible para el problema $JSO(n, p)$ también lo es para el clásico problema JSP. Por lo tanto, los casos significativos son aquellos en los que $p < min(n, m)$, de lo contrario nuestro problema se convierte en un problema JSP estándar (Agnetis y col. 2011).

El objetivo del algoritmo 4 es convertir una solución sin operadores en una donde la restricción de operadores sea considerada. La idea es establecer un número de máquinas ($remainingMachines$) igual al número de operadores p e intentar planificar las tareas de las otras máquinas ($machinesFewerTasks$) dentro de un conjunto llamado $remainingMachines$. Las tareas de las otras máquinas tienen que ser ordenadas por su tiempo de comienzo st ($tasksToPut$). Cada tarea θ_{ij} se encuentra en un conjunto llamado $tasksToPut$ y se sitúa en el primer hueco disponible entre las dos tareas de cada máquina del conjunto de tareas a planificar $remainingMachines$. Para cada máquina la búsqueda se inicia desde el estado anterior ($savedState$). Hay casos donde la tarea θ_{ij} tiene que ser colocada sin que haya un hueco entre las dos tareas, para preservar la restricción de precedencia. Por ejemplo, si se encuentra una tarea θ_{ik} del mismo trabajo que la tarea θ_{ij} y la tarea θ_{ik} tiene que ser planificada después de la tarea θ_{ij} según la restricciones de precedencia ($k > j$), la tarea θ_{ij} es colocada justo antes de la tarea θ_{ik}, y por lo tanto la tarea θ_{ik} será retrasada. Cuando una tarea es retrasada, otras tareas pueden ser también retrasadas. El coste computacional del algoritmo es $O(tasksToPut * |remainingMachines|)$.

El mejor estado para colocar a la tarea θ_{ij} es el estado donde la función $\frac{N_{buf}}{C_{max}}$ se maximice. Esta función puede ser ajustada dependiendo de los requisitos del usuario, por

Algoritmo 4: Post-Proceso

Data: S: Solución sin operadores; m: máquinas; p: operadores;
Result: Una solución considerando operadores

Ordenar las máquinas por el número de tareas;
$machinesFewerTasks \leftarrow$ conjunto de $m - p$ máquinas con menos tareas;
$tasksToPut \leftarrow$ tareas de las máquinas $machinesFewerTasks$;
$remainingMachines \leftarrow m - machinesFewerTasks$;
Ordenar $tasksToPut$ por el tiempo de comienzo;

$actualState \leftarrow S$;
foreach $\theta_i \in tasksToPut$ **do**
    $savedState \leftarrow actualState$;
    $states \leftarrow \{\}$;
    for $r \in remainingMachines$ **do**
        $foundGap :=$ `HayUnGap`(r);
        if $not\ foundGap$ **then**
            insertar θ_i antes de la siguiente tarea en función de su trabajo;
        else
            insertar θ_i es este gap;
        Retrasar las tareas necesarias de acuerdo con las restricciones entre ellas;
        $states \leftarrow states \cup \{actualState\}$;
        $actualState \leftarrow savedState$;
    $actualState \leftarrow$ `elegirMejorEstado`$(states)$;

return $actualState$;

ejemplo, incluyendo una constante para dar más peso al makespan o al número de buffers generados.

3.4.3 Método de Distribución de Buffers

Mediante los pasos previos se obtiene una solución optimizada que satisface todas las restricciones del $JSO(n, p)$. Esta solución esta optimizada en términos de minimización del makespan y maximización del número de buffers. Sin embargo, cuando el objetivo principal es obtener soluciones robustas donde no hay información sobre la aparición de incidencias, la mejor opción es distribuir los buffers disponibles para cubrir el mayor número de tareas. A las tareas que están en un camino crítico no se les puede asignar un buffer, ya que esto repercutiría directamente en el makespan. Sin embargo, el resto de tareas son candidatas a que se les pueda asignar un buffer, redistribuyendo uno existente, siempre y cuando esto no afecte al makespan. De esta forma se puede maximizar el número de buffers y por lo tanto obtener una solución más robusta, ya que más tareas serán capaces de absorber pequeñas incidencias. La figura 3.2(b) muestra una solución obtenida por el paso 2, donde el número de buffers es 10. En la figura 3.2(c) se puede observar una solución tras distribuir los buffers y en la que se ve que aumenta a 14. Hay que destacar que nuestro objetivo es obtener el máximo número de buffers, ya que no se tiene información sobre incidencias de manera que, todas las tareas tienen la misma probabilidad de sufrir una incidencia.

Algoritmo 5: Distribución de Buffers

Data: Sch: Schedule; $buffers$;
Result: Schedule nuevo

```
foreach b ∈ buffers do
    sizeB :=tamaño del buffer b;
    repeat
        continue := false;
        θ_ij := tarea asignada antes b;
        Retrasar θ_ij;
        if Este movimiento genera otro buffer nb then
            continue := true;
            Actualizar schedule Sch;
    until continue;
return Sch;
```

El algoritmo 5 representa el tercer paso de la técnica. En el algoritmo se buscan las tareas que tienen un buffer tras su ejecución y se intenta retrasar esta tarea para generar un buffer delante. Una tarea θ_{ij} será retrasada si un nuevo buffer (nb) es generado. Si es así, este proceso será repetido con la tarea que está justo delante del nuevo buffer. El coste computacional de este algoritmo es $O(\text{tareas que no están en el camino crítico})$, ya que las tareas que están en el camino crítico no pueden ser movidas para distribuir los buffers.

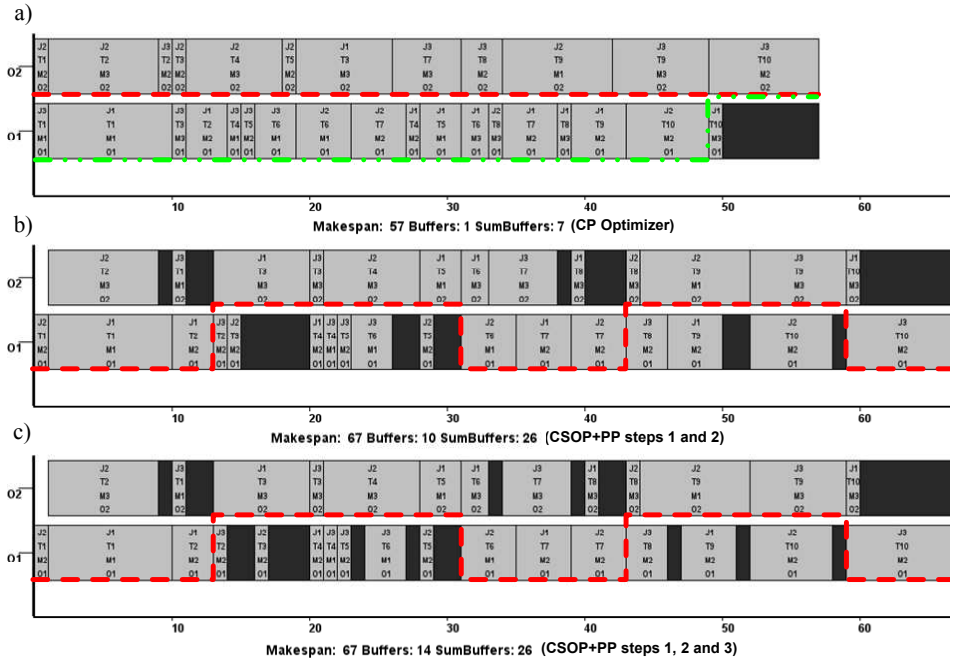


Figura 3.2: Soluciones a un JSP: Una solución óptima y soluciones robustas.

3.4.4 Ejemplo

La figura 3.2 muestra diferentes schedules obtenidos por el resolutor CP Optimizer y nuestra técnica para una instancia de $JSO(n, p)$. Esta instancia representa un problema de scheduling con 3 trabajos, con 10 tareas cada trabajo y 2 operadores. Cada rectángulo representa una tarea donde su longitud corresponde a su tiempo de procesado. Dentro del rectángulo se muestra el trabajo, tarea, máquina y operador de la tarea. Las líneas de puntos representan los posibles caminos críticos del schedule.

El schedule de la figura 3.2(a) representa una solución obtenida por el CP Optimizer (solo minimizando el makespan), la solución se muestra con respecto a los dos operadores. Se puede observar que el makespan obtenido es 57, pero solo se genera un buffer, ya que, $\theta_{1,10}$ es la única tarea que no está en un camino crítico. En la figura 3.2(b) se representa la solución obtenida aplicando los pasos 1 y 2 de nuestro algoritmo, es decir, teniendo en cuenta la robustez en la función de optimización. Se puede observar que el número de buffers crece hasta 10 y que el makespan hasta 67.

Finalmente, en la figura 3.2(c) se representa el schedule obtenido por el tercer paso de nuestro algoritmo. Se puede observar que los buffers (cajas negras) se distribuyeron entre

todas las tareas que no formaban parte de un camino crítico. De este modo, se incrementa la robustez del schedule, mientras que el makespan permanece estable. Estos buffers se pueden utilizar para absorber incidencias o retrasos en tareas. Por ejemplo, si la máquina asignada a la tarea θ_{23} sufre una pequeña incidencia, la solución podría no verse afectada. En este ejemplo, con una pérdida de 10 unidades de tiempo hemos aumentado en 13 el número de tareas que podrían ser capaces de absorber pequeñas incidencias.

3.5 Evaluación

Para la realización de la evaluación se ha utilizado el benchmark propuesto en Agnetis y col. 2011, donde todas las instancias tienen tres trabajos ($n = 3$) y dos operadores ($p = 2$) y están caracterizadas por el número de máquinas (m), el número máximo de tareas por trabajo (v_{max}) y el rango de los tiempos de procesamiento (p_i). El conjunto de instancias se compone combinando tres valores para cada parámetro: $m = 3, 5, 7$; $v_{max} = 5, 7, 10$ y $p_i = [1, 10], [1, 50], [1, 100]$. En todos los casos, 10 instancias fueron consideradas para cada combinación y en las tablas siguientes se muestran los resultados del promedio de las 10 instancias. Las instancias se identifican como: $< m\_v_{max}\_p_i >$.

3.5.1 Evaluación del Modelo CSOP

El conjunto de problemas fue resuelto aplicando las técnicas explicadas en la sección 3.3. El principal objetivo es encontrar una solución que minimice el makespan. Las tablas 3.1, 3.2 y 3.3 muestran los resultados de las instancias con 10, 50 y 100 unidades máximas de tiempo de procesado. Para cada instancia se ejecuta nuestra técnica durante un tiempo máximo de 300 segundos. La columna $\mathbf{C_{max}}$ representa el makespan medio obtenido para las 10 instancias, la columna "Tiempo" representa el tiempo computacional medio (en segundos) utilizado para encontrar la mejor solución, la columna Óptimo $\mathbf{C_{max}}$ representa el makespan óptimo medio para las 10 instancias y la columna Dif $\mathbf{C_{max}}$ representa la diferencia entre el makespan obtenido y el óptimo.

Para las instancias donde $p_i = [1, 10]$, el makespan obtenido es muy cercano al óptimo porque el dominio temporal es más pequeño y en consecuencia el espacio de búsqueda es también más pequeño. La diferencia media de makespan cuando $p_i = [1, 10]$ fue de 1.73 unidades de tiempo. Sin embargo, para las instancias donde $p_i = [1, 50]$ fue de 20.41 unidades y para las instancia cuando $p_i = [1, 100]$ fue de 41.81 unidades. Este aumento está justificado porque cuanto mayor es el dominio temporal, mayor es el espacio de búsqueda. Además, hay que tener en cuenta que, para las instancias con un tiempo de procesado mayor el makespan también será mayor. Es por ello, que se ha calculado el porcentaje del incremento del makespan para las tablas 3.1, 3.2 y 3.3 siendo este 3.27 %, 7.53 % y 7.46 % respectivamente. De este modo, se puede ver que para las instancias

Tabla 3.1: Makespan y tiempo computacional ($p_i = [1, 10]$)

Instancias	**CSOP con operadores**			
	$\mathbf{C_{max}}$	Tiempo	Óptimo $\mathbf{C_{max}}$	Dif $\mathbf{C_{max}}$
3_5_10	44,20	46,35	42,70	1,50
3_7_10	60,90	63,90	57,90	3,00
3_10_10	68,70	24,56	65,20	3,50
5_5_10	40,30	43,84	39,40	0,90
5_7_10	55,10	70,00	53,90	1,20
5_10_10	63,30	15,83	60,60	2,70
7_5_10	32,10	3,00	31,00	1,10
7_7_10	46,30	11,92	44,70	1,60
7_10_10	72,20	22,18	70,40	1,80

Tabla 3.2: Makespan y tiempo computacional ($p_i = [1, 50]$)

Instancias	**CSOP con operadores**			
	$\mathbf{C_{max}}$	Tiempo	Óptimo $\mathbf{C_{max}}$	Dif $\mathbf{C_{max}}$
3_5_50	218,90	49,29	201,10	17,80
3_7_50	296,10	4,68	267,20	28,90
3_10_50	385,80	7,72	353,30	32,50
5_5_50	200,20	22,28	184,30	15,90
5_7_50	274,00	36,93	253,50	20,50
5_10_50	395,40	18,55	363,00	32,40
7_5_50	186,00	1,77	177,10	8,90
7_7_50	275,10	19,36	252,00	23,10
7_10_50	388,00	12,46	363,90	24,10

Tabla 3.3: Makespan y tiempo computacional ($p_i = [1, 100]$)

Instancias	**CSOP con operadores**			
	$\mathbf{C_{max}}$	Tiempo	Óptimo $\mathbf{C_{max}}$	Dif $\mathbf{C_{max}}$
3_5_100	412,80	36,94	389,00	23,80
3_7_100	617,80	3,93	561,40	56,40
3_10_100	858,80	37,45	768,70	90,10
5_5_100	391,60	9,42	349,00	42,60
5_7_100	536,40	53,43	495,00	41,40
5_10_100	771,60	55,84	698,20	73,40
7_5_100	411,90	33,70	391,40	20,50
7_7_100	533,40	12,80	516,00	17,40
7_10_100	797,30	24,99	744,80	52,50

donde el dominio temporal es menor, las instancias son más fáciles de resolver que cuando el dominio es mayor.

Otro resultado a analizar es que cuando el número de máquinas incrementa, las soluciones mejoran. Esto sucede porque el problema está más restringido, y por lo tanto, el espacio de búsqueda se reduce. Esto se puede observar en la tabla 3.3, donde el valor más bajo para Dif $\mathbf{C_{max}}$, si se compara entre el mismo valor de $v_{max} = 5, 7, 10$, se ha obtenido en las instancias con 7 máquinas.

3.5.2 Evaluación del Modelo de Técnica de 3 Pasos

El objetivo de esta experimentación es evaluar nuestra técnica de 3 pasos y compararla con la herramienta IBM ILOG CPLEX CP Optimizer (CP Optimizer). En CP Optimizer los operadores p fueron modelados como un recurso no renovable acumulativo de capacidad. Además, el CP Optimizer se configuró para aprovechar la propagación de restricciones sin superposición (*NoOverlap*) y la función acumulativa (*CumulFunction*). La estrategia de búsqueda utilizada fue búsqueda en profundidad con reinicios (configuración por defecto).

Para aplicar nuestra técnica se debe partir de una solución optimizada, esta es encontrada mediante la ejecución de nuestro CSOP. A continuación se aplica el post-proceso y la redistribución de los buffers para aumentar su número de buffers (N_{buf}). Las incidencias (Z) para medir la robustez se modelan como un retraso en una tarea al azar θ_{ij} del schedule. Para cada instancia se generan 100 incidencias con un retraso (d) que sigue una distribución uniforme entre 1 % y 10 % del tiempo máximo de procesado p_i.

Las Tablas 3.4(a), 3.4(b) y 3.4(c) muestran el comportamiento de ambas técnicas para absorber incidencias. Para cada técnica se especifica el makespan C_{max}, el número de buffers generados (N_{buf}) y la robustez (R).

Como se puede observar en las tablas, la media de buffers obtenidos por CSOP+PP fue mayor que la obtenida por CP Optimizer. Por lo tanto, cuanto mayor sea el número de buffers, se podrá considerar que la solución será más robusta, ya que tendrá más posibilidades de poder absorber una incidencia. Sin embargo, el makespan siempre es mayor para las soluciones obtenidas por CSOP+PP. En consecuencia, tras generar las incidencias aleatorias sobre las soluciones, se puede observar que CSOP+PP es capaz de absorber un mayor número de incidencias.

En la figura 3.3 se representa el tiempo computacional necesario para ejecutar cada uno de los 3 pasos. El eje y representa el tiempo en segundos y el eje x el conjunto de instancias. En la gráfica se puede observar que el tiempo para calcular el paso 1 depende de la duración de las tareas y el número de tareas por trabajo, mientras que el tiempo empleado para ejecutar el paso 2 y 3 es solo dependiente del número de tareas, ya que permanece estable para cada uno de los gráficos.

Tabla 3.4: Makespan, Número de buffers y Robustez

(a) Máximo retraso 1 unidad de tiempo

Instancias	CP Optimizer			CSOP+PP		
	C_{max}	N_{buf}	R (%)	C_{max}	N_{buf}	R (%)
3_5_10	42.70	2.00	12.50	55.50	4.40	29.20
3_7_10	57.90	1.10	6.80	71.20	7.80	38.60
3_10_10	65.20	1.20	4.60	87.00	10.90	32.60
5_5_10	39.40	0.20	1.20	51.20	4.90	32.90
5_7_10	53.90	0.70	3.10	71.50	7.50	35.90
5_10_10	60.60	0.60	1.70	78.60	8.10	25.50
7_5_10	31.00	0.90	5.40	42.60	5.20	31.10
7_7_10	44.70	0.50	2.30	61.44	5.60	29.40
7_10_10	70.40	0.50	1.50	99.00	8.20	26.90

(b) Máximo retraso 5 unidades de tiempo

Instancias	CP Optimizer			CSOP+PP		
	C_{max}	N_{buf}	R (%)	C_{max}	N_{buf}	R (%)
3_5_50	201.10	2.20	12.90	232.50	6.60	39.80
3_7_50	267.20	2.20	7.20	317.30	8.80	34.00
3_10_50	353.30	3.50	8.60	450.00	12.60	35.30
5_5_50	184.30	1.20	2.70	252.20	5.70	27.70
5_7_50	253.50	1.10	1.60	340.20	8.30	31.80
5_10_50	363.00	0.90	0.30	467.10	11.60	31.70
7_5_50	177.10	1.10	2.50	237.80	5.80	27.90
7_7_50	252.00	0.40	0.40	380.20	6.20	29.10
7_10_50	363.90	1.00	1.00	512.60	10.40	28.50

(c) Máximo retraso 10 unidades de tiempo

Instancias	CP Optimizer			CSOP+PP		
	C_{max}	N_{buf}	R (%)	C_{max}	N_{buf}	R (%)
3_5_100	389.00	2.20	12.70	459.00	7.40	41.00
3_7_100	561.40	4.40	16.70	680.90	9.70	38.50
3_10_100	768.70	2.90	4.40	948.90	12.70	34.60
5_5_100	349.00	1.10	2.70	438.90	6.40	34.20
5_7_100	495.00	0.80	0.40	643.90	8.30	33.90
5_10_100	698.20	0.70	1.00	932.10	12.40	35.00
7_5_100	391.40	1.00	2.30	500.80	6.30	37.10
7_7_100	516.00	0.40	2.00	695.80	6.70	27.80
7_10_100	744.80	0.70	0.50	1043.20	10.20	30.00

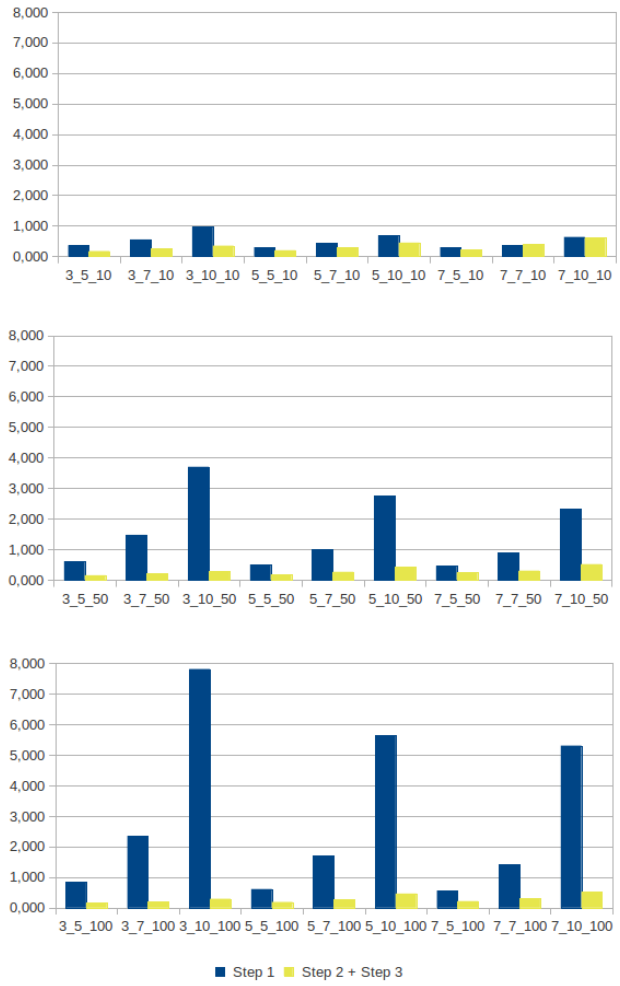


Figura 3.3: Tiempo computacional para calcular paso 1 y paso 2 + paso 3

La tabla 3.5 presenta el comportamiento de los schedules cuando el retraso de las incidencias aumenta. Cuando d se incrementa, el porcentaje de incidencias absorbidas se reduce, encontrándose el caso donde el CP Optimizer obtuvo una robustez de $0\,\%$ para un gran número de instancias cuando $p_i = 100$. Para la instancia $<$ 7_10_100 $>$ CP Optimizer no fue capaz de absorber ninguna incidencia, mientras que CSOP+PP obtuvo una robustez de $6{,}9\,\%$ para las incidencias más grandes.

Tabla 3.5: Porcentaje de robustez para incidencias grandes ($p_i = [1, 100]$)

Instancias	CP Optimizer			CSOP+PP		
	d [1,20]	**d [1,50]**	**d [1,100]**	**d [1,20]**	**d [1,50]**	**d [1,100]**
3_5_100	9.90	6.10	3.10	35.20	20.80	11.90
3_7_100	8.90	6.40	3.30	34.90	22.40	12.00
3_10_100	3.80	1.90	1.30	29.10	17.10	9.50
5_5_100	1.90	1.40	0.50	26.90	12.00	8.30
5_7_100	0.20	0.00	0.00	28.60	14.80	8.50
5_10_100	1.00	0.50	0.30	26.70	16.80	10.80
7_5_100	1.90	0.50	0.40	25.00	11.30	6.70
7_7_100	0.80	0.10	0.10	20.10	10.60	5.60
7_10_100	0.00	0.00	0.00	23.50	13.80	6.90

3.6 Conclusiones

El problema Job-shop scheduling es una representación de numerosos problemas de scheduling. La extensión con operadores aporta más complejidad al problema pero al mismo tiempo se puede acercar más a la realidad. La mayoría de las técnicas que resuelven problemas de scheduling buscan la optimalidad del problema minimizando el makespan, el tardiness, el flow-time, etc. Pero en ocasiones puede interesar tener en cuenta otros factores como puede ser la robustez. Una solución robusta puede permanecer válida frente a la aparición de incidencias. Es por ello, que las soluciones obtenidas por nuestra técnica tienen en cuenta makespan y robustez.

En este capítulo proponemos una técnica que se compone de 3 pasos. En el primer paso se resuelve el problema JSP, es decir, sin tener en cuenta los operadores. Este paso puede ser realizado por cualquier resolutor del problema Job-shop scheduling. En nuestro caso el primer paso se ha resuelto mediante el modelado del problema como un CSOP, mediante el cual obtenemos una solución válida y optimizada del problema. En el segundo paso esta solución es utilizada para introducir los operadores y para obtener una solución para problema $JSO(n, p)$. Para ello se busca una solución factible minimizando el makespan y maximizando el número de buffers. En este paso se puede controlar el peso de la optimalidad o de la robustez mediante la función de optimización según los intereses del cliente en nuestro caso se maximiza la función $\frac{N_{buf}}{C_{max}}$. Esta función puede ser ajustada, por ejemplo, incluyendo una constante para dar más peso al makespan o al número de buffers generados. Por lo tanto, si los intereses cambiaran, no sería necesario recalcular el paso 1, que es el más costoso. Por último, en el tercer paso se incrementa la robustez distribuyendo los buffers de forma más homogénea a lo largo del schedule.

Finalmente podemos concluir que, las soluciones obtenidas han sido comparadas a las obtenidas por el resolutor comercial CP Optimizer y se puede observar que mediante la pérdida de un poco de optimalidad se obtienen soluciones más robustas capaces de absorber incidencias. El coste de realizar el paso 2 y 3 es muy bajo respecto al tiempo empleado

para ejecutar el paso 1. Es por ello que el cálculo repetitivo de 2 y 3 no supone un gran aumento computacional. En cambio mediante su ejecución podemos obtener diversos tipos de soluciones cambiando el peso de las variables, y de este modo poder elegir la que más se ajuste a las necesidades.

Capítulo 4

Eficiencia Energética en Job Shop Scheduling

4.1 Introducción

En los problemas de scheduling se suelen considerar indicadores de optimización como el tiempo de procesado, la calidad y el coste. Sin embargo, cada vez más el consumo de energía se está convirtiendo en un indicador decisivo. Este cambio de enfoque es debido, por una parte, a la necesidad de frenar los problemas ambientales producidos por el cambio climático y el calentamiento global, y por otra el aumento de la demanda y los precios de los combustibles y la reducción de las reservas de materias primas energéticas. Todas estas causas se han traducido en un mayor esfuerzo por minimizar el consumo de energía (Duflou y col. 2012). Debido al interés que despierta el ahorro de energía en los problemas de scheduling, se han realizado varios trabajos sobre esta temática (Bruzzone y col. 2012, Escamilla, Salido y col. 2014, Cheng y col. 2013).

Esta capítulo está organizado de la siguiente forma, en la sección 4.2 se muestran diversos trabajos de scheduling donde se ha tenido en cuenta la eficiencia energética. En la sección 4.3 se extiende el clásico problema JSP a un problema job shop scheduling donde las máquinas pueden trabajar a distintas velocidades (JSMS). De este modo, dependiendo de la velocidad a la que la máquina ejecuta cada tarea, el consumo de energía y el tiempo de procesado varían. En la sección 4.4 se desarrolla un algoritmo Genético y un algoritmo Memético para resolver el problema JSMS. Dichos algoritmos optimizan los objetivos de makespan y consumo de energía. En la sección 4.5 los resultados de los dos algoritmos desarrollados son comparados con los obtenidos por la herramienta IBM ILOG CPLEX CP Optimizer. Por último, se presentan las conclusiones a este capítulo.

4.2 Modelos Energéticos en Scheduling

Recientemente, se ha despertado un creciente interés en el desarrollo de ahorro de energía debido a una serie de graves impactos ambientales y el aumento de los costes de energía. La investigación que busca minimizar el consumo de energía de los sistemas de fabricación se ha centrado en tres puntos de vista: a nivel de máquina, a nivel de producto y a nivel de sistema de fabricación. Desde la perspectiva a nivel de máquina, es un objetivo importante para las empresas el desarrollo y diseño de máquinas más eficientes energéticamente y equipos que reduzcan las fuentes de energía y las demandas de energía de los distintos componentes de las máquinas (Li, Zein y col. 2011, Neugebauer y col. 2011). Los estudios citados han sido desarrollados en la industria de la extracción del metal y desafortunadamente muestran que la proporción de la demanda de energía para la extracción de metal en comparación con la cuota de energía necesaria para apoyar las diversas funciones de los sistemas de fabricación es muy pequeña (menos del 30 %) del total del consumo de energía (Dahmus y Gutowski 2004). Desde la perspectiva a nivel de producto, el modelado se basa en el diseño de productos donde la reducción de energía es beneficiosa para apoyar las mejoras de diseño del producto y las decisiones operativas (Weinert, Chiotellis y Seliger 2011, Seow y Rahimifard 2011). Estas técnicas requieren un fuerte programa de simulación para facilitar el análisis y la evaluación de la energía del producto. Los resultados no pueden aplicarse fácilmente en la mayoría de las empresas, especialmente en las empresas pequeñas y medianas, debido a que requiere enormes inversiones financieras. Por último, desde la perspectiva de fabricación a nivel de sistema, gracias a los modelos de decisión que apoyan el ahorro de energía, es factible lograr una reducción significativa en el consumo de energía. En la literatura especializada sobre scheduling los objetivos de producción para modelos de decisión de producción como el coste, el tiempo y la calidad se han discutido ampliamente. Sin embargo, la disminución del consumo de energía en los sistemas de fabricación no ha sido un objetivo demasiado considerado.

En los problemas de scheduling de transporte de productos con coste mínimo, el consumo de energía también tiene una gran importancia, puesto que, está relacionado directamente con el coste y la eficiencia. En el caso del sector del transporte, el 91 % de los vehículos funcionan con productos derivados del petróleo (Energy 2015) aunque cabe destacar que este porcentaje se ha reducido en un 5 % en los últimos años por el crecimiento de otro tipos de energía como el gas natural, la electricidad o energías renovables. Aun así, la dependencia de los productos petrolíferos es muy grande, por lo tanto, el aumento en el coste del petróleo se ha visto directamente reflejado en el aumento en el coste del sector del transporte. Es por ello que han surgido varias medidas para reducir el consumo de los vehículos (Rodrigue, Comtois y Slack 2013). Como por ejemplo, en la creación de motores y su mantenimiento para que sean más eficientes, la mejora de infraestructuras y su mantenimiento y la inclusión de la energía consumida a la hora de planificar y gestionar rutas. Es este último punto es donde se pueden aplicar modelos de decisión para poder obtener soluciones que minimicen el coste teniendo en cuenta el consumo de energía.

Otros ejemplos de problemas de scheduling donde tiene una gran importancia el consumo de energía son los problemas de asignación de tareas a procesadores, ya que el consumo de energía es uno de los factores críticos en el diseño de sistemas que funcionan con baterías (Quan y Hu 2001). En muchas ocasiones en dichos problemas se trabaja con procesadores de voltaje variable con lo que si se utiliza correctamente, se puede reducir drásticamente el consumo de energía (Seo y col. 2008). Los recientes avances en circuitos de alimentación han permitido a los sistemas la posibilidad de operar dinámicamente bajo diferentes tensiones de alimentación. De tal forma que, la velocidad del sistema puede ser controlado de forma dinámica.

La mayoría de investigaciones sobre problemas de scheduling no consideran estrategias de ahorro de energía. A continuación, se presentan varios trabajos que proporcionan un punto de partida para explotar la planificación de recursos teniendo en cuenta la eficiencia energética. Cabe destacar la investigación de Mouzon, Yildirim y Twomey 2007 donde se estudia el problema de scheduling para una máquina CNC con el objetivo de reducir el consumo de energía y el tiempo total de finalización. Para ello han desarrollado varios algoritmos y modelos de programación matemática multi-objetivo. En sus estudios determinan que se puede reducir el consumo de energía hasta un 80 % desconectando las máquinas, en vez de mantenerlas en “Stand by”, cuando estas no forman parte de un cuello de botella. En un trabajo posterior, Mouzon e Yildirim (Mouzon y Yildirim 2008) propusieron un algoritmo GRASP para resolver un problema de scheduling multi-objetivo que minimiza el total de energía usada y la tardanza total en una máquina. Además de este son muchos los estudios que han tratado este tema, como en Fang y col. 2011 donde se presenta un modelo de programación lineal entera mixta para planificar un problema flow shop que combina consumo de energía total y el uso de carbón con el makespan. También, cabe destacar el trabajo de Yan y Li 2013 en el que se realiza un estudio de optimización multi-objetivo basado en el análisis relacional gris ponderado y la metodología de superficies de respuesta. Y por último, el trabajo de Bruzzone y col. 2012 en el que se presenta un algoritmo de scheduling basado en la programación entera mixta que tiene en cuenta el ahorro de energía para el problema fexible flow shop, teniendo en cuenta que se requiere mantener la asignación y secuencia de trabajos que existía originalmente.

4.3 Modelo de Job Shop Scheduling con Consumo de Energía (JSMS)

La forma más natural de resolver el problema tradicional de Job shop scheduling es representar el problema como términos de variables (trabajos, tareas y máquinas) y las correspondiente restricciones para que sean resueltas mediante una técnica de búsqueda (Blazewicz y col. 1986)(Garrido y col. 2000)(Huang y Liao 2008). Los objetivos tradicionales se centran en obtener soluciones donde se minimice una función objetivo como puede ser el makespan, el tardiness, el lateness máximo, el flow-time total, etc. Este problema

es NP-hard, por lo tanto, las soluciones óptimas solo pueden ser encontradas mediante los métodos tradicionales cuando las instancias a resolver sean pequeñas y por lo tanto el espacio de búsqueda es más reducido. Es por ello que, con instancias de problemas más grandes, es necesario el uso de métodos heurísticos. Además, con la intención de simular el comportamiento de algunos problemas reales, se ha extendido el problema JSP a un problema job shop scheduling donde las máquinas pueden trabajar a distintas velocidades (JSMS). Así que, si la velocidad a la que trabaja la máquina para realizar una tarea es mayor, el consumo de energía también será mayor y por lo tanto el tiempo de procesado se reducirá. En cambio, si la velocidad es baja, el consumo de energía será más reducido, pero el tiempo de procesado aumentará. Formalmente el problema JSMS puede ser definido como:

- Un conjunto de n trabajos $\{J_1, \ldots, J_n\}$.
- Un conjunto de m recursos o máquinas $\{R_1, \ldots, R_m\}$.
- Cada trabajo J_i está formado por una secuencia de l tareas $(\theta_{i1}, \ldots, \theta_{il})$.
- Cada tarea θ_{il} tiene que ser ejecutada por una máquina $R_{\theta_{il}}$ y se le tiene que asignar un tiempo de comienzo $st_{\theta_{il}}$.
- A cada tarea θ_{il} se le asignará un tiempo de procesado ($pt_{\theta_{il1}}$, $pt_{\theta_{il2}}, \ldots, pt_{\theta_{ilp}}$) y su correspondiente consumo de energía $e_{\theta_{il1}}$, $e_{\theta_{il2}}, \ldots, e_{\theta_{ilp}}$ dependiendo de la velocidad a la que sea ejecutada por la máquina $R_{\theta_{il}}$.

Un schedule factible es cuando todas las tareas tienen asignado un tiempo de comienzo y se satisfacen todas las restricciones de un JSP clásico:

- Las tareas de cada trabajo deben estar secuencialmente ejecutadas.
- Cada máquina puede ejecutar una sola tarea al mismo tiempo.
- No se permiten preferencias entre las tareas.

4.3.1 Generación de banco de pruebas (Benchmarks) en JSMS

Para la generación de los benchmarks JSMS se han extendido 4 benchmarks existentes del problema JSP (Agnetis y col. 2011, Lawrence 1984, Taillard 1993 y Watson y col. 1999). La generación del nuevo banco de pruebas ha sido debido a la ausencia de instancias que reflejaran las características citadas anteriormente.

Todas las instancias se caracterizan por el número de trabajos(j), el número de máquinas (m), el número máximo de tareas por trabajo (v_{max}) y el tiempo de procesado (p).

En las instancias de Agnetis, j se fija en 3 y cada tipo de instancia se representa como $m\_v_{max}\_p$. Los autores consideran 2 tipos de instancias: pequeñas y grandes:

- j = 3; m = 3, 5, 7; v_{max} = 5, 7, 10; p = [1, 10], [1, 50], [1, 100]
- j = 3; m = 3; v_{max} = 20, 25, 30; p = [1, 50], [1, 100], [1, 200]

Para las instancias de Lawrence las características son las mismas que en el caso anterior pero en este tipo de instancias el tiempo de procesado será siempre fijo y el número máximo de tareas por trabajo es igual al número de máquinas:

- j = 10, 15, 20, 30; m = 5, 10, 15; v_{max} = 5, 10, 15; p = [1, 100]

Las instancias de Taillard siguen las mismas características que las de Lawrence, pero en este caso, la variable que cambia es el número de trabajos (j). Para nuestra evaluación solo hemos hecho uso de las instancias más grandes que son las de 50 y 100 trabajos y muestran las siguientes características:

- j = 50, 100; m = 20; v_{max} = 20; p = [1, 100]

Las instancias Watson siguen las mismas características que las instancias Taillard pero en este caso llegando hasta 200 trabajos:

- j = 50, 100, 200; m = 20; v_{max} = 20; p = [1, 100]

Cada tipo de benchmark dispone de 10 instancias, excepto para las instancias Lawrence que tiene 5 instancias por cada tipo, por lo tanto los resultados mostrados son la media de los resultados obtenidos para cada instancia.

En los nuevos benchmarks, cada máquina puede ejecutarse en tres niveles de velocidad. Cada tarea está asociada a una máquina, por lo tanto, dependiendo de la velocidad de la máquina, el tiempo de procesado y la energía consumida variarán. Para la primera velocidad (la más lenta) se ha conservado el tiempo de procesado original (pt_1). El resto de tiempos de procesado se han calculado mediante la aplicación de las fórmulas 4.1 y 4.2, donde $maxdur$ es la duración máxima de las tareas para la correspondiente instancia, y la función $Rand$ genera un valor aleatorio entre las 2 expresiones.

$$pt_2 = Max(maxdur * 0{,}1 + pt_1, Rand(1{,}25 * pt_1, 2{,}25 * pt_1)) \tag{4.1}$$

$$pt_3 = Max(maxdur * 0{,}1 + pt_2, Rand(1{,}25 * pt_2, 2{,}25 * pt_2)) \tag{4.2}$$

Para el cálculo del consumo de energético acorde a las tres velocidades se han usado las siguientes expresiones 4.3, 4.4 y 4.5.

$$e_1 = Rand(pt_1, 3 * pt_1)) \tag{4.3}$$

$$e_2 = Max(1, Min(e_1 - maxdur * 0{,}1, Rand(0{,}25 * e_1, 0{,}75 * e_1)) \tag{4.4}$$

$$e_3 = Max(1, Min(e_2 - maxdur * 0{,}1, Rand(0{,}25 * e_2, 0{,}75 * e_2)) \tag{4.5}$$

De esta forma, cada tarea está representada por 3 tuplas ($< id, e, pt >$) donde en cada una de ellas se representa el identificador de la tarea (id_k), la energía usada (e_k) y el tiempo de procesado (pt_k).

Un ejemplo de representación de las tareas de un problema con 3 trabajos y 5 tareas por trabajo sería el siguiente:

$< id, e_3, pt_3 >, < id, e_2, pt_2 >, < id, e_1, pt_1 >$
$< 1, 14, 14 >, < 1, 16, 10 >, < 1, 19, 7 >,$
$< 2, 4, 6 >, < 2, 5, 5 >, < 2, 6, 4 >,$
$< 3, 13, 15 >, < 3, 14, 11 >, < 3, 15, 10 >,$
$< 4, 3, 5 >, < 4, 4, 4 >, < 4, 5, 3 >,$
$< 5, 1, 4 >, < 5, 2, 3 >, < 5, 3, 2 >,$
$< 6, 16, 14 >, < 6, 17, 12 >, < 6, 18, 8 >,$
$< 7, 5, 9 >, < 7, 10, 7 >, < 7, 12, 6 >,$
$< 8, 2, 4 >, < 8, 3, 3 >, < 8, 4, 2 >,$
$< 9, 1, 3 >, < 9, 1, 2 >, < 9, 2, 1 >,$
$< 10, 1, 3 >, < 10, 1, 2 >, < 10, 2, 1 >,$
$< 11, 8, 9 >, < 11, 12, 8 >, < 11, 17, 6 >,$
$< 12, 4, 10 >, < 12, 7, 7 >, < 12, 13, 5 >,$
$< 13, 1, 4 >, < 13, 1, 3 >, < 13, 2, 2 >,$
$< 14, 3, 10 >, < 14, 5, 9 >, < 14, 9, 7 >,$
$< 15, 3, 6 >, < 15, 5, 4 >, < 15, 6, 3 >,$

Las instancias generadas para el problema JSMS están disponibles en la página web del grupo de investigación GPS[1].

[1] http://gps.webs.upv.es/jobshop/

4.3.2 IBM ILOG CPLEX CP Optimizer

Una vez la instancia ha sido correctamente generada, puede ser resuelta. En este caso se crea un modelo con la herramienta CP Optimizer, dicho modelo también está disponible en la página web de grupo.

El objetivo es minimizar la función 4.6, para optimizar el makespan y el consumo de energía. El peso de cada una de las variables se puede cambiar dependiendo del valor de $\lambda \in [0, 1]$.

$$F(i) = \lambda * NormMakespan(i) + (1 - \lambda) * NormEnergy(i) \tag{4.6}$$

$$NormMakespan(i) = \frac{Makespan(i)}{MaxMakespan} \tag{4.7}$$

$$NormEnergy(i) = \frac{SumEnergy(i)}{MaxEnergy} \tag{4.8}$$

Como el valor del makespan y el valor de la energía usada no son proporcionales es necesario normalizar ambos (4.7 y 4.8). Es por ello que en la función objetivo aparecen los términos NormMakespan y NormEnergy. El valor NormEnergy se calcula dividiendo la suma de las energías usadas en la ejecución de todas las tareas, por MaxEnergy. MaxEnergy es la suma de la energía necesaria para ejecutar todas las tareas a máxima velocidad. El valor de NormMakespan se calcula dividiendo el makespan por MaxMakespan. MaxMakespan es el makespan máximo alcanzado en una ejecución del algoritmo genético cuando λ es igual a 0. Estos valores están también disponibles en la página web del grupo de investigación GPS.

4.4 Diseño de Algoritmo Genético para JSMS

En esta sección se propone un algoritmo genético (GA) para resolver el problema de job shop scheduling con máquinas a distintas velocidades (JSMS). Los algoritmos genéticos (GAs) son métodos adaptativos, los cuales pueden ser usados para resolver problemas de optimización (Beasley, Martin y Bull 1993). Los GAs están basados en el proceso genético de los organismos biológicos. Durante muchas generaciones, las poblaciones naturales evolucionan de acuerdo con el principio de la selección natural, es decir, la supervivencia del más apto. En cada generación, cada nuevo individuo (cromosoma) puede corresponder a una solución, es decir, una solución para una instancia dada del JSMS. Antes de que un GA se pueda ejecutar se debe realizar una codificación adecuada (o representación) del problema. La esencia de un GA es codificar un conjunto de parámetros (conocidos

como genes) y unirlos con el fin de formar una cadena de valores (cromosoma). También se requiere una función de "fitness", la cual asigna un valor de calidad para cada solución codificada. El fitness de un individuo depende de su cromosoma y es evaluado por la función de fitness. Durante la ejecución, los padres deben ser seleccionados para la reproducción y recombinados para generar la descendencia. Los padres son seleccionados al azar del total de la población, utilizando un esquema que favorece a los individuos más aptos. Después de haber seleccionado dos padres, sus cromosomas se combinan, por lo general mediante el uso de mecanismos de cruce y mutación, para generar una mejor descendencia, lo que significa mejores soluciones. El proceso se repite hasta que un criterio de parada se satisface.

4.4.1 Codificación y Decodificación del Cromosoma

En un algoritmo genético, un cromosoma puede representar una solución en el espacio de búsqueda. El primer paso para construir el GA es definir una representación apropiada del cromosoma. Una buena representación es crucial porque afecta significativamente a todos los pasos siguientes del GA. Se han creado muchas representaciones del problema JSP y en este caso hemos elegido como referencia una de las más eficiente y usadas (Varela, Serrano y Sierra 2005).

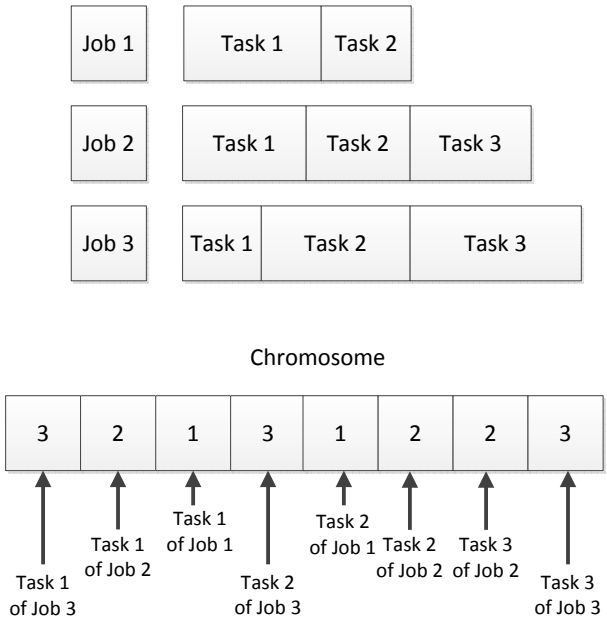


Figura 4.1: Codificación de un cromosoma para el problema JSP

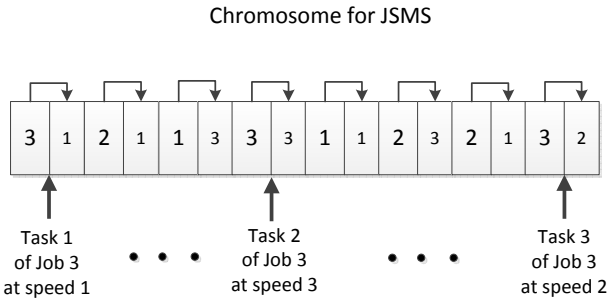


Figura 4.2: Codificación de un cromosoma para el problema JSMS

Un cromosoma es una permutación de un conjunto de tareas, las cuales representan el orden de un schedule. Cada una de estas tareas está representada por su número de trabajo. La figura 4.1 muestra el ejemplo de un problema job shop con 3 trabajos, donde el trabajo 1 tiene 2 tareas y los trabajos 2 y 3 tienen 3 tareas. Cada número del cromosoma (3,2,1,3,1,2,2,3) representa el trabajo de la tarea. El primer número 3 representa la primera tarea del trabajo 3.

En el problema JSMS la velocidad de la máquina para cada tarea tiene que estar representada, por lo tanto, un nuevo valor debe añadirse en el cromosoma para cada tarea con el fin de representar la velocidad de la máquina. Siendo un cromosoma válido el que tiene una longitud de $2n$, donde n es el número total de tareas. La figura 4.2 muestra la codificación de un cromosoma para el problema JSMS. Dicha codificación modifica la antigua codificación al añadir la velocidad a la que se procesa cada tarea. Por lo tanto, los dos primeros dígitos 3 y 1 indican que la primera tarea del trabajo 3 se procesa a velocidad 1. De nuevo, una asignación aleatoria de los valores de los trabajos y la velocidad genera una solución válida. Cuando la representación del cromosoma se decodifica, cada tarea se inicia tan pronto como sea posible siguiendo las restricciones de precedencia y de máquina. Con la representación de velocidad de la máquina se puede calcular el tiempo de procesamiento de cada tarea y el consumo de energía de la máquina. A partir de esto se pueden calcular el makespan y el consumo de energía total.

4.4.2 Población inicial

Cuando se genera la población inicial es muy importante evitar que se creen soluciones no factibles y conseguir que la población presente pluralidad. Siguiendo la codificación citada anteriormente es fácil obtener soluciones factibles, ya que, solo hay que tener en cuenta que el número de veces que aparezca un trabajo en el cromosoma tiene que ser igual al número de tareas que tiene dicho trabajo. Para generar pluralidad se suele realizar un mezclado aleatorio (randomShuffle) de todos los identificadores de trabajo, aunque

este método de crear la población inicial puede ser mejorado para obtener mejores soluciones iniciales. Con este propósito se utilizan reglas de secuenciación mediante las cuales se crea un individuo de la población. Se han generado varias reglas que se presentan a continuación, pero además, se conserva la opción de generar individuos de forma aleatoria para mantener la pluralidad de la población.

Algoritmo 6: Inicializar-Población(Input (Gensize, Speed), Output Population)

```
for (i=0;i<Populationsize;i++) do
   VecTasks= newVector[1,Gensize];
   DispatchingRule= Random(0,9);
   switch (DispatchingRule)
   case 4:
      VecTasks = SPT();
      break;
   case 5:
      VecTasks = LPT();
      break;
   case 6:
      VecTasks = JML();
      break;
   case 7:
      VecTasks = JMT();
      break;
   case 8:
      VecTasks = MML();
      break;
   case 9:
      VecTasks = MMT();
      break;
   default:
      VecTasks = randomShuffle();
   end switch
   Vectasks.Speed= Speed;
   Population[i]= Vectasks;
end for
```

Reglas de secuenciación:

- SPT (Tiempo de Procesamiento más Corto): Se da prioridad de selección a las tareas donde el tiempo de procesado es menor.

- LPT (Tiempo de Procesamiento más Largo): Se da prioridad de selección a las tareas donde el tiempo de procesado es mayor.
- JML (Trabajo con más carga): Se da prioridad de selección a las tareas del trabajo que le quedan más unidades de tiempo por ejecutar.
- JMT (Trabajo con más tareas): Se da prioridad de selección a las tareas del trabajo que le quedan más tareas por ejecutar.
- MML (Máquina con más carga): Se da prioridad de selección a las tareas de la máquina que le quedan más unidades de tiempo por ejecutar.
- MMT (Máquina con más tareas): Se da prioridad de selección a las tareas de la máquina que le quedan más tareas por ejecutar.

Cada regla de secuenciación tiene una posibilidad de ser seleccionada del 10 % y el mezclado aleatorio una probabilidad del 40 % (algoritmo 6). La velocidad de las máquinas para cada tarea en las primeras versiones se generaba aleatoriamente entre 1 y 3. Pero posteriormente, tras realizar diferentes pruebas, se comprobó que era mejor relacionar la velocidad con el valor de λ, el cual controla el peso entre los objetivos makespan y consumo de energía. Evidentemente cuando el valor de λ es menor, la velocidad debe ser menor, ya que en este caso la eficiencia energética tiene más peso. Tras realizar varias versiones y pruebas hemos seleccionado la distribución mostrada por la tabla 4.1, donde para los valores de λ menores de 0.6 la velocidad es 1 y si λ es igual a 0.6 la velocidad es 2. Finalmente, para los valores de λ igual a 0.7, 0.8 y 0.9 la velocidad de las máquinas es un valor aleatorio entre 2 y 3, siendo igual a 3 cuando λ es igual a 1.

λ	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Velocidad	1	1	1	1	1	1	2	[2,3]	[2,3]	[2,3]	3

Tabla 4.1: Velocidad de las máquinas para la población inicial

4.4.3 Cruce

Para realizar el cruce entre individuos hemos utilizado la técnica “Job-based Order Crossover (JOX)” descrita en Bierwirth 1995. Dados dos padres, JOX selecciona de forma aleatoria un subconjunto de trabajos y copia sus genes al hijo (offspring) en la misma posición que están en el primer padre (algoritmo 7). El resto de genes se seleccionan del segundo padre de modo que mantiene su orden relativo. Como cada tarea está representada por el índice del trabajo al que pertenece, se mantiene siempre la restricción de precedencia. Un ejemplo de la aplicación de la técnica se puede ver en la figura 4.3, donde:

Algoritmo 7: Cruce (Input (Parent1,Parent2), Output Offspring)

```
Begin
jobselected = Random(1,njobs);
gensize= Njobs * Ntasks;
for (i=0;i<gensize;i++) do
   if (Parent1[i].jobid==jobselected) then
      Offspring[i]=Parent1[i];
   end if
end for
j= FirstPositionEmpty(Offspring);
for (i=0;i<gensize;i++) do
   if (Parent2[i].jobid!=jobselected) then
      Offspring[j]=Parent2[i];
      j= NextPositionEmpty(Offspring);
   end if
   Evaluate-Fitness(Offspring);
end for
End
```

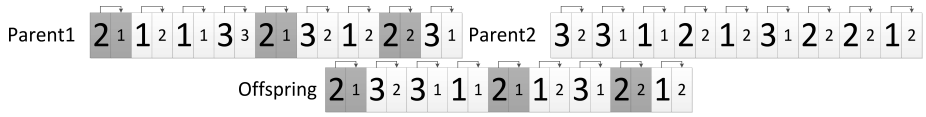


Figura 4.3: Resultado del cruce entre dos padres

El subconjunto seleccionado del primer padre solo incluye al trabajo 2 (Casillas grises del padre 1 en la figura 4.3). Tal y como se ha explicado, los genes que representan las tareas del trabajo 2 en el primer padre son copiados al nuevo hijo y los genes que representan el resto de tareas son completados con el orden del segundo padre.

La selección de los padres se hace de forma aleatoria, es decir, se mezclan todos los individuos de la población y posteriormente son seleccionados de dos en dos. Según la probabilidad de cruce cada pareja puede ser cruzada o no. Si dicha pareja se cruza se generaran 2 nuevos individuos, ya que, cada padre será elegido como primer y segundo padre. El coste del algoritmo 7 es $O(2 * gensize)$ ya que se recorren todos los genes del individuo 2 veces en la primera pasada se copiaran las tareas de los trabajos seleccionados y en la segunda el resto de tareas.

4.4.4 Mutación

Una vez los dos hijos han sido generados, estos pueden ser también mutados según la probabilidad de mutación (algoritmo 8). Para realizar la mutación, dos posiciones del cromosoma son elegidas (posición "a" y posición "b"), donde "a" tiene que ser más pequeña que "b". Los genes entre "a" y "b" son mezclados de forma aleatoria y el gen que representa la velocidad será cambiado de forma aleatoria por un valor válido de velocidad. El algoritmo 8 tiene un coste de $O(2 * gensize)$ en el peor de los casos que se produciría cuando se selecciona la posición del primer gen y del último.

Algoritmo 8: Mutación (Input Offspring, Output Offspring')

```
Begin
a,b = Random(1,gensize);
if (a>b) then
   CambiarValores(a,b);
end if
aux=a;
for (i=0;i<=b-a;i++) do
   AuxVector[i]=aux); //guardar el subvector del índice "a" a "b"
   aux++;
end for
Random_shuffle(AuxVector); //Cambiar aleatoriamente los valores del subvector
k=0;
for (j=a; j<=b;j++) do
   Offspring'[j]=Offspring[AuxVector[k]];
   Offspring'[j].speed= Random(1,NSpeed);
   k++;
end for
Evaluate-Fitness(Offspring');
Update-Population(Population,Offspring');
End
```

Finalmente, se realiza un torneo de reemplazo entre los padres y los hijos, y de este modo se actualiza la siguiente generación (Función Update-Population en el algoritmo 10)

4.4.5 Función Fitness

Cuando tratamos con un problema de optimización simple, solo existe una solución óptima o varias equivalentes. Sin embargo cuando tratamos un problema de optimización multiobjetivo no existe una solución óptima, sino que existen un conjunto de soluciones. Este conjunto de soluciones forma la frontera pareto (término acuñado por Vilfredo Pareto (Pareto y Page 1971)). Dada una variedad de objetivos y uno o varios criterios de evaluación, el concepto se utiliza a fin de analizar las posibles opciones óptimas. Dado un grupo de soluciones se puede determinar el conjunto de las que son eficientes de acuerdo con Pareto, es decir, aquellas soluciones que cumplen la condición de no poder satisfacer mejor uno de los objetivos sin empeorar algún otro.

Varias técnicas han sido desarrolladas para resolver los problemas de optimización multiobjetivo. Uno de los métodos más conocidos para resolverlos es la función "Normalized Weighted Additive Utility" (NWAUF), donde múltiples objetivos se normalizan y se añaden para formar una función de utilidad. NWAUF ha sido utilizada para muchas aplicaciones de optimización multiobjetivo debido a su sencillez y capacidad natural para identificar soluciones eficientes. Siendo f_i el i-ésimo valor de la función objetivo, la NWAUF es definida como:

$$U_j = w_1 f'_1 + w_2 f'_2 + ... + w_k f'_k \tag{4.9}$$

donde $w_1, w_2, ..., w_k$ son los pesos de cada objetivo y $f'_1, f'_2, ..., f'_k$ son valores normalizados de $f_1, f_2, ..., f_k$. Al normalizar los diferentes objetivos, todos los objetivos se evalúan en la misma escala. Los pesos muestran la importancia de cada objetivo donde, $\sum_{i=1}^{k} w_i = 1$ y $0 \leq w_i \leq 1$ para $i = 1, ..., k$. Utilizando esta función, la optimización multiobjetivo se puede resolver como un problema de optimización simple.

La definición de la función fitness es el recíproco del valor de la función objetivo. Donde el objetivo es encontrar una solución que minimice el makespan y el consumo de energía. Siguiendo las reglas NWAUF, nuestra función de fitness (4.10) es creada. Los pesos asignados a ambas variables están dados por el valor de λ.

Al igual que en la función objetivo 4.6 los valores de makespan y consumo de energía deben de ser proporcionales. Es por ello que nuevamente en la función objetivo se usan los valores "NormMakespan" y "NormEnergy" calculados en las expresiones 4.7 y 4.8. El algoritmo 9 muestra cómo se evalúa el fitness de un individuo.

$$F' = \lambda * NormMakespan + (1 - \lambda) * NormEnergy \tag{4.10}$$

Algoritmo 9: Evaluate-Fitness(Input Offspring', Output Fitness)

```
Begin
Makespan = 0;
EnergyUsed = 0;
for (i=0;i<gensize;i++) do
   EndTask= Offspring'[i].initialTime+ Offspring'[i].duration;
   if (EndTask>Makespan) then
      Makespan=EndTask;
   end if
   EnergyUsed += Offspring'[i].EnergyUsed;
end for
Fitness= λ * (Makespan/MaxMakespan) + (1 - λ) * (EnergyUsed/MaxEnergy);
End
```

4.4.6 Búsqueda Local

Mediante la aplicación de algoritmos genéticos se pueden obtener buenos resultados. Sin embargo, las mejoras significativas se obtienen mediante la hibridación con otros métodos. Por ejemplo, la hibridación de un GA y un algoritmo de búsqueda local puede producir mejores resultados, ya que los algoritmos de búsqueda local consisten en desplazarse de una solución a otra en el espacio de búsqueda mediante la aplicación de pequeños cambios locales. Un algoritmo de búsqueda local (LS) comienza a partir de una solución y luego se desplaza de forma iterativa a las soluciones vecinas. Esto es posible solo si se define una relación de vecindad en el espacio de búsqueda, de modo que para cada individuo seleccionado se calculará su vecindario aplicando la regla de vecindad, por lo que entre los individuos de la vecindad será seleccionado el mejor y posteriormente será introducido en la población.

En esta sección se propone una estructura de vecindad basada en los conceptos de la ruta crítica y el bloque crítico (Matsuo, Suh y Sullivan 1988, Van Laarhoven, Aarts y Lenstra 1992, Nowicki y Smutnicki 1996). Una ruta crítica es la secuencia de tareas que marcan el tiempo más corto en el que es posible completar el plan. La duración de la ruta crítica determina la duración del schedule entero. Cualquier retraso en un elemento de la ruta crítica afecta al tiempo de finalización. Un bloque crítico es una subsecuencia de operaciones de una ruta crítica que requiere la misma máquina.

En Mattfeld 1995 se define la estructura de vecindad N_1 para JSP. Mattfeld considera un conjunto de movimientos definidos por el cómo intercambiar los pares de operaciones que se encuentran al principio o al final de un bloque crítico.

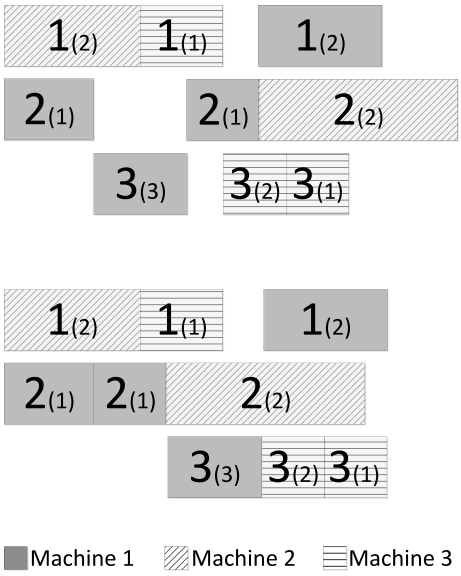


Figura 4.4: Ejemplo de dos cromosomas decodificados antes y después de aplicar la búsqueda local

Siguiendo la idea de la estructura de vecindad N_1 y los conceptos de la ruta crítica y el bloque crítico, hemos definido una relación de vecindad que tiene en cuenta la eficiencia energética (N_{EE}). Cada tarea en la ruta crítica es analizada para comprobar si las próximas tareas de la misma máquina son consecutivas y se encuentran en la ruta crítica. Si se cumplen estas condiciones se crea un nuevo vecino intercambiando ambas tareas y se aumenta la velocidad de la máquina siempre y cuando la función fitness no empeore. Por otra parte, cuando una tarea no está en una ruta crítica, si no se empeora la función fitness, su velocidad se reduce, y se crea un nuevo vecino. De este modo se consigue ahorrar energía.

Ejemplo de Búsqueda local

En la figura 4.4 se muestra una solución a un problema con 3 trabajos, 3 tareas por trabajo y 3 máquinas. Si aplicamos la regla de vecindad N_{EE}, podemos ver que tenemos dos tareas consecutivas que están en el camino crítico (primera tarea del trabajo 3 y segunda tarea del trabajo 2). Cambiando estas dos tareas crearíamos un vecino y como se muestra en la figura 4.4, el nuevo individuo mejoraría al anterior ya que se reduce el makespan sin cambiar el consumo de energía.

Algoritmo 10: Algoritmo Memético (JSMS, λ)

```
Begin
Gensize= Njobs * Ntasks;
if (λ < 0,6) then
   Initial-Population(Gensize, Speed=1, Population);
end if
if (λ = 0,6) then
   Initial-Population(Gensize, Speed=2, Population);
end if
if (λ > 0,6 λ <= 0,9) then
   Initial-Population(Gensize, Speed=Random(2,3), Population);
end if
if (λ > 0,9) then
   Initial-Population(Gensize, Speed=3, Population);
end if
Evaluate-Fitness(Population);
while (Criterio de parada no se cumpla) do
   Population= Shuffle(Population); // Mezclar población
   for (i=0;i<Populationsize;i=i+2) do
      Parent1= Population[i];
      Parent2= Population[i+1];
      Crossover(Parent1,Parent2,Offspring);
      Mutation(Offspring,Offspring');
      if (RUNTIME > 80 % TIMEOUT) then
         LocalSearch(Offspring');
      end if
      Evaluate-Fitness(Offspring');
      Update-Population(Population,Offspring');
   end for
end while
Report Best Schedule;
End
```

4.4.7 Algoritmo Memético

El algoritmo memético (GA+LS) combina el algoritmo genético (GA) con la búsqueda local (LS). La búsqueda local en ocasiones puede estancarse en mínimos locales, ya que la búsqueda solo se realiza entre vecinos próximos. En cambio el algoritmo genético da mucha más diversidad en la búsqueda.

Es por ello que la búsqueda local solamente se lleva a cabo si el tiempo de ejecución excede el 80 % del tiempo máximo de ejecución del algoritmo. Con esta condición, al comienzo trabaja únicamente el algoritmo genético y es al final del algoritmo de búsqueda cuando trabajan conjuntamente el algoritmo genético y la búsqueda local. En el algoritmo 10 se muestra de forma resumida como sería el algoritmo memético.

4.5 Evaluación de JSMS

El objetivo de este estudio experimental es analizar el comportamiento de tres métodos ya presentados en los apartados anteriores para resolver el problema JSMS: IBM ILOG CPLEX CP Optimizer tool 12.5 (CP Optimizer), nuestro algoritmo genético (GA) y el algoritmo memético (GA+LS). CP Optimizer (ILOG 2012) es un resolutor comercial que incorpora potentes técnicas de propagación de restricciones y un método autoadaptable de búsqueda local dedicada a problemas de scheduling (Laborie 2009). De CP Optimizer se espera que sea un resolutor muy eficiente para una variedad de problemas de scheduling como se señala en (IBM 2009).

En todos los casos se trata de optimizar la misma función multiobjetivo, donde se minimiza el makespan y la eficiencia energética. El peso de cada uno de los objetivos se controla con el parámetro λ. Para la comparación de todas las técnicas, estas han sido ejecutadas siempre en una máquina Intel Core2 Quad CPU Q9550, 2.83GHz y 4Gb Ram con el sistema operativo Ubuntu 12.04. Las instancias pequeñas de Agnetis han sido ejecutadas durante 5 segundos y el resto de instancias durante 100 segundos.

Para las instancias Agnetis, el número máximo de operaciones es de 90 en las instancias 3_30_p. Para las instancias de Taillard, el número máximo de operaciones es de 2000 cuando las instancias tienen 100 trabajos y 20 tareas por trabajo. Para las instancias Watson, el número de operaciones va desde 1000 (j=50 y v_{max}=20) a 4000 (j=200 y v_{max}=20). Por lo tanto, mediante esta evaluación se muestra el comportamiento de los distintos algoritmos para resolver problemas cada vez más complejos en un mismo tiempo límite. Para todas las instancias el tiempo empleado ha sido el mismo, excepto las instancias pequeñas de Agnetis, donde cada algoritmo es ejecutado durante 5 segundos, para comprobar cómo se compartan dichos algoritmos con instancias más pequeñas.

Los resultados de las evaluaciones se muestran mediante las tablas (4.2, 4.3, 4.4, 4.5, 4.6, 4.8, 4.9 y 4.10,) y las gráficas (4.5 y 4.6). En las tablas se muestran los valores medios

Tabla 4.2: Resultados de las instancias Agnetis <3_5_10>.

	<3_5_10>								
	CP Optimizer			GA			GA+LS		
λ	Mk	En	F	Mk	En	F	Mk	En	F
0	71.4	84.4	**0.553762**	65.8	84.4	**0.553762**	65.8	84.4	**0.553762**
0.1	65.2	84.5	**0.556581**	65.2	84.5	**0.556581**	65.2	84.5	**0.556581**
0.2	64.4	84.7	**0.558703**	64.4	84.7	**0.558703**	64.4	84.7	**0.558703**
0.3	63.2	85.2	**0.559422**	63.2	85.2	**0.559422**	63.2	85.2	**0.559422**
0.4	59.7	88.1	**0.557816**	59.7	88.1	**0.557816**	59.7	88.1	**0.557816**
0.5	53.9	94.3	**0.547187**	54	94.2	0.547239	54	94.2	0.547239
0.6	48.4	104.2	0.529935	49	103	**0.529690**	49	103	0.529761
0.7	45.3	111.9	0.500419	45.4	111.7	0.500596	44.8	113.3	**0.500084**
0.8	42.2	123.4	**0.461368**	42.2	123.5	0.461503	42.2	123.5	0.461503
0.9	41	133.2	**0.414361**	41	133.7	0.414711	41	133.5	0.414570
1	41	143.1	**0.363050**	41	152	**0.363050**	41	152	**0.363050**

Tabla 4.3: Resultados de las instancias Agnetis <7_10_100>.

	7_10_100								
	CP Optimizer			GA			GA+LS		
λ	Mk	En	F	Mk	En	F	Mk	En	F
0	1088.4	1571.4	**0.533616**	1006.3	1571.4	**0.533616**	1006.3	1571.4	**0.533616**
0.1	999.3	1572.6	**0.540931**	999.3	1572.6	**0.540931**	999.3	1572.6	**0.540931**
0.2	987.2	1576.5	**0.547509**	989.7	1575.5	0.547528	987.2	1576.5	**0.547509**
0.3	922.2	1613.3	**0.550868**	928.4	1610.2	0.551269	922.2	1613.3	**0.550868**
0.4	885.9	1649.2	0.550650	888.1	1646.6	0.550749	888	1646	**0.550556**
0.5	838.8	1716	0.545249	846.9	1705.8	0.546065	842.1	1711.3	**0.545187**
0.6	779.1	1859.3	0.535095	769.9	1877.6	0.534793	774.4	1861.1	**0.534304**
0.7	708.5	2068.4	0.511331	697.1	2118.9	0.511625	704	2085.1	**0.511140**
0.8	651.8	2346	0.475184	642.9	2421.7	0.476217	644.1	2394.3	**0.475081**
0.9	626	2560.7	**0.428228**	627.4	2561.1	0.428935	626	2560.7	**0.428228**
1	625.9	2664.1	**0.378956**	625.9	2935.1	**0.378956**	625.9	2935.1	**0.378956**

obtenidos de makespan, consumo de energía y valor F al ejecutar cada una de las técnicas evaluadas. Mediante las gráficas se representa el valor de la F con respecto al valor de λ. De este modo, se puede observar el comportamiento de cada una de las técnicas para las distintas estancias evaluadas.

Como se ilustra en las tablas, los resultados obtenidos en las tres técnicas para las instancias Agnetis son muy parecidos y muchas veces se obtiene el mismo valor de F. La única diferencia destacable se observa cuando se analizan las instancias large (3_25_100) donde el CP Optimizer obtiene unos resultados levemente mejores que el resto de las técnicas. Las diferencias entre las distintas técnicas se pueden ver más en detalle mediante los resultados mostrados en las tablas 4.2, 4.3 y 4.4, donde se destaca en negrita los mejores resultados obtenidos.

Tabla 4.4: Resultados de las instancias Agnetis <3_25_100>.

	3_25_100								
	CP Optimizer			GA			GA+LS		
λ	Mk	En	F	Mk	En	F	Mk	En	F
0	3160	3827.1	**0.533532**	2888.8	3827.1	0.533532	2884.4	3827.2	0.533546
0.1	2768.1	3827.6	**0.537461**	2786.8	3827.6	0.537847	2790.1	3827.7	0.537925
0.2	2719.3	3842.5	**0.540966**	2777	3835.9	0.542636	2767.4	3836.1	0.542231
0.3	2597.9	3904.6	**0.542188**	2679.5	3881.3	0.544943	2708.1	3866.4	0.545254
0.4	2480.7	4005.8	**0.540172**	2524.5	4007.3	0.543965	2629.9	3942.6	0.547249
0.5	2342	4181.6	**0.533724**	2356.8	4208.8	0.537069	2445.6	4131	0.540795
0.6	2147	4548.6	0.520427	2113.4	4611.1	**0.519679**	2143.9	4659.5	0.526111
0.7	1935.5	5075.6	**0.492575**	1946.9	5060.3	0.493846	1915.4	5346.8	0.501229
0.8	1806.2	5666	**0.456913**	1790.2	5791.3	0.457769	1786.2	5968.6	0.462180
0.9	1725.9	6251.3	**0.408634**	1723.5	6377.4	0.409699	1731.7	6542.6	0.413791
1	1673.4	6732.2	**0.346046**	1697	7088.5	0.350873	1688.6	7129.8	0.349065

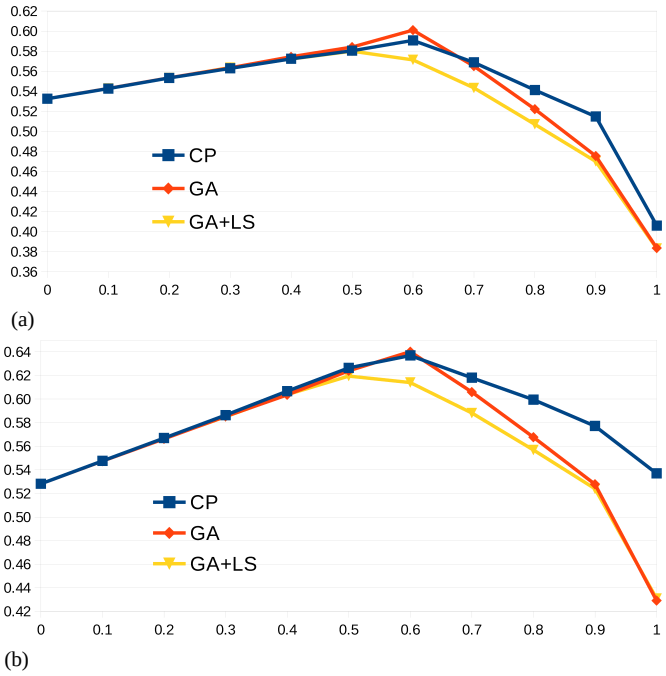


Figura 4.5: Valores de F para CP, GA y GA+LS en las instancias Taillard de 50 y 100 trabajos.

Tabla 4.5: Resultados de las instancias Taillard50.

	Taillard50								
	CP Optimizer			GA			GA+LS		
λ	Mk	En	F	Mk	En	F	Mk	En	F
0	5705.2	53157.8	**0.532581**	5584	53157.8	**0.532581**	5584.5	53157.8	**0.532581**
0.1	5386.2	53162.5	**0.542629**	5398.8	53174.5	0.542885	5409.6	53158.8	0.542873
0.2	5406.8	53190.4	**0.553343**	5418.6	53179.5	0.553532	5393.7	53231.2	0.553354
0.3	5375.6	53248.6	**0.562857**	5403.9	53185.1	0.563422	5380.4	53344.3	0.563693
0.4	5359.9	53328.9	0.572383	5409.6	53289.4	0.574484	5316.7	53641.1	**0.572247**
0.5	5305.6	53684	0.580496	5350.4	53883	0.584167	5182.2	55036.5	**0.580058**
0.6	4660.1	65552.6	0.590794	4189	76316.3	0.601083	4341.2	66252.4	**0.571432**
0.7	4228.8	73585.5	0.568867	3809.7	83820.4	0.565151	3883.6	74552.1	**0.543415**
0.8	4167.8	74679.2	0.541264	3706.5	86798.5	0.522193	3756.3	76903.6	**0.507115**
0.9	4173.6	73445.4	0.514757	3664.9	87886.7	0.475492	3703.3	78360.7	**0.470012**
1	3456.8	84280.1	0.405993	3266.3	99677.2	0.383647	3262.7	96455.3	**0.383233**

Tabla 4.6: Resultados de las instancias Taillard100.

	Taillard100								
	CP Optimizer			GA			GA+LS		
λ	Mk	En	F	Mk	En	F	Mk	En	F
0	9869.4	104713.5	**0.528207**	9806.9	104713.5	**0.528207**	9824.7	104713.5	**0.528207**
0.1	9616.3	104718.6	0.547583	9591.4	104721	**0.547418**	9613.3	104719.3	0.547573
0.2	9610.9	104747.3	0.566969	9567.9	104732.3	**0.566281**	9584.5	104753.7	0.566618
0.3	9611.9	104780.5	0.586404	9558.4	104809.1	0.585334	9536.6	104881.8	**0.585104**
0.4	9645.8	104773	0.606708	9537.5	104816.9	**0.603612**	9499.4	105276.6	0.603870
0.5	9647.2	104835.7	0.626444	9502.8	106010	0.624049	9115.9	109975.6	**0.619544**
0.6	7859.5	140364.3	0.636968	7022.5	160598.5	0.640262	7352.3	140260.6	**0.614082**
0.7	7457	149547.4	0.618119	6625.6	170392.7	0.606006	6742.3	154573.6	**0.588195**
0.8	7470.2	149575.8	0.599499	6558.6	172371.7	0.567768	6626.5	157587.6	**0.556910**
0.9	7422.1	150168.6	0.577171	6512.4	174317.4	0.527902	6565.6	159514.4	**0.524020**
1	7154.1	152604.7	0.537014	5717.4	198005.4	**0.429188**	5740.6	196581.2	0.430917

En la figura 4.5 se puede ver el comportamiento de las tres técnicas para las instancias Taillard50 (Figura 4.5a) y Taillard100 (Figura 4.5b). Aquí ya se ilustran las diferencias entre las distintas técnicas, sobre todo para los valores de λ mayores de 0.5 donde la técnica GA+LS obtiene mejores resultados. Para Taillard100 la mejoría respecto a las otras técnicas aún es más notoria, ya que, las instancias son más grandes (2000 tareas) y por lo tanto el espacio de búsqueda es mayor. Los resultados se pueden observar con más detalle en las tablas 4.5 y 4.6.

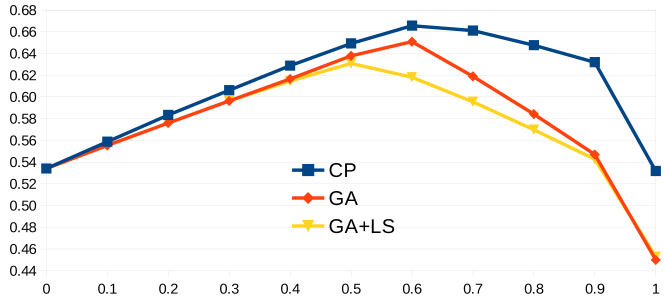
Las instancias más grandes utilizadas para esta evaluación son las instancias Watson. Al evaluarlas con las tres técnicas es donde se ven las diferencias más significativas. En los primeros resultados del CP Optimizer para las instancia Watson200, el resolutor no fue capaz de obtener soluciones para la mayoría de los valores de λ. Es por ello, que tras el análisis y la validación del modelo por parte de expertos en CP Optimizer se configuró el parámetro cp.param.Worker, que por defecto está configurado a 4 (Salido, Escamilla,

Tabla 4.7: Comparación entre los valores de la F obtenidos por el CP Optimizer para los valores 1 y 4 en el parámetro cp.param.Worker.

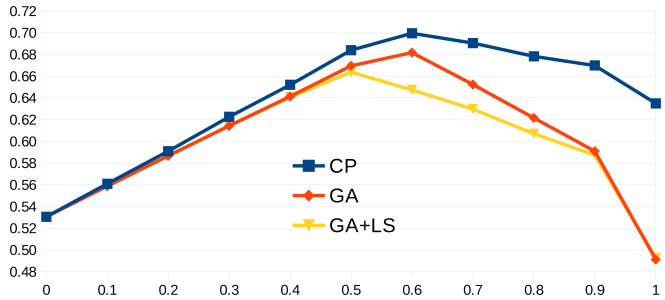
	CP Optimizer					
λ	F50_4W	F50_1W	F100_4W	F100_1W	F200_4W	F200_1W
0	0.534084	0.534084	0.530618	0.530618	0.772881	0.530985
0.1	0.558991	0.558776	0.561141	0.561048	No Sol.	0.785615
0.2	0.583292	0.583375	0.591380	0.591010	No Sol.	0.797274
0.3	0.608365	0.606205	0.621446	0.622653	No Sol.	0.808934
0.4	0.630387	0.628810	0.651177	0.652105	No Sol.	0.793322
0.5	0.653347	0.649284	0.683171	0.684038	No Sol.	0.749543
0.6	0.669361	0.665704	0.697939	0.699602	No Sol.	0.731213
0.7	0.659376	0.661105	0.692349	0.690528	No Sol.	0.725809
0.8	0.643461	0.647691	0.680517	0.678396	No Sol.	0.719746
0.9	0.631977	0.632014	0.664297	0.669925	No Sol.	0.710794
1	0.526759	0.531812	0.635000	0.635030	0.694521	0.698323

Tabla 4.8: Resultados de las instancias Watson50.

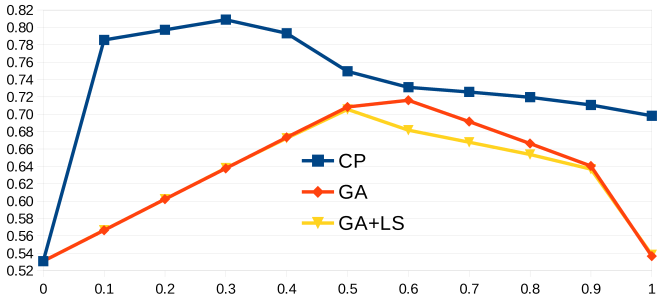
	Watson50								
	CP Optimizer			GA			GA*+LS		
λ	Mk	En	F	Mk	En	F	Mk	En	F
0	7648.9	53631.5	**0.534084**	7261.3	53631.5	**0.534084**	7267.5	53631.5	**0.534084**
0.1	7322.7	53669.5	0.558776	7025.6	53639.7	**0.555350**	7036.4	53635.2	0.555427
0.2	7258.6	53875	0.583375	6997.9	53652.3	0.576045	6979.8	53698.9	**0.576029**
0.3	7192.5	54094.1	0.606205	6966.1	53712.1	**0.596344**	6945.4	53822.3	0.596453
0.4	7172	54257.6	0.628810	6939.9	53871	0.616654	6824.5	54351.4	**0.614622**
0.5	7121	54467.3	0.649284	6885.8	54685.8	0.637878	6680.6	55472.4	**0.630886**
0.6	6552.1	62253.7	0.665704	5042.8	82747.3	0.650957	5305.1	70358	**0.618196**
0.7	6039.1	71014.7	0.661105	4880.2	85781.6	0.619024	4970.9	75679.6	**0.595589**
0.8	5947.3	71489.3	0.647691	4841.9	86865.7	0.584348	4896.8	77262.9	**0.569877**
0.9	5876.3	70699.2	0.632014	4807.8	87870.9	0.547009	4862.8	78307.4	**0.542734**
1	5008.3	81527.7	0.531812	4237.7	100226.6	**0.449998**	4265.7	98395.6	0.452947



(a)



(b)



(c)

Figura 4.6: Valores de F para CP, GA y GA+LS en las instancias Watson50, Watson100 y Watson200.

Tabla 4.9: Resultados de las instancias Watson100.

	Watson100								
	CP Optimizer			GA			GA*+LS		
λ	Mk	En	F	Mk	En	F	Mk	En	F
0	12671.2	105478.1	**0.530617**	12342.4	105478.1	**0.530617**	12339.4	105478.1	**0.530617**
0.1	12378.1	105513.8	0.561048	12058.4	105483.9	**0.558754**	12085.9	105483.1	0.558938
0.2	12297.4	105710.8	0.591010	12048.2	105485.4	0.586739	12074.3	105490.1	**0.587106**
0.3	12408	105654	0.622653	12013.2	105565.2	0.614342	11949.3	105921.3	**0.614317**
0.4	12376.4	105634.3	0.652105	11951	105889.6	0.641428	11828.6	106739.4	**0.640697**
0.5	12097.2	110089.6	0.684038	11956.5	106227.5	0.669622	11573.3	109046.2	**0.663841**
0.6	10296.9	140905.5	0.699602	8371.8	170813.3	0.681839	8761.3	145892.3	**0.647497**
0.7	9835.1	150495.9	0.690528	8336.8	172021.3	0.652456	8560.4	150040.1	**0.629834**
0.8	9785.8	150468.8	0.678396	8326.3	172246.5	0.621724	8406.5	153414.3	**0.607110**
0.9	9813.2	149937.6	0.669925	8318.4	173134.1	0.591085	8415.8	154301.4	**0.587494**
1	9434.5	152047.9	0.635030	7299	198662.3	**0.491358**	7318.7	196574.7	0.492683

Tabla 4.10: Resultados de las instancias Watson200.

	Watson200								
	CP Optimizer			GA			GA*+LS		
λ	Mk	En	F	Mk	En	F	Mk	En	F
0	22399.6	211041.5	0.530985	21999.6	211040.1	**0.530981**	21982.7	211040.1	**0.530981**
0.1	21776	307619.8	0.785615	21703.8	211040.1	**0.566642**	21725.5	211040.1	0.566732
0.2	21776	307619.8	0.797274	21705.9	211051.3	0.602344	21697.7	211073.5	**0.602325**
0.3	21776	307619.8	0.808934	21669.5	211164.1	**0.637766**	21674.5	211331.5	0.638119
0.4	22701.1	279768.9	0.793322	21672.2	211322.4	0.673529	21576.5	211657.1	**0.672474**
0.5	22428.6	231369	0.749543	21553.3	212850.7	0.708480	21123.5	217626.6	**0.705700**
0.6	18215.4	282412.4	0.731213	15094.3	343694.1	0.716274	15590.1	297314.6	**0.681767**
0.7	17439.5	300096.2	0.725809	15088.4	344133.8	0.691692	15311.9	304140.6	**0.667889**
0.8	17379	300292.6	0.719746	15066.5	344890	0.666474	15277.5	305966.2	**0.653797**
0.9	17253.8	300856.2	0.710794	15046.3	345008.8	0.640608	15191.8	309181.4	**0.636952**
1	17074.7	299562.5	0.698323	13126.6	397428	**0.536829**	13165.5	397436.9	0.538403

Giret y Barber 2015). Sin embargo, las pruebas demuestran que si este parámetro se fija a 1, en este tipo de instancias los resultados mejoran. La diferencia para las instancias Watson50 y Watson100 no es muy grande, tal y como se puede observar en la tabla comparativa 4.7. Dado que el CP Optimizer mejoraba en este tipo de instancias, las comparativas del CP Optimizer frente a nuestras técnicas para las instancias de Watson se realizaron con el parámetro $cp.param.Worker = 1$. El cambio de este parámetro también se realizó para el resto de instancias, pero los resultados no presentaron ninguna mejora.

A la vista de las figuras 4.6a y 4.6b, se puede concluir que para las instancias Watson50 y Watson100 tanto GA como GA+LS obtienen mejores resultados que CP Optimizer. Para las instancias Watson200, la diferencia entre nuestras técnicas y el CP es más grande, excepto para el valor de $\lambda = 0$, cuando solo se minimiza el objetivo de la energía. Esta mejoría es debida a que la dimensión de las instancias es mayor, y por lo tanto, el número

de combinaciones posibles que existen también es más significativo. Como se ha explicado anteriormente, el algoritmo genético, debido a su aleatoriedad en la búsqueda, es capaz de encontrar soluciones de forma más rápida. Los resultados obtenidos por las tres técnicas se pueden ver de forma más extendida en las tablas 4.8, 4.9 y 4.10.

El beneficio de aplicar la búsqueda local se puede observar tanto en las instancias Taillard como Watson, aunque esta diferencia es mayor para las instancias Watson. En todos los casos se observa que la diferencia al aplicar la LS es notoria para los valores de λ mayores de 0,5 y menores de 0,9 como se puede observar en las tablas 4.8, 4.9 y 4.10. Esto se produce porque en estos casos es donde la LS, además de reducir el makespan, reduce la energía usada mediante la reducción de las máquinas que no están en el camino crítico.

4.6 Conclusiones

Muchos problemas de la vida real pueden ser modelados como un job shop scheduling donde las máquinas pueden trabajar a distintas velocidades (JSMS). JSMS representa una extensión del clásico problema job shop scheduling, donde cada tarea tiene que ser ejecutada en una máquina y con la posibilidad de trabajar a distintas velocidades. Es esta sección se han presentado tres técnicas, para posteriormente evaluar cada una de ellas resolviendo distintas instancias de JSMS. Hemos utilizado la aplicación IBM ILOG CPLEX CP Optimizer (CP) y hemos desarrollado un algoritmo genético (GA) y un algoritmo memético (GA+LS), el cual incorpora al algoritmo genético una búsqueda local. La evaluación se ha realizado con distintas instancias de JSMS de distintos tamaños. Observando los resultados obtenidos se puede concluir que para las instancias más pequeñas, las soluciones de todas las técnicas son muy parecidas. Sin embargo, cuando las instancias son cada vez más grandes y el tiempo que se le da a cada técnica permanece constante, se puede observar que el GA y GA+LS obtienen soluciones más optimizadas. También se puede concluir que el GA+LS presenta una mejora respecto a GA, en especial en las soluciones donde el makespan tiene más peso y donde es más fácil reducir la velocidad de las máquinas para las tareas que no están en el camino crítico, consiguiendo ahorrar energía sin perder optimalidad.

Capítulo 5

Robustez y Estabilidad en Job Shop Scheduling

5.1 Introducción

La robustez es una característica común en los problemas de la vida real. La vida biológica, los sistemas funcionales, objetos físicos, etc. persisten si permanecen funcionando y mantienen sus principales características, a pesar de continuas perturbaciones, cambios, incidencias o agresiones (Szathmáry 2006). Por lo tanto, la robustez es un concepto relacionado con la persistencia del sistema, de su estructura, de su funcionalidad, etc, contra interferencias externas. Podemos decir que un sistema es robusto si persiste.

Intuitivamente, la robustez es fácil de definir, pero su formalización depende del sistema, de su funcionalidad esperada y del conjunto particular de incidencias a afrontar (Rizk, Batt, Fages y Solima 2009). El concepto de robustez se puede definir de maneras diferente dependiendo del dominio de aplicación. La mayoría de los autores prefieren utilizar el concepto de sistema robusto (Kitano 2007, Rizk, Batt, Fages y Soliman 2009).

En la sección 5.2 se definen los conceptos de robustez y estabilidad para el problema JSMS y se desarrolla un modelo dual mediante el cual se utilizan las características del problema para obtener soluciones robustas (Escamilla y Salido 2016). Dicho modelo es evaluado en la sección 5.3, donde se comparan los resultados obtenidos utilizando el modelo dual frente a los resultados sin utilizar el modelo dual. Al final del capítulo se muestran las conclusiones.

5.2 Robustez en Job Shop Scheduling con Consumo de Energía (JSMS)

Cuando tratamos con el problema JSMS, además de poder absorber las incidencias mediante gaps o buffers de tiempo, existe la posibilidad de cambiar la velocidad de las máquinas. Aumentando la velocidad se puede recuperar el tiempo perdido por las incidencias (buffer de eficiencia energética). Por lo tanto, en este caso, una incidencia puede ser absorbida por un buffer o por la aceleración de la máquina que ejecuta la tarea sobre la que recae la incidencia. En ocasiones un buffer y un buffer de eficiencia energética se pueden unir para absorber una incidencia, de modo que la definición de robustez $R_{S,I}$ de un sistema (S) ante una incidencia (I) quedaría de la siguiente forma:

- $R_{S,I} = 1$ Si solo la tarea afectada por I es modificada. Por lo tanto un "gap" o "buffer" asignado a esta tarea puede absorber la incidencia o la máquina que ejecuta dicha tarea puede acelerar su velocidad para absorber la incidencia.
- $R_{S,I} = 0$ Si más de una tarea se ven afectadas por I. Esto significa que el "buffer" asignado a esta tarea no puede absorber la incidencia y se propaga al resto del schedule.

Si un sistema no es robusto frente a una incidencia se puede considerar como estable. La estabilidad es la habilidad de una solución para recuperarse tras una incidencia con tan solo unos pocos cambios Jen 2003. Un sistema S es considerado $s - estable$ si existe otro sistema S', de forma que $||S' - S|| < s$, donde $||.||$ es cierta n-dimensionalidad definida en el espacio de soluciones para evaluar la diferencia entre S y S' Barber y Salido 2014. En nuestro caso la diferencia entre S y S', podría ser el número de tareas que tienen que cambiar para mantener la factibilidad. Por lo tanto, dada una incidencia I, 0-estable es equivalente a la robustez, ya que la incidencia no se propaga a ninguna otra tarea y es absorbida por la tarea afectada. 1-estable o 2-estable requieren cambiar el tiempo de comienzo de una o dos tareas, respectivamente.

La mayoría de los problemas de scheduling en general son dinámicos por naturaleza. Estos problemas dinámicos se enfrentan a incidencias que generan retrasos en las tareas. En algunos casos la solución original se recupera en un tiempo dado, pero en otros casos la incidencia se propaga en cascada, por lo que la solución original puede no ser válida. El uso de buffers es utilizado en los procesos industriales para compensar los cambios en el schedule. Sin embargo, la inclusión de buffers en la solución tiene ventajas y desventajas. La ventaja más importante del uso de buffers, si se hace correctamente, es que las soluciones tienden a ser más robustas y por lo tanto se consigue aumentar la eficiencia de la producción, reducir los costes generales y mantener las operaciones funcionando sin problemas. Sin embargo, cuando se hace de forma incorrecta, puede suceder lo contrario. Por ejemplo, si un schedule tiene demasiados buffers entre las tareas, el makespan aumenta,

Tabla 5.1: Datos del histórico de las incidencias

Instancias	Ntareas	Nincidencias	NTareasDinamicas
3_5_10	15	100	5
5_10_50	30	200	10
7_10_100	30	200	10
3_20_50	60	400	20
3_25_100	75	500	25
3_30_200	90	600	30
Watson50	1000	7000	400
Watson100	2000	14000	800
Watson200	4000	28000	1600

lo que puede suponer que la solución obtenida no sea factible en términos de rentabilidad. Por ello hay que mantener un equilibrio entre robustez y rentabilidad de un schedule.

5.2.1 Análisis de Incidencias

En todo schedule dinámico se deben gestionar los buffers que se pueden generar. Las tareas a las que se deben proteger con un buffer son las tareas más propensas a sufrir incidencias (tareas dinámicas). Para saber qué tareas son las más dinámicas, se debe realizar un estudio del histórico o consultar a un experto.

En este caso hemos optado por analizar un histórico. Entendemos como histórico un conjunto de datos de los cuales mediante su análisis se puede extraer información de lo ocurrido anteriormente y por lo tanto usar dicha información para futuras ocasiones. Para tal fin se ha simulado la creación de un histórico generando incidencias a las instancias de Agnetis y Watson. El número de incidencias generadas se ha estipulado en proporción al número de tareas de cada grupo de instancias, ya que cuanto mayor sea el tamaño de las instancias, el histórico debe ser mayor para obtener resultados significativos del análisis. La duración de cada incidencia es un valor aleatorio entre el 1 % y el 20 % del tiempo de procesado máximo de cada tipo de instancia. Una vez obtenido el histórico se debe definir el número de tareas dinámicas, que en este caso también se considera acorde al número de tareas de la instancia. El número de incidencias analizadas por cada grupo de instancias (Nincidencias) y el número de tareas dinámicas (NTareasDinamicas) se pueden observar en la tabla 5.1. Estos valores han sido calculados acorde al número de tareas (Ntareas).

Tras analizar las incidencias se detectan las tareas dinámicas para cada instancia. Mediante el análisis de las incidencias también se obtiene el valor máximo o medio del histórico de las incidencias, valores que luego serán usados para reforzar a dichas tareas dinámicas (tiempo de buffer histórico). El termino tiempo de buffer histórico es el tiempo de buffer necesario que debe tener una tarea para poder absorber gran parte de las incidencias estudiadas en el histórico.

5.2.2 Modelado dual del problema JSMS

Una vez determinadas las tareas dinámicas y el tiempo de buffer histórico para cada tarea dinámica Ti se genera una tarea dual (Ti'), la cual vendrá reforzada para que sea capaz de absorber posibles incidencias. A la tarea Ti se le asignará el tiempo de buffer histórico, o parte de él (t), para que la suma de t y el buffer de eficiencia energética sea igual al tiempo de buffer histórico. El valor de t se obtiene mediante la aplicación del algoritmo 11, el cual calcula, para cada velocidad, el tiempo que será sumado para proteger dicha tarea. Como se puede observar en el algoritmo 11, cuando la tarea es ejecutada a la máxima velocidad, el buffer histórico es asignado siempre por completo. Mediante la figura 5.1 se muestran los distintos casos que podrían aparecer:

a) El tiempo de procesado a máxima velocidad unido al tiempo de buffer histórico es más grande que el tiempo de procesado a velocidad media y baja. En este caso, todos los tiempos de procesado son completados con un tiempo de buffer, de modo que la suma del tiempo de procesado y el buffer asignado es igual para todas las velocidades de la máquina.

b) Se puede observar que el tiempo de procesado a la máxima velocidad más el tiempo de buffer histórico es mayor que el tiempo de procesado a velocidad media pero no que el tiempo a la velocidad más lenta. En este caso el buffer solo se asigna a la tarea dual cuando la velocidad es máxima y media.

c) El tiempo de procesado a la máxima velocidad más el tiempo de buffer histórico es menor que el tiempo de procesado a velocidad media y lenta. Por lo tanto, en este caso, el tiempo de buffer solo es asignado cuando la tarea dual se ejecuta a máxima velocidad.

Algoritmo 11: CalcularTiemposBuffer(Input (Tasks,BuffTime), Output (Tasks,BuffersTimes))

```
Begin
for (i=0;i<Ntasks;i++) do
    if (isDynamic(Tasks[i])) then
        if (BuffTime+Tasks[i].durSpeed3>Tasks[i].durSpeed2) then
            diff2=BuffTime+Tasks[i].durSpeed3-Tasks[i].durSpeed2;
        end if
        if (BuffTime+Tasks[i].durSpeed3>Tasks[i].durSpeed1) then
            diff1=BuffTime+Tasks[i].durSpeed3-Tasks[i].durSpeed1;
        end if
        diff3=BuffTime;
        BuffersTimes[i]=[diff1,diff2,diff3];
    end if
end for
End
```

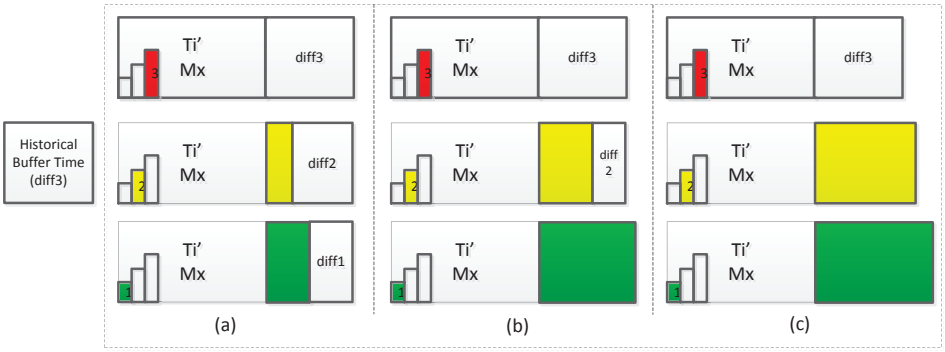


Figura 5.1: Los tres diferentes casos para la asignación del tiempo de buffer

5.2.3 Resolución del Problema JSMS

Una vez todas las tareas dinámicas han sido adaptadas a tareas duales, tenemos el problema en modo dual. El problema puede ser resuelto por cualquier algoritmo adecuado para ello, en nuestro caso usamos el GA+LS explicado en el capítulo anterior. Las soluciones obtenidas minimizan los objetivos de makespan y consumo de energía, según la función de optimización 4.10. El peso de cada uno de los objetivos es determinado por el valor de $\lambda \in [0, 1]$, de modo que finalmente obtenemos un conjunto de soluciones que forman la frontera pareto.

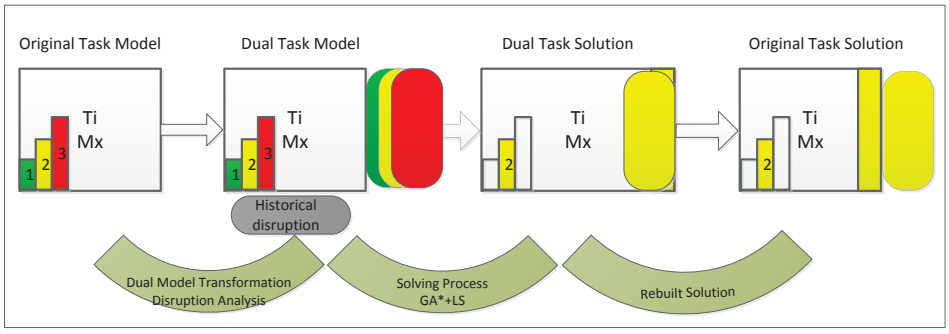


Figura 5.2: Transformación dual de una tarea del schedule

5.2.4 Adaptación de la Solución JSMS

Una vez las soluciones del problema dual han sido obtenidas, estas tienen que ser reconstruidas para obtener las soluciones del modelo original tal y como se representa en la figura 5.2, de tal modo que el tiempo que fue añadido para reforzar las tareas más dinámicas, ahora es considerado como un buffer y junto al buffer de eficiencia energética se puede absorber una incidencia del tamaño del tiempo de buffer histórico o menor. La solución se reconstruye mediante la aplicación del algoritmo 12.

Algoritmo 12: ReconstruirSolución(Input (Buffers,Solución), Output (Solución))

```
Begin
for (i=0;i<NBuff;i++) do
    task=Buff[i].idtask;
    speed=Solution[task].speed;
    switch (speed)
    case 1:
        buff= Buff[i].1;
        break;
    case 2:
        buff= Buff[i].2;
        break;
    case 3:
        buff= Buff[i].3;
        break;
    end switch
    if (buff>0) then
        Solution[task].endtime-=buff;
        Solution[task].time-=buff;
    end if
end for
End
```

En ocasiones algunos buffers se pueden generar para satisfacer las restricciones de preferencia. Si dicho buffer se genera a continuación de un buffer asignado por el modelo dual, y la suma de ambos es mayor que la máxima duración del historial de incidencias, la velocidad de la máquina se reduce (si es posible) para obtener una solución con menor consumo de energía manteniendo el mismo grado de robustez.

5.3 Evaluación de Robustez en JSMS

En esta sección, se evalúa el modelo dual propuesto y se compara frente al modelo original. Además, en la evaluación se compara el modelo dual dependiendo del valor del buffer histórico (IncMax e IncMed). Por último, se realiza un estudio de la estabilidad en las soluciones que no han podido absorber las incidencias mediante la robustez. La evaluación se ha realizado con las instancias de Agnetis y Watson presentadas anteriormente.

Tabla 5.2: Estudio comparativo entre el modelo original y el modelo dual con la duración media de las incidencias

3_5_10	Modelo Dual (IncMed)+ GA+LS				Modelo Original + GA+LS				Comparativa
λ	Mk	En	F	%Rob	Mk	En	F	%Rob	Diff %Rob
0	66	84.4	0.5537616	98.8	65.8	84.4	0.5537616	97.60	1.20
0.1	65.4	84.5	0.5567521	98.8	65.2	84.5	0.5565811	98.10	0.70
0.2	64.6	84.7	0.5590445	98.8	64.4	84.7	0.5587027	98.20	0.60
0.3	63.3	85.3	0.5602253	98	63.2	85.2	0.5594217	97.10	0.90
0.4	60.2	87.8	0.5586224	94	59.7	88.1	0.5578161	92.50	1.50
0.5	55.8	92.5	0.5497107	88	54	94.2	0.5472392	83.80	4.20
0.6	52.4	98.2	0.5350526	81.1	48.6	104	0.5298928	71.10	10.00
0.7	49.2	106	0.5125795	75.3	45.2	112.1	0.5002799	61.20	14.10
0.8	46.1	117.8	0.4810526	68.7	42.2	123.6	0.4616438	51.20	17.50
0.9	45.6	122.7	0.4435071	68.5	41	133.6	0.4146373	38.60	29.90
1	45.5	152	0.4023035	65.9	41	152	0.3630501	29.60	36.30

3_25_100	Modelo Dual (IncMed)+ GA+LS				Modelo Original + GA+LS				Comparativa
λ	Mk	En	F	%Rob	Mk	En	F	%Rob	Diff %Rob
0	2908.7	3827.1	0.5335316	99.80	2894.2	3827.1	0.5335316	91.80	8.00
0.1	2791.6	3827.7	0.5379550	98.30	2782.3	3827.6	0.5377443	91.58	6.72
0.2	2750.4	3840.5	0.5419934	97.26	2753	3839	0.541975	91.76	5.50
0.3	2642.1	3899.8	0.5444719	95.32	2632.7	3903.4	0.5441884	90.14	5.18
0.4	2529.1	3993	0.5433040	93.26	2516.7	4005.1	0.5432539	89.74	3.52
0.5	2380.1	4182.5	0.5376294	86.22	2334.2	4216.7	0.5353502	83.00	3.22
0.6	2147.6	4573.4	0.5217626	76.04	2129.6	4543.6	0.5177883	75.20	0.84
0.7	2022	4906.1	0.4980327	69.20	1919.4	5123.7	0.492521	61.44	7.76
0.8	1899.9	5451.8	0.4667197	59.72	1806.2	5671.6	0.4572355	50.04	9.68
0.9	1839.2	5943.9	0.4253255	53.64	1725.9	6261.1	0.4084004	36.72	16.92
1	1816.6	6948.4	0.3757725	48.18	1692.5	7084.6	0.3499826	25.74	22.44

Watson100	Modelo Dual (IncMed)+ GA+LS				Modelo Original + GA+LS				Comparativa
λ	Mk	En	F	%Rob	Mk	En	F	%Rob	Diff %Rob
0	12324.9	105478.1	0.5306175	100.00	12354.7	105478.1	0.5306175	93.74	6.26
0.1	12080.4	105480.1	0.5588896	100.00	12047	105478.1	0.5586523	93.46	6.54
0.2	12037.6	105509.6	0.5866911	99.98	12024.9	105502.8	0.5864981	93.43	6.55
0.3	12015.3	105699.9	0.6148584	99.85	11979.6	105806.4	0.6145107	93.37	6.47
0.4	11868.3	106408.8	0.6407627	99.49	11813.5	106537.2	0.6396736	93.11	6.38
0.5	11619.4	108681.5	0.6644313	97.97	11557	109370.6	0.6640938	91.63	6.34
0.6	9013.5	141328.7	0.6484557	75.91	8692.1	146631.3	0.6461636	72.36	3.55
0.7	8873.8	144645	0.6364502	73.70	8537.6	150891.8	0.6300417	69.19	4.51
0.8	8739	147060.5	0.6185831	71.63	8421.7	153772	0.6082655	66.21	5.42
0.9	8706.2	148108.1	0.6020297	70.71	8419.4	153626.4	0.5873666	67.41	3.30
1	7932.1	195038.4	0.5339947	45.23	7306.7	190808.3	0.4918773	23.68	21.55

Primero se compara el comportamiento del modelo original y del modelo dual. Ambos modelos han sido ejecutados mediante el algoritmo anteriormente explicado GA+LS. Ambos modelos son analizados en términos de makespan, consumo de energía y robustez. Las tablas 5.2 muestran los resultados para las instancias Agnetis 3_5_10 y 3_25_100 y Watson100. En todos los casos los experimentos fueron realizados con 10 instancias diferentes de cada tipo, por lo que los valores de las tablas muestran las medias obtenidas. Los valores de robustez son también medias de todas las incidencias analizadas, por lo que dichos valores se muestran como un porcentaje. Las columnas Diff %Rob es el resultado de restar el valor de robustez de ambos modelos. Las incidencias son generadas aleatoriamente siguiendo la proporción de la tabla 5.1 y la duración de cada incidencia es un valor aleatorio entre el 1 % y el 20 % del tiempo de procesado máximo.

Las tablas 5.2 muestran los valores de makespan, consumo de energía, F y robustez para los diferentes valores de λ. Se puede observar en las tablas que todos los valores de makespan son menores para el modelo original. Esto es debido a que en el modelo dual existen buffers que empeoran dicho valor, consiguiendo a cambio que el valor de la robustez sea significativamente más alto. Un claro ejemplo puede ser visto en los resultados para las instancias 3_5_10 de la tabla 5.2, donde cuando $\lambda = 1$ el makespan medio para el modelo original es igual a 41, mientras que para el modelo dual es igual a 45,5. En cambio el valor de robustez para el modelo original es de 29.6 % y para el modelo dual de 65.9 %. Por lo tanto, se puede observar que empeorando la solución con 4,5 unidades de makespan, la robustez se ve incrementada en un 36.3 %. Si comparamos el valor de energía consumida para los mismos valores de λ de ambos modelos, podemos observar que para la mayoría de los valores de $\lambda > 0{,}5$ el consumo de energía es menor en el modelo dual. Esto es debido a que, las tareas que no están en el camino crítico pueden ser ejecutadas a menor velocidad con mayor facilidad y a que en la fase de reconstrucción de la solución algunas máquinas pueden reducir su velocidad.

En general, las soluciones obtenidas por el modelo dual obtienen un valor mayor de F, pero las soluciones muestran un grado de robustez considerablemente mayor. Como se puede observar el valor de Diff %Rob es siempre positivo, y esta diferencia aumenta a medida que va aumentando el valor de λ, que es cuando el makespan adquiere mayor importancia. Al aplicar del modelo dual se consigue que:

- Cuando solo se optimiza el makespan, se obtenga un valor peor de makespan pero en cambio mejora la robustez considerablemente.
- Cuando solo se optimiza la eficiencia energética, la robustez aumente sin perder optimalidad.
- Cuando se optimizan las dos variables, el makespan empeore pero, en muchos casos, el consumo de energía se reduce y el grado de robustez siempre mejora.

Tabla 5.3: Comparativa entre el modelo Dual con IncMax y el modelo dual con IncMEd

7_10_100	Modelo Dual (IncMax) + GA+LS				Modelo Dual (IncMed)+ GA+LS				Comparativa
λ	Mk	En	F	%Rob	Mk	En	F	%Rob	Diff %Rob
0	1008.9	1571.4	0.5336161	97.80	1006.6	1571.4	0.5336161	97.30	0.50
0.1	1002.9	1572.5	0.5411091	97.65	1000.6	1572.5	0.5409746	97.15	0.50
0.2	991	1576.4	0.5479191	96.85	988.5	1576.4	0.5476258	96.60	0.25
0.3	933	1610.1	0.5520634	94.40	928.9	1611.2	0.5514574	93.25	1.15
0.4	901.5	1641	0.5526405	92.10	894.4	1642.3	0.5512247	89.55	2.55
0.5	868.4	1687.1	0.5492773	88.05	853.7	1703.4	0.5475819	85.00	3.05
0.6	812.6	1817.3	0.5416006	81.55	801	1823.1	0.5385088	77.70	3.85
0.7	764.5	1983.5	0.5259537	71.35	744.4	2006	0.5198095	66.95	4.40
0.8	720.8	2203.5	0.4988613	63.00	683.1	2314.9	0.4883531	53.15	9.85
0.9	709.6	2323.9	0.4656537	62.00	673.1	2420.3	0.4489425	49.25	12.75
1	705.9	2935.1	0.4272285	62.20	671.3	2935.1	0.4062202	40.50	21.70

Watson50	Modelo Dual (IncMax) + GA+LS				Modelo Dual (IncMed)+ GA+LS				Comparativa
λ	Mk	En	F	%Rob	Mk	En	F	%Rob	Diff %Rob
0	7252.9	53631.5	0.5340844	100.00	7253.7	53631.5	0.5340844	100.00	0.00
0.1	7035.8	53631.5	0.5553904	100.00	7019.8	53637.3	0.5552724	99.99	0.01
0.2	6994.7	53694.2	0.5763237	99.97	6985.5	53675.8	0.5759795	99.91	0.05
0.3	6960	53754.4	0.5964355	99.86	6910.7	53877.4	0.5957130	99.71	0.15
0.4	6886.9	54071.2	0.6155995	99.67	6821	54422.1	0.6149003	98.95	0.72
0.5	6749	54848.6	0.6314064	98.76	6690.5	55244.8	0.6302893	97.97	0.79
0.6	5711.4	65252.4	0.6238081	88.64	5411	68906.6	0.6192607	80.55	8.08
0.7	5459.9	68755.2	0.6112227	85.20	5185.6	72321.3	0.6014895	75.37	9.83
0.8	5363.4	70593.5	0.5962423	83.45	5078.9	74153.8	0.5791645	72.34	11.11
0.9	5302.5	71753.7	0.5782274	82.05	5030.8	75624.5	0.5561081	70.41	11.64
1	4945.9	95791.2	0.5252671	70.55	4633.3	98728.8	0.4920303	47.29	23.26

Tabla 5.4: Estudio de estabilidad

3_25_100	Modelo Dual (IncMed)+ GA+LS			
λ	% Rob	% S1	% S2	% Estabilidad Total
0	99.80	0.20	0.00	100.00
0.1	98.30	0.80	0.74	99.84
0.2	97.26	1.38	0.88	99.52
0.3	95.32	1.48	2.06	98.86
0.4	93.26	1.68	3.06	98.00
0.5	86.22	3.34	4.70	94.26
0.6	76.04	4.70	6.74	87.48
0.7	69.20	5.46	6.42	81.08
0.8	59.72	5.22	5.40	70.34
0.9	53.64	5.60	4.40	63.64
1	48.18	9.24	6.64	64.06

Watson50	Modelo Dual (IncMed)+ GA+LS			
λ	% Rob	% S1	% S2	% Estabilidad Total
0	100.00	0.00	0.00	100.00
0.1	99.99	0.01	0.00	100.00
0.2	99.91	0.07	0.00	99.98
0.3	99.71	0.17	0.05	99.93
0.4	98.95	0.52	0.15	99.62
0.5	97.97	1.09	0.23	99.29
0.6	80.55	8.31	2.34	91.20
0.7	75.37	9.93	2.55	87.85
0.8	72.34	10.57	2.66	85.57
0.9	70.41	10.72	2.66	83.78
1	47.29	17.02	4.42	68.73

Las tablas 5.3 muestran los resultados de las instancias Agnetis y Watson50 para analizar el comportamiento del modelo dual cuando cambia el tamaño del tiempo histórico de buffer. Con este fin, hemos analizado dos tiempos distintos de buffer: Tiempo de buffer con la máxima duración de las incidencias para cada tipo de instancias (IncMax) y tiempo de buffer con el tamaño medio de todas las incidencias para cada tipo de instancias (IncMed). Mediante las tablas 5.3 se representa el comportamiento general del resto de instancias Agnetis y Watson, ya que es similar en todas ellas.

Como se puede observar en las tablas 5.3 el modelo dual con IncMax tiene un valor de F mayor que el modelo dual con IncMed. Sin embargo, el valor de la robustez es siempre mayor para el modelo dual con IncMax. Esto es debido a que el tiempo de buffer IncMax es mayor que IncMed, por lo tanto el valor de makespan es mayor y eso supone que el valor de F aumente. Por el contrario, al aumentarse el tamaño de los buffer, la robustez también aumenta, ya que un buffer mayor puede absorber más incidencias. Por ejemplo para $\lambda = 1$ en la instancia 7_10_100, se puede observar que la diferencia entre ambos makespans es de 34.6, pero a cambio la robustez aumenta en un 21,70 %.

Para finalizar con la evaluación, hemos analizado la estabilidad del modelo dual para las instancias Agnetis y Watson. Las tablas 5.4 muestran el porcentaje de robustez o

0-estabilidad (%Rob), 1-estabilidad (%S1) y 2-estabilidad (%S2), y el porcentaje de estabilidad total (% Estabilidad Total) el cual se obtiene mediante la suma de los tres parámetros previos. En las tablas 5.4 se muestran los resultados para las instancias Agnetis 3_25_100 y Watson50 ya que el comportamiento para el resto de instancias es muy similar.

Como se puede observar, %Rob tiene un valor mayor que el %S1 y %S2. Esto es debido a que el modelo dual destina tiempos de buffers a las tareas más dinámicas, y por lo tanto son ellas mismas las que absorben las incidencias. Cuando aparece una incidencia, se trata de absorber por la misma tarea que sufre la incidencia. Si esta tarea no es capaz de absorberla, se comprueba si la siguiente tarea del mismo trabajo y si la siguiente tarea ejecutada por la misma máquina son capaces de absorber la incidencia. Si la incidencia puede ser absorbida solo cambiando una de las tareas (misma máquina o mismo trabajo), consideramos que es 1-estable. Sin embargo, si debemos modificar ambas tareas lo consideraremos 2-estable.

Tanto para las instancias Agnetis como Watson, el porcentaje de estabilidad total es cercano al 100 % para valores de $\lambda < 0{,}5$. Por ejemplo, cuando $\lambda = 0$, todas las incidencias fueron absorbidas con 0, 1 o 2 cambios sobre las soluciones originales como indica el 100 % de estabilidad total. Además, cuando los valores de λ aumentan (el makespan es minimizado), el porcentaje de robustez disminuye y el número de incidencias no absorbidas aumenta. Sin embargo, la 1-estabilidad y la 2-estabilidad aumentaron para absorber algunas de las incidencias que no fueron absorbidas directamente por el modelo dual mediante la robustez. Por ejemplo, en las instancias Watson50, para $\lambda = 1$ el porcentaje de estabilidad total aumenta un 21,44 % con respecto a la robustez (47,29 %), alcanzando un porcentaje total de la estabilidad del 68,73 %.

5.4 Conclusiones

Los procesos industriales pueden involucrar un gran número de problemas de scheduling. Mediante el problema job shop scheduling donde las máquinas pueden trabajar a distintas velocidades se pueden representar un gran número de problemas combinatorios. En esta sección se ha propuesto un modelo dual para relacionar el criterio de consumo de energía con la robustez y estabilidad. Mediante este modelo se protege a las tareas dinámicas contra incidencias con el fin de obtener soluciones robustas y al mismo tiempo soluciones eficientes energéticamente.

El modelo dual propuesto ha sido evaluado con un algoritmo memético (GA+LS) para comparar el comportamiento frente al modelo original. Los resultados muestran que el modelo dual propuesto fue capaz de obtener soluciones más robustas en todas las instancias, y soluciones de mayor eficiencia energética en la mayoría de los casos. Esto es debido al hecho de que los buffers asignados a algunas tareas se utilizan para protegerlas

de las incidencias que puedan aparecer, pero además, si durante la ejecución no aparece ninguna incidencia, la máquina que ejecuta la tarea puede trabajar a menor régimen de velocidad aprovechando el buffer, y reduciendo el consumo de energía. Se ha analizado también que cuando se aumente el tamaño de los buffers, las soluciones son más robustas, pero se pierde optimalidad. Es por ello que hay que buscar un equilibrio entre ambos conceptos dependiendo de las necesidades del problema. Si además de tener en cuenta la robustez, se aplica el concepto de estabilidad, añadimos la opción de que si una incidencia no es absorbida directamente por la tarea interrumpida puede ser recuperado el schedule mediante la participación de sólo un pequeño número de tareas. Así, la solución original se puede recuperar con el fin de mantener viable el resto del schedule original.

Capítulo 6

Análisis de Parámetros en JSMS: Robustez, Consumo Energético y Makespan

6.1 Introducción

Como se ha mostrado anteriormente existe una clara relación entre el consumo de energía y la optimalidad y es por ello que existen un gran número de trabajos que se han encargado de investigar esta relación, como por ejemplo el trabajo realizado por Lee y Zomaya 2009, donde abordan el problema de scheduling en aplicaciones paralelas con limitaciones, teniendo en cuenta tanto el tiempo total de finalizado como el consumo de energía. Otro trabajo donde también se relacionan ambos conceptos es el realizado por Chang-tian y Jiong 2012, en el cual se consideran tareas de scheduling independientes en un ambiente de computación en la nube como un problema bi-objetivo donde se optimiza el makespan y el consumo energía. Dos características que también están relacionadas en los problemas de scheduling son la robustez y la optimalidad. Ambos objetivos también han sido ampliamente estudiados en numerosos trabajos, un ejemplo de ello es el trabajo realizado por Abbasi, Shadrokh y Arkat 2006 donde optimizan el makespan y la robustez en el problema de scheduling de proyectos de recursos limitados. Otro trabajo a destacar puede ser el realizado por Goren y Sabuncuoglu 2008, donde se estudia la robustez y la estabilidad en un contexto de scheduling con la simulación de averías de máquinas al azar.

Pero como se ha visto en los capítulos anteriores, en el problema JSMS también existe una relación entre la robustez, el makespan y el consumo energético. En este capítulo se

realiza un análisis para ver cuál es la relación entre el makespan, consumo de energía y la robustez. Mediante el estudio de estas relaciones se pueden encontrar soluciones muy interesantes para sistemas de producción industriales, ya que, son muchas las áreas donde podría resultar interesante optimizar robustez, makespan y consumo de energía (Salido, Escamilla, Giret, Barber y col. 2013). Pese a ello, esta relación no ha sido estudiada en la literatura y su estudio podría derivar en la investigación y desarrollo de nuevas técnicas para mejorar estos objetivos de forma conjunta.

6.2 Makespan versus Consumo Energético

En esta sección se analiza la relación entre el makespan y el consumo de energía en el problema job shop scheduling donde las máquinas pueden trabajar a distintas velocidades. En la figura 6.1 se muestra la frontera pareto para la media de un conjunto de 10 instancias con 5 máquinas, 10 tareas por trabajo y un tiempo máximo de procesado de 50 unidades. Para $\lambda = 1$ se puede observar que la media del consumo de energía es de 1311 y la media del makespan de 317. Sin embargo para $\lambda = 0$, la media del consumo de energía es de 745 y la media del makespan de 565,4. Dependiendo de las necesidades del usuario puede interesar obtener un tipo de solución u otro dependiendo del tipo de problema o del objetivo a optimizar. Es por ello que, mediante el valor de λ se puede dar más prioridad al makespan o al consumo de energía. En la tabla 6.1 se muestra el makespan y el consumo de energía para cada valor de λ en diferentes instancias. Como se puede observar, la relación entre el consumo de energía y el makespan es parecida para todos los casos, de modo que esta relación no es dependiente ni del número de máquinas, ni del número de tareas por trabajo, ni de los tiempos de procesamiento.

Tabla 6.1: Makespan y consumo de energía para las instancias $< m, v_{max}, p >$

	5_10_50		7_10_100		3_20_50		3_25_100		3_30_200	
λ	**Mk**	**Energ.**	**Mk**	**Energ.**	**Mk**	**Energ.**	**Mk**	**Energ.**	**Mk**	**Energ.**
0	565.4	745	1088.4	1571.4	1296	1507.4	3160	3827.1	7289.6	9162.7
0.1	524.4	745	1004.3	1571.6	1168.7	1507.4	2768.4	3827.5	6600.8	9163.5
0.2	515.8	747.1	992.1	1574.5	1145.7	1513.7	2734.2	3835.7	6513.2	9184
0.3	502.6	752.3	970.2	1584.2	1112.6	1527.3	2667.1	3866.7	6364.8	9258.8
0.4	483.6	764.2	918.3	1616.3	1079	1559.9	2553	3946.1	6158	9386.5
0.5	454.3	792.1	884.5	1650.3	1011.7	1628.6	2421.8	4077	5825	9771.5
0.6	410.1	854.1	843.7	1709.8	946.8	1722.5	2280.6	4300.6	5452	10239.4
0.7	384.7	917.4	768.2	1879.5	854.6	1933.7	2056.1	4742.9	4953.3	11370
0.8	343.1	1053	690.1	2147.3	777.8	2216.3	1854.4	5466.1	4530.8	12935.4
0.9	322.8	1179.1	635.7	2466.4	728.3	2498	1738.4	6164.9	4182.9	14696.1
1	317	1311.5	625.9	2664.1	707.8	2764.3	1686.1	6667.1	4048.9	16235.5

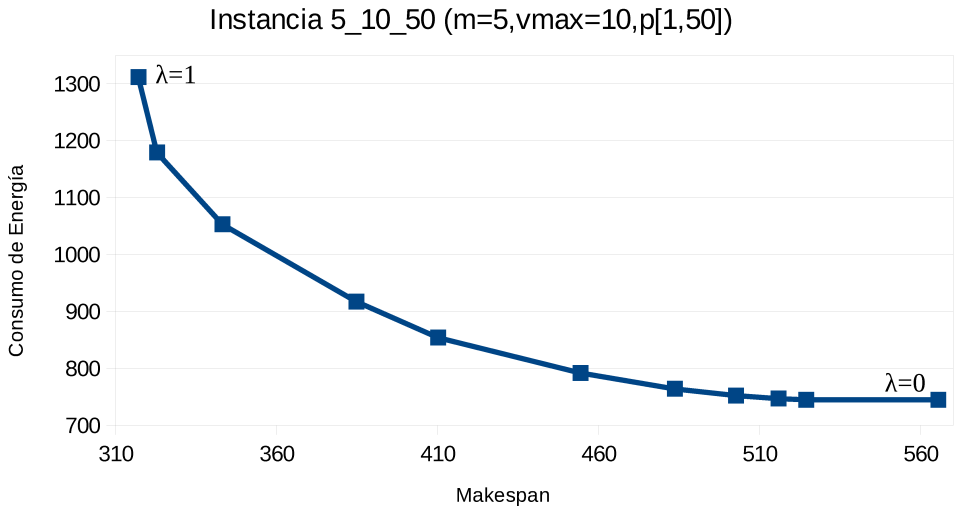


Figura 6.1: Frontera pareto para la optimización del makespan y el consumo de energía.

Mediante el análisis de las instancias analizadas se puede realizar una estimación que represente la relación entre el consumo de energía y makespan, dicha relación es representada por la fórmula (6.1):

$$\frac{Energy(\lambda)}{Makespan(\lambda)} \approx e^{0,25+1,2\lambda} - \frac{sin(\lambda\pi)}{2} + \frac{cos(\lambda\pi)}{8} \tag{6.1}$$

Mediante el uso de esta fórmula, dada una instancia de un schedule con un makespan dado y un valor de λ, podemos estimar el consumo de energía necesario para ejecutar este schedule. De la misma forma, dada una instancia de un schedule con un umbral de consumo de energía dado y un valor λ, podemos estimar el makespan necesario para ejecutar dicho schedule.

6.3 Robustez versus Consumo Energético

El objetivo principal de esta sección es mostrar la relación entre la robustez y el consumo de energía. De esta manera, la ventaja podría ser doble. El desarrollo de nuevas técnicas para la búsqueda de soluciones eficientemente energéticas también puede significar la búsqueda de soluciones robustas, ya que pequeñas interrupciones pueden ser absorbidas mediante la aceleración de las máquinas para recuperar la solución original. Por lo tanto, si tenemos una solución donde las máquinas trabajan a poca velocidad, el consumo de

energía será menor. Además, en caso de que aparezca una incidencia, mediante la aceleración de las máquinas se puede conservar la solución sin que sea necesario replanificar.

Con el fin de analizar la cantidad de incidencias que pueden ser absorbidas por el schedule resultante, hemos simulado 100 incidencias para cada instancia. Una incidencia es un retraso en una tarea aleatoria del schedule. La duración máxima de la incidencia (%incid) está limitada al 20 % del tiempo máximo de procesamiento de las tareas para cada tipo de instancia. La figura 6.2 muestra la frontera pareto para un conjunto de 10 instancias con 7 máquinas, 10 tareas por trabajo y un tiempo máximo de procesamiento de 100 unidades de tiempo. Se puede observar que a medida que aumenta la robustez, disminuye el consumo de energía. Esto es debido al hecho de que las soluciones más robustas permiten que las máquinas trabajen a velocidad mínima, por lo que el consumo de energía disminuye. Es decir, si todas las máquinas funcionan a la velocidad mínima, todas las tareas tienen un buffer de eficiencia energética (tiempo entre ejecutar la tarea a la velocidad máxima y ejecutarla a la velocidad mínima). Por lo tanto, si una interrupción se produce en una máquina m_i a velocidad (s_{i1}) durante la ejecución de la tarea t_i, esta máquina puede acelerar su velocidad a s_{i2} en la tarea t_i con el fin de terminar a tiempo (antes de que comience la tarea siguiente t_{i+1}). En este caso, consideramos que el schedule es robusto.

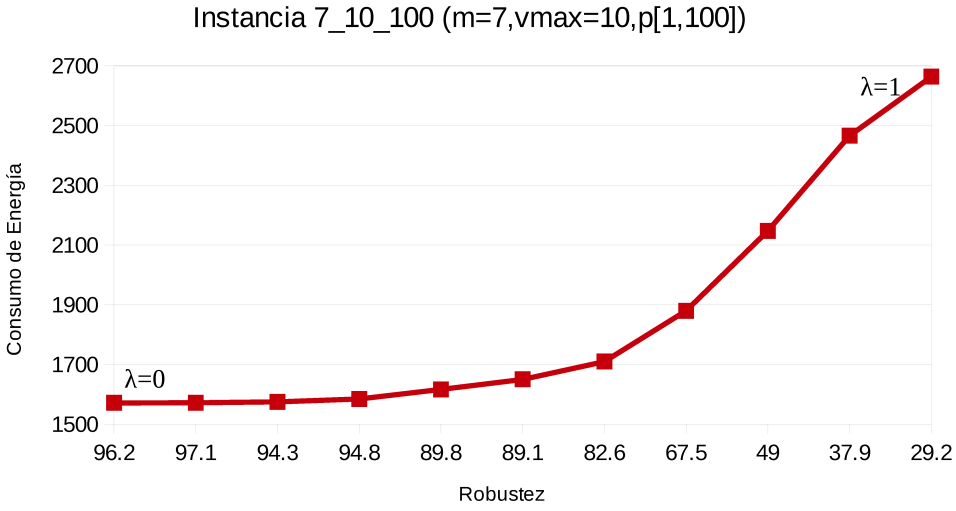


Figura 6.2: Frontera pareto para la optimización de la robustez y el consumo de energía

La tabla 6.2 muestra el consumo de energía y robustez de diferentes instancias. Hay que tener en cuenta que la robustez mantiene el mismo comportamiento en todos los casos, por lo que se deduce que la robustez no es directamente dependiente ni de la cantidad de máquinas, ni del número de tareas por trabajo, ni de los tiempos de procesamiento. Sin embargo, en la mayoría de los casos, para $\lambda = 0, 1$ y $\lambda = 0$, la energía necesaria es similar, pero la robustez es diferente (ver últimos dos filas de la tabla 6.2). Por lo tanto, dado un

Tabla 6.2: Consumo de energía y robustez para las instancias $< m, v_{max}, p >$

	5_10_50		7_10_100		3_20_50		3_25_100		3_30_200	
λ	**Ener.**	**Robust.**	**Ener.**	**Robust.**	**Ener.**	**Robust.**	**Ener.**	**Robust.**	**Ener.**	**Robust.**
0	745	97.2 %	1571.4	96.2 %	1507.4	99.0 %	3827.1	98.9 %	9162.7	99.2 %
0.1	745	93.7 %	1571.6	97.1 %	1507.4	97.3 %	3827.5	97.6 %	9163.5	96.3 %
0.2	747.1	97.0 %	1574.5	94.3 %	1513.7	96.6 %	3835.7	96.0 %	9184	96.8 %
0.3	752.3	94.2 %	1584.2	94.8 %	1527.3	95.3 %	3866.7	94.9 %	9258.8	95.9 %
0.4	764.2	91.8 %	1616.3	89.8 %	1559.9	93.4 %	3946.1	94.1 %	9386.5	92.7 %
0.5	792.1	86.6 %	1650.3	89.1 %	1628.6	88.0 %	4077	86.3 %	9771.5	88.0 %
0.6	854.1	79.8 %	1709.8	82.6 %	1722.5	79.6 %	4300.6	78.9 %	10239.4	80.9 %
0.7	917.4	68.1 %	1879.5	67.5 %	1933.7	66.0 %	4742.9	67.9 %	11370	65.2 %
0.8	1053	48.2 %	2147.3	49.0 %	2216.3	50.6 %	5466.1	49.5 %	12935.4	48.9 %
0.9	1179.1	37.2 %	2466.4	37.9 %	2498	36.3 %	6164.9	36.8 %	14696.1	39.7 %
1	1311.5	26.7 %	2664.1	29.2 %	2764.3	26.8 %	6667.1	27.0 %	16235.5	25.6 %

umbral de consumo de energía, podemos obtener diferentes soluciones con diferente nivel de robustez y makespan.

La relación entre el consumo de energía y la robustez se puede obtener de forma empírica tras el análisis de los resultados obtenidos en las instancias analizadas, de modo que, mediante el uso de la fórmula (6.2) se puede estimar dicha relación:

$$\frac{Energy(\lambda)}{Robustness(\lambda, \%incid)} \approx \frac{ttasks \cdot p/2}{(6,3 - 6\frac{\%incid}{100}) \cdot (8 * Cos(\lambda^2\pi) - Sin(\lambda\pi) + 11)} \tag{6.2}$$

Esta fórmula es más precisa para los valores de λ cercanos a 0 (de 0 a 0,6), debido al hecho de que el consumo de energía tiene más importancia para estos valores en la función objetivo. Por lo tanto, dado un porcentaje de robustez para una duración de incidencia dada (%incid) y un valor de λ, podemos estimar la energía necesaria para llevar a cabo este schedule. De la misma manera, en un schedule con un consumo de energía dado, un valor de λ y un umbral de la duración de la incidencia (%incid), podemos estimar su robustez.

6.4 Robustez versus Makespan

Se puede decir que existe una relación directa entre el makespan y la robustez porque cuando aumenta el makespan, la robustez es mayor debido a que las tareas están más dispersas en el tiempo y son capaces de absorber más incidencias. Sin embargo, no es realista generar schedules demasiado dispersos. En ocasiones en problemas reales se define un makespan máximo y se trata de encontrar el schedule más robusto para el umbral del makespan dado.

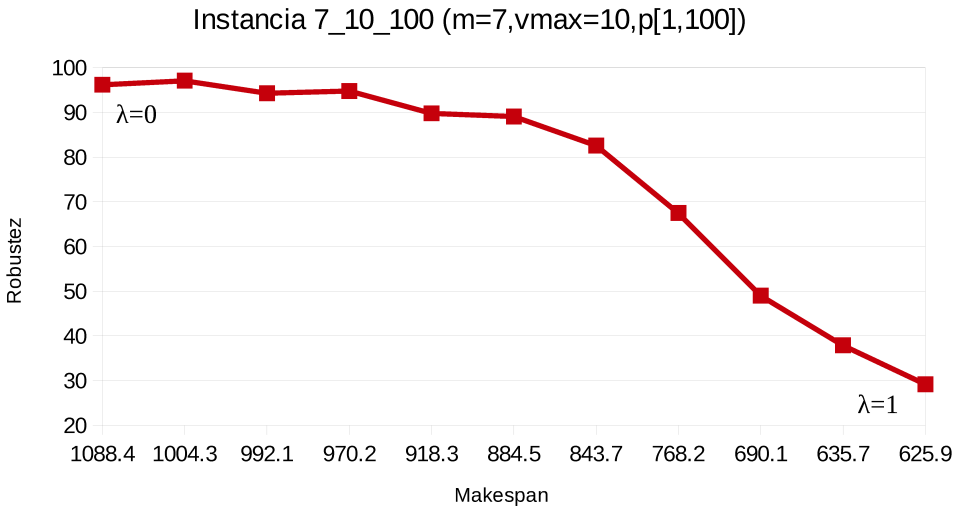


Figura 6.3: Frontera pareto para la optimización de la robustez y el makespan.

La simulación realizada en el apartado anterior obtuvo como resultado la cantidad de incidencias que se pueden absorber modificando el consumo de energía. La figura 6.3 muestra la frontera Pareto para un conjunto de 10 instancias con 7 máquinas, 10 tareas por trabajo y un tiempo máximo de procesamiento de 100 unidades de tiempo. Se puede observar que a medida que el makespan aumenta la robustez también lo hace. La tabla 6.3 muestra la comparación entre el makespan y la robustez en diferentes instancias. Hay que tener en cuenta que la robustez es bastante similar en todos los casos, lo que demuestra que no es directamente dependiente ni de la cantidad de máquinas, ni del número de tareas por trabajo, ni del tiempo de procesamiento. Cuando el makespan se ajusta al mínimo posible (para alcanzar la solución óptima), estas soluciones son capaces de absorber un promedio de 27 % de las incidencias (primera fila de la Tabla 6.3). Esto es debido al hecho de que los buffers naturales fueron capaces de absorber este porcentaje de incidencias. Por último, cuando el makespan es ajustado a un límite superior (reduciendo al mínimo el consumo de energía), el porcentaje de incidencias absorbidas se encuentra próximo al 100 %. Esto significa que los buffers están bien distribuidos entre todas las tareas y casi todas las interrupciones han sido capaces de ser absorbidas.

La relación entre makespan y robustez se puede obtener a partir de las fórmulas anteriores (6.1) y (6.2) para obtener la fórmula (6.3):

$$\frac{Makespan(\lambda)}{Robustness(\lambda,\ \%incid)} \approx$$

Tabla 6.3: Makespan y robustez para las instancias $< m, v_{max}, p >$

	5_10_50		7_10_100		3_20_50		3_25_100		3_30_200	
λ	**Mk**	**Robust.**	**Mk**	**Robust.**	**Mk**	**Robust.**	**Mk**	**Robust.**	**Mk**	**Robust.**
0	565.4	97.2 %	1088.4	96.2 %	1296	99.0 %	3160	98.9 %	7289.6	99.2 %
0.1	524.4	93.7 %	1004.3	97.1 %	1168.7	97.3 %	2768.4	97.6 %	6600.8	96.3 %
0.2	515.8	97.0 %	992.1	94.3 %	1145.7	96.6 %	2734.2	96.0 %	6513.2	96.8 %
0.3	502.6	94.2 %	970.2	94.8 %	1112.6	95.3 %	2667.1	94.9 %	6364.8	95.9 %
0.4	483.6	91.8 %	918.3	89.8 %	1079	93.4 %	2553	94.1 %	6158	92.7 %
0.5	454.3	86.6 %	884.5	89.1 %	1011.7	88.0 %	2421.8	86.3 %	5825	88.0 %
0.6	410.1	79.8 %	843.7	82.6 %	946.8	79.6 %	2280.6	78.9 %	5452	80.9 %
0.7	384.7	68.1 %	768.2	67.5 %	854.6	66.0 %	2056.1	67.9 %	4953.3	65.2 %
0.8	343.1	48.2 %	690.1	49.0 %	777.8	50.6 %	1854.4	49.5 %	4530.8	48.9 %
0.9	322.8	37.2 %	635.7	37.9 %	728.3	36.3 %	1738.4	36.8 %	4182.9	39.7 %
1	317	26.7 %	625.9	29.2 %	707.8	26.8 %	1686.1	27.0 %	4048.9	25.6 %

$$\approx \frac{ttasks \cdot p/2}{(6,3 - 6\frac{\%incid}{100}) \cdot (8 * Cos(\lambda^2\pi) - Sin(\lambda\pi) + 11) \cdot (e^{(0,25+1,2\lambda)} - \frac{Sin(\lambda\pi)}{2} + \frac{Cos(\lambda\pi)}{8})} \tag{6.3}$$

Podemos concluir que, dado el makespan de un schedule con un valor de λ dado, y la duración de la incidencia (%incid) conocida, se puede estimar la robustez de este schedule. De la misma manera, dado un umbral de robustez, la duración de la incidencia (%incid) y el valor de λ, podemos estimar el makespan de dicho schedule.

6.5 Análisis General

En esta sección, se ha llevado a cabo un análisis general para los distintos tipos de instancias. El objetivo principal es analizar la relación entre todos los parámetros relevantes de robustez y eficiencia energética para diferentes valores de λ (eje horizontal). La figura 6.4 muestra los resultados para las interrupciones del 40 % del tiempo máximo de procesamiento (%incid=40). En el eje principal vertical se representa la robustez. Por lo tanto, la curva azul (*% of Absorbed (40 %)*) representa el porcentaje de incidencias absorbidas para cada valor de λ (Robustez). La curva amarilla (*%Natural Buff*) representa el porcentaje de incidencias absorbidas por un buffer natural. La curva verde (*%EffEn Buff*) representa el porcentaje de incidencias absorbidas por la aceleración de una máquina. De esta manera, la robustez es *% of Absorbed (40 %)= %Natural Buff + %EffEn Buff*.

En el eje vertical secundario, las columnas granate (*NbuffEff*) representan el número medio de buffers generados por el aumento de la velocidad de las máquinas y las columnas azul claro (*NbuffNat*) representan el número medio de buffers naturales.

Se puede observar que *NbuffNat* es prácticamente constante porque es independiente del objetivo (minimizar el consumo de energía o makespan). Sin embargo, la cantidad total de tiempo que participa en estos buffers naturales disminuye a medida que el valor de λ aumenta. Esto se debe al hecho de que a medida que λ se incrementa, la función objetivo

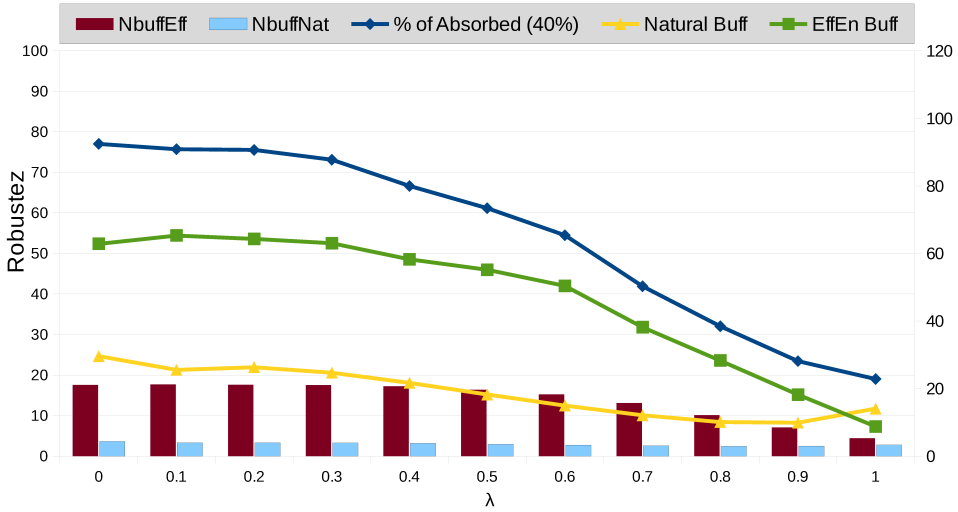


Figura 6.4: Análisis General de las interrupciones del 40 % del máximo tiempo de procesamiento (%incid=40).

da más importancia a minimizar el makespan, reduciéndose, de este modo, la holgura entre las tareas al mínimo. Por lo tanto, el porcentaje de veces que la incidencia es absorbida por un buffer natural (*% Natural Buff*) también disminuye. La misma tendencia se lleva a cabo con *NbuffEff*, donde el número de buffers generados disminuye a medida que λ aumenta. Esto se debe al hecho de que, cuando λ aumenta, el objetivo prioriza reducir al mínimo el makespan, asignándoles a más máquinas la máxima velocidad, por lo que pocas veces se pueden generar buffers por la aceleración de las máquinas.

El porcentaje absorbido de incidencias (*%Effen Buff*) también disminuye cuando λ aumenta. Sin embargo, se puede observar la diferencia entre el porcentaje de incidencias absorbidas por la aceleración de las máquinas (*%Effen Buff*) y el porcentaje de incidencias absorbidas por los buffers naturales (*%Natural Buff*). Se puede observar que para $\lambda = 0$ (minimizando el consumo de energía), casi todas las incidencias pueden ser absorbidas. Por lo tanto, cuando se consideran schedules eficientemente energéticos, también se consideran schedules robustos que pueden absorber incidencias.

En la figura 6.5 hemos simulado incidencias de diferente longitud, desde el 10 % al 40 % del tiempo de procesamiento máximo (de %indic=10 a %indic=40). La curva roja (*% of absorbed (10 %)*), la curva verde (*% of absorbed (20 %)*), la curva de granate (*% of absorbed (30 %)*) y la curva gris (*% of absorbed (40 %)*) representan el porcentaje de incidencias absorbidas para cada valor de λ cuando las incidencias son del 10 %, 20 %, 30 % y 40 % del tiempo de procesamiento máximo, respectivamente. Se puede observar que todas las curvas mantienen el mismo comportamiento para los distintos valores de λ,

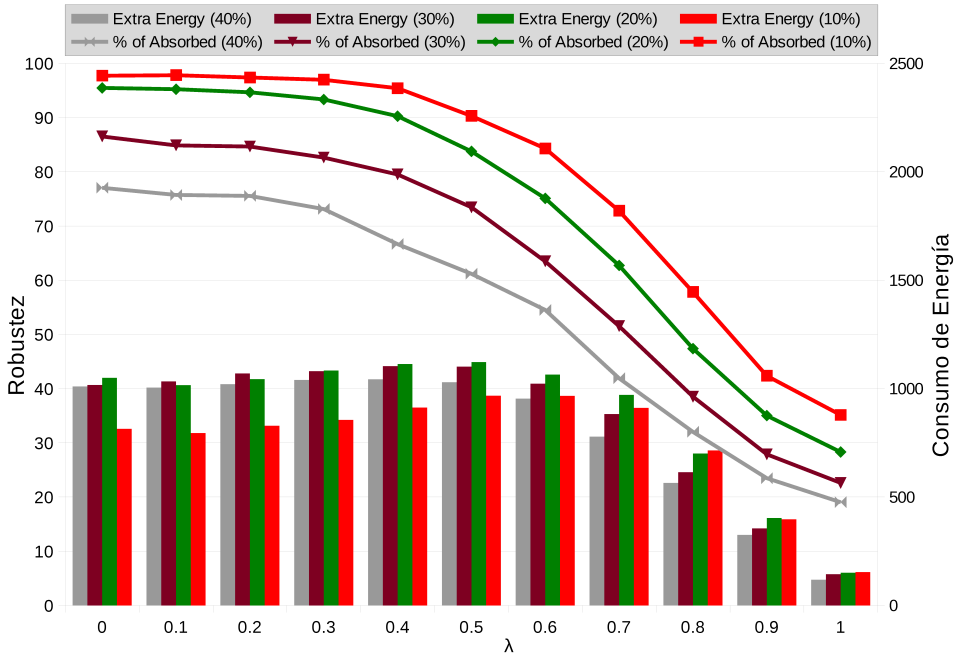


Figura 6.5: Interrupciones desde el 10 % hasta el 40 % del máximo tiempo de procesamiento.

y que los valores son proporcionales a la longitud de las interrupciones. Esto es debido al hecho de que es más fácil absorber incidencias pequeñas que grandes, pero la diferencia no es demasiado alta. Por lo tanto, las incidencias más largas que el 40 % mantendrán la misma tendencia (proporcional a la presentada en la figura 6.5).

En el eje vertical secundario se representa la energía extra necesaria para absorber las incidencias dependiendo de la longitud de las interrupciones (del 10 % al 40 %). Se puede observar que, aunque *% of Absorbed (20 %)* fue capaz de absorber menos interrupciones que *% of Absorbed (10 %)*, este necesita más energía adicional que en muchos casos del 10 %. Debe tenerse en cuenta que a medida que el número de interrupciones absorbidas aumenta, la energía extra necesaria para absorber estas interrupciones también aumenta, siendo la magnitud de la energía necesaria proporcional al tamaño de la interrupción. Por ejemplo, para $\lambda = 0$ (minimizando solo consumo de energía), el porcentaje de interrupciones absorbidas de tamaño 40 % fue de alrededor del 77 %, mientras que el porcentaje de interrupción absorbida de tamaño 30 % fue de alrededor de 87 %. Sin embargo, la energía extra necesaria para absorber estas incidencias fue casi la misma en ambos casos debido al hecho de que las interrupciones mayores precisan más energía extra.

6.6 Conclusiones

En este capítulo se analiza la relación entre tres objetivos importantes que se pueden tener en cuenta en el problema JSMS: la eficiencia energética, la robustez y el makespan (Salido, Escamilla, Barber y col. 2015). Se han desarrollado fórmulas analíticas para estimar la relación entre estos objetivos. Los resultados muestran la relación entre makespan y robustez, y la relación directa entre la robustez y eficiencia energética. Para reducir el makespan, el consumo de energía tiene que ser aumentado para poder procesar las tareas más rápido. Cuando el consumo de energía es bajo las máquinas no están trabajando a la velocidad más alta, por lo que, si una incidencia aparece, la velocidad de estas máquinas puede ser aumentada con el fin de recuperar el tiempo perdido por la incidencia. Así la robustez está directamente relacionada con el consumo de energía. La robustez también está directamente relacionada con el makespan porque cuando aumenta el makespan hay más holgura en la solución, y a veces las incidencias pueden ser absorbidas por los buffers naturales que se generan. Es por ello que nuevas técnicas pueden ser desarrolladas para encontrar soluciones robustas y, al mismo tiempo, garantizan soluciones eficientes energéticamente. Por lo tanto, en scheduling la solución robusta obtenida puede ser ejecutada de forma eficiente energéticamente y solo en caso de interrupciones, las máquinas involucradas se pueden acelerar para absorber las perturbaciones y el resto de las tareas se ejecutan siguiendo el schedule programado.

Capítulo 7

Rescheduling en JSMS

7.1 Introducción

En los problemas de scheduling puede haber muchas interrupciones/incidencias cada día (averías de la máquina, modificaciones en el orden de tareas, eventos perturbadores, cancelaciones de pedidos, etc.). Después de una incidencia, el schedule original puede no ser válido para las nuevas condiciones. En algunos casos es posible modificar fácilmente la solución para absorber las interrupciones como se ha mostrado en el capítulo 5, pero en muchos casos es obligatorio replanificar para minimizar los efectos de dicha interrupción y recuperar la solución original tan pronto como sea posible. En Goodall, Rosamond y Harding 2014 un grupo de técnicas son comparadas para evaluar la viabilidad de replanificar.

En la literatura hay muchos métodos para tratar el scheduling en línea. En Hall y Potts 2004 trabajan con problemas de scheduling donde originalmente se planifica un conjunto de trabajos para minimizar el coste, pero en cualquier instante puede llegar un nuevo conjunto de trabajos. Los nuevos trabajos deben de ser insertados en el schedule original pero sin alterarlo en exceso. En Qi, Bard y Yu 2006 se propone el problema de la actualización de una máquina cuando se produce una interrupción después de haber procesado un subconjunto de trabajos. En Vieira, Herrmann y Lin 2000 se presentan nuevos modelos analíticos que pueden predecir el funcionamiento de las estrategias de replanificación y cuantificar el equilibrio entre las diferentes medidas de rendimiento. En Arnaout y Rabadi 2008 se aborda el problema de scheduling para las máquinas paralelas no relacionadas y dependientes de los tiempos de preparación y las diferentes averías que estas máquinas puedan sufrir o los trabajos urgentes que pueden aparecer. En Tian y col. 2014 se aplica la replanificación a un problema de automoción, dos casos de estudio son presentados

para obtener los principales factores que influyen en la tecnología de la industria del automóvil. En Guide 2000 se presenta un estudio de la planificación de la producción y el control de actividades en las empresas de fabricación. En Subramaniam y Raheja 2003 se muestran complicados procesos de reparación provocados por varias interrupciones, para demostrar que la reparación se puede resolver con la aplicación de cuatro acciones básicas como son:

- Introducir tiempo de inactividad: Para interrupciones como averías de máquinas o falta de disponibilidad, la máquina afectada se declarara como inactiva durante la duración de la interrupción.
- Ajustar el tiempo: Cuando aparecen retrasos pueden provocar cambios en el tiempo de inicio y fin de las operaciones. Como consecuencia el tiempo de procesamiento de la operación se ajusta mediante la inserción de un periodo de tiempo equivalente al tiempo adicional que se prevé.
- Insertar Operación: Nuevas Operaciones pueden ser insertadas en el schedule por la llegada de un nuevo trabajo, el aplazamiento de un trabajo o la relajación de las fechas de vencimiento.
- Eliminar operación: Este mecanismo de reparación consiste en la eliminación de una operación del schedule original si ya no es necesaria. Un ejemplo sería la cancelación de un trabajo.

En los problemas de scheduling donde se tiene en cuenta la eficiencia energética, la mayor parte de la investigación existente se ha centrado en modelos estáticos. Por lo tanto, es necesario desarrollar nuevas técnicas para hacer frente al rescheduling y reducir el consumo de energía en el problema de job-shop scheduling. En esta sección se desarrolla una nueva técnica de rescheduling para recuperar la solución original reduciendo al mínimo el consumo de energía en la zona replanificada.

Este capítulo está organizado de la siguiente forma. En la sección 7.2 se presenta el concepto de recuperabilidad, y se introducen los métodos más comunes de rescheduling. En la sección 7.3 se desarrolla una técnica para realizar el rescheduling en una solución de un problema JSMS, tras sufrir una incidencia y no poder ser absorbida mediante la robustez. En la sección 7.4 se evalúa dicha técnica de rescheduling. Finalmente, presentamos las conclusiones de este capítulo.

7.2 Rescheduling y Recuperabilidad

Como se ha señalado, en los problemas reales de scheduling ocurren todos los días eventos impredecibles como hemos comentado en los capítulos anteriores, a veces, estos eventos pueden ser absorbidos por el schedule original sin la necesidad de aplicar ninguna técnica de replanificación. En este caso, el schedule se considera robusto y es capaz de absorber las interrupciones menores. Sin embargo, cuando el evento no puede ser absorbido por el schedule, un proceso de rescheduling es necesario para obtener una solución factible. En este proceso se intenta reducir al mínimo el número de tareas afectadas. Es por ello que aparece el concepto de "nervousness"que se utiliza para medir la cantidad de cambios necesarios para recuperar un schedule original. El término "nervousness"fue acuñado por Steele 1975 para ser utilizado en el contexto del sistema de planificación de los materiales (MRP). Posteriormente, fue utilizado por Pujawan 2004 para representar la propagación de los cambios en los problemas de fabricación en el plan maestro de producción (MPS) sobre la inestabilidad en los requisitos de las piezas o componentes en los niveles inferiores de la estructura del producto. En capítulos anteriores se ha definido la robustez y la estabilidad, a continuación se define el concepto de recuperabilidad:

- Un schedule es ***recuperable*** si únicamente algunas tareas consecutivas se ven afectadas y el schedule original se recupera en un punto de tiempo.

Así, si una máquina se interrumpe durante la ejecución de una tarea, solo el comienzo de algunas tareas consecutivas se ve afectado por la interrupción y el resto del schedule permanece inalterado.

La principal diferencia entre la estabilidad y recuperabilidad es que en la recuperabilidad, las tareas reparadas son consecutivas en el tiempo y distribuidas entre las máquinas. Por lo tanto, hay un punto en el tiempo (llamado punto match-up) donde se recupera el schedule original y todas las tareas posteriores a este punto de tiempo mantienen su tiempo de inicio original. Sin embargo, en la estabilidad se modifica el tiempo de inicio de las diferentes tareas repartidas a lo largo del schedule.

Los tres métodos de reparación de schedule más usados son: regeneration (Church y Uzsoy 1992, Wu, Storer y Pei-Chann 1993), partial rescheduling (Li, Shyu y Adiga 1993 Wu y Li 1995) y right-shift scheduling (Abumaizar y Svestka 1997):

- ***Regeneration*** construye un schedule completo con la replanificación de todas las tareas, también llamado rescheduling total. Esta estrategia requiere mucho esfuerzo computacional para ser ejecutada, ya que, muchas tareas pueden ser replanificadas. Produce un alto nivel de nervousness y baja estabilidad.

- ***Partial rescheduling*** tiene en cuenta únicamente las tareas que se vieron afectadas por la incidencia. Esto reduce el nervousness y aumenta la estabilidad.

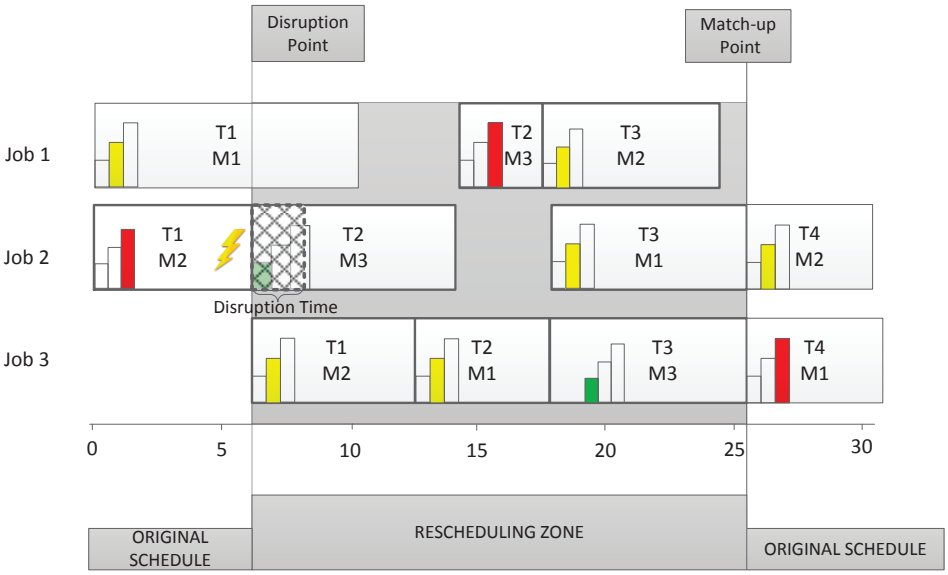


Figura 7.1: Rescheduling por recuperación

- ***The Right-Shift method*** pospone las tareas pendientes por el tiempo de inactividad. En algunos casos, Right-Shift podría ser un caso especial de Partial rescheduling. El Right-Shift produce muy poco nervousness y maximiza la estabilidad. Esta idea fue utilizada por Akturk y Gorgulu 1999 para determinar el punto match-up con el objetivo de replanificar el menor número de tareas posibles.

En este trabajo centramos nuestra atención en el rescheduling utilizando una técnica match-up para reducir el nervousness, aumentar la estabilidad y recuperar el schedule original tan pronto como sea posible. Por lo tanto, después de una incidencia, se determina un punto match-up para cada máquina y para el schedule global. La zona que se debe replanificar es la que se encuentra entre el punto de incidencia y el punto match-up (ver figura 7.1). Para ello surgen dos problemas a tratar:

1. El problema de encontrar un punto match-up en el schedule para determinar el intervalo que a replanificar (zona sombreada en la figura 7.1). Una técnica específica debe ser desarrollada para encontrar el punto match-up, que minimice la ventana temporal entre la incidencia y dicho punto.

2. El problema de buscar un nuevo schedule que minimice el consumo de energía entre el punto de interrupción y el match-up obtenido (zona de rescheduling en la figura 7.1).

Ambos problemas se deben gestionar de forma diferente. El primer problema podría ser considerado como un nuevo problema de scheduling en el que el objetivo es minimizar makespan (punto de match-up), penalizando las variables modificadas con respecto a la solución original. La solución a este problema es devolver un schedule estable o devolver que el schedule no ha sido recuperado. Por lo tanto, en primer lugar se debe desarrollar una técnica de búsqueda en amplitud para minimizar el efecto de propagación a lo largo del schedule. El segundo problema es un problema de scheduling con un umbral de makespan dado, por lo que una técnica de búsqueda metaheurística debe aplicarse para minimizar el consumo de energía. Hay que tener en cuenta que el umbral de makespan es una restricción dura en este problema y no un parámetro a optimizar, debido a que el resto del schedule original (a partir del punto de match-up) no se modifica. Este problema de replanificación podría devolver una solución que no mejorara energéticamente a la actual, por lo que la solución obtenida en el primer problema sería una solución válida.

7.3 Diseño de Algoritmo para Rescheduling en JSMS

Cuando un schedule sufre una incidencia y no puede ser absorbida mediante el concepto de robustez, se requiere aplicar una técnica de rescheduling para minimizar los cambios necesarios en el schedule original.

Nuestro objetivo principal es desarrollar una técnica para buscar un punto temporal llamado match-up donde el schedule original pueda restablecerse o ser recuperado. De este modo se reduce el coste computacional y el tiempo necesario para volver a restablecer el schedule. Al mismo tiempo, parte del schedule permanece inalterado, por lo que se aumenta la estabilidad y se reduce el nervousness.

7.3.1 Desarrollo de una Técnica Match-up

En el problema JSMS, las máquinas pueden trabajar a diferentes velocidades, con sus correspondientes consumos de energía y tiempos de procesamiento durante la ejecución de las tareas. Esta variabilidad puede ser utilizada para reducir al mínimo el punto match-up para cada máquina. Debido al hecho de que una incidencia aparece sobre una tarea aleatoria ejecutada por una máquina específica, estas incidencias se pueden propagar a otras tareas ejecutadas en otras máquinas. El objetivo principal del algoritmo propuesto es analizar la propagación de la tarea interrumpida para acelerar cada máquina implicada y, de este modo, absorber parcial o totalmente la incidencia. Una vez que la incidencia ha sido absorbida totalmente, el schedule original se recupera en un momento determinado (punto match-up). Hay que tener en cuenta que el algoritmo devuelve un punto de match-up y también un schedule válido. Sin embargo, se puede observar que las máquinas que intervienen en el proceso de rescheduling se han acelerado, por lo que la solución obtenida en el rescheduling no es una solución más eficiente desde el punto de vista energético. Por

ello posteriormente mediante un algoritmo memético se pretenderá obtener una solución que minimice el consumo de energía.

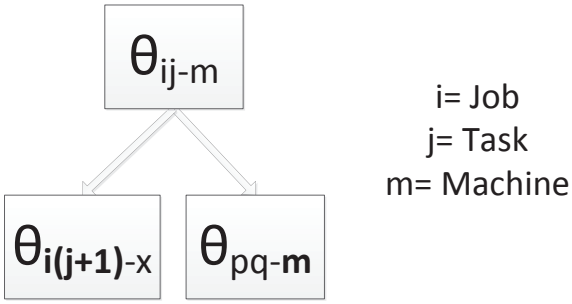


Figura 7.2: Relación entre tareas por trabajo y máquina

La figura 7.2 muestra la relación entre una tarea interrumpida θ_{ij-m} y las dos siguientes tareas relacionadas. Una tarea $\theta_{i(j+1)-x}$ relacionada por la restricción de precedencia, y una tarea θ_{pq-m} relacionada con la máquina que comparten. El algoritmo 13 muestra el pseudo-código de la técnica match-up. La entrada del algoritmo es el schedule original (Schedule) y la incidencia que se produce en una máquina durante la ejecución de la tarea (IncidentTask). La salida del algoritmo es el punto match-up del schedule (MatchUpSchedule) y un reschedule válido (Solution), si la incidencia es absorbida. Si por lo contrario no se puede absorber, el algoritmo devolverá que se ha alcanzado el makespan.

Algoritmo 13: CalcularMatch-Up(Input (Schedule,IncidentTask), Output ((MatchUpSchedule,Solution)∨(MakespanReached))

```
InvolvedTasks= φ; //La cola es inicializada a vacía
InvolvedTasks ← IncidentTask;
MakespanReached= False;
while (InvolvedTasks≠ φ and !MakespanReached) do
    CurrentTask=InvolvedTasks.pop; //Acceso al siguiente elemento de la cola
    NextTaskMach = GetNextbyMach(CurrentTask); //Seleccionar siguiente tarea por máquina
    NextTaskJob = GetNextbyJob(CurrentTask); //Seleccionar siguiente tarea por restricción de precedencia
    Check(NextTaskMach, &MakespanReached);
    Check(NextTaskJob, &MakespanReached);
end while
if (!MakespanReached) then
    MatchUpSchedule = Max(MatchUp[NMach]);
    Return (MatchUpSchedule,Solution);
else
    Return MakespanReached
end if
```

El algoritmo 13 funciona de la siguiente manera, hay una cola llamada InvolvedTasks que almacena todas las tareas afectadas por la incidencia. Estas tareas, insertadas en la cola, almacenan su propio retraso propagado por la incidencia. La primera tarea insertada en

InvolvedTasks es IncidentTask, la cual almacena el retraso de propagación en el momento inicial de la interrupción (ver figura 7.1). A continuación, se comprueba si la próxima tarea por máquina y la siguiente tarea por restricción de precedencia son capaces de absorber la incidencia (véase algoritmo 14). Si ambas tareas absorben la incidencia (tiempo de interrupción), entonces el algoritmo devuelve el schedule match-up, es decir, el final más tardío de las tareas afectadas. Sin embargo, si cualquiera de las tareas no puede absorber por completo la incidencia, esta se insertará en la cola InvolvedTask para comprobarse en las próximas iteraciones. La cantidad de tiempo que esta tarea no fue capaz de absorber se calcula y se almacena. Una vez que una tarea es capaz de absorber la incidencia propagada, se almacena el match-up para la máquina involucrada (MatchUp[NMach]). El algoritmo se ejecuta de forma iterativa hasta que no haya ninguna tarea en la cola o hasta que el makespan del schedule original haya sido alcanzado (MakespanReached = True). Si la cola InvolvedTasks está vacía, entonces la incidencia ha sido absorbida, y se obtiene un punto match-up y un reschedule válido. De lo contrario, la incidencia no ha sido absorbida en el makespan dado y se devuelve que el makespan ha sido alcanzado.

Algoritmo 14: Check(Input (Task,MakespanReached), Output (MakespanReached))

```
RecovTime=AbsorbIncidence(Task);
Update(end_time(Task));
if (RecovTime ≥ delay-propagated(Task)) then
    UpdateMatchUp(machine(Task),end_time(Task));
else
    Delay_propagated(Task)=Delay_propagated(task)-RecovTime;
    InvolvedTasks ← Task; //La tarea es añadida a la cola
    if (MakespanReached(Task)) then
        MakespanReached=True;
    end if
end if
Return MakespanReached;
```

El algoritmo 14 comprueba si una tarea es capaz de absorber el retraso remanente. Con este fin, la función AbsorbIncidence se encarga de ajustar las máquinas implicadas a la mayor velocidad para reducir al mínimo la duración de estas tareas. Por lo tanto, se calcula el tiempo recuperado (RecovTime). Si este tiempo es suficiente para absorber la incidencia, el punto match-up de la máquina implicada se actualiza con el nuevo tiempo de fin de la tarea. Si no es así, se actualiza el retraso propagado por esta tarea y la tarea se inserta en la cola. Por último, se comprueba que la tarea actualizada no ha alcanzado el makespan de la solución original.

7.3.2 Rescheduling

Una vez que se ha obtenido un punto de match-up, un nuevo sub-problema de scheduling se puede definir desde el punto de la incidencia hasta el punto de match-up. En la sección anterior se obtuvo una solución válida, por lo que nuestro objetivo en esta sección es mejorar esta solución en términos de consumo de energía.

Para ello, hemos utilizado el algoritmo memético que combina un algoritmo genético (GA) con una búsqueda local (LS) presentado en el capítulo 4. En el algoritmo 15 se representa cómo se utiliza el algoritmo memético. La idea principal es obtener una solución donde el makespan sea menor o igual que el match-up y la energía utilizada sea menor que la utilizada en el schedule obtenido en el paso anterior. Con este fin, mediante el valor de λ se controla el peso de cada objetivo. Cuando $\lambda = 0$ solo el consumo de energía se tiene en cuenta, pero cuando el valor de λ aumenta, el valor del makespan comienza a tener más peso. De modo que el valor de λ se inicializa a 0 y se va incrementando. El algoritmo se detiene cuando el peso del makespan es suficiente para obtener un schedule con un makespan igual o menor que el umbral establecido en la solución del paso anterior. Si el schedule ha conseguido reducir la energía utilizada, este será devuelto.

Algoritmo 15: Rescheduling (Sub-problem, Schedule, MatchUp)

```
lambda = 0;
Makespan=MatchUp + 1;
while (lambda <=1 and Makespan>MatchUp) do
    ScheduleImp=MemeticAlgorithm (Sub-problem, λ)
    Makespan=ScheduleImp.Mk;
    lambda = lambda + 0.1;
end while
if ((ScheduleImp.Mk<=Schedule.Mk) and (ScheduleImp.En<Schedule.En)) then
    Return ScheduleImp;
else
    Return Schedule;
end if
```

Ejemplo

La figura 7.3 muestra un ejemplo de cómo funciona el algoritmo. En esta figura se representa un problema de scheduling con 3 trabajos, 3 máquinas y 3 tareas por trabajo. La figura 7.3a muestra un schedule optimizado en términos de consumo de energía y makespan. Las tareas en verde fueron ejecutadas a baja velocidad, por lo que requieren un gran tiempo de procesamiento; las tareas en amarillo fueron ejecutadas a velocidad media, por lo que requieren un tiempo de procesamiento medio; y las tareas en rojo fueron ejecutadas a alta velocidad, por lo que requieren un tiempo de procesamiento bajo. Dada una interrupción, el algoritmo match-up busca una solución mediante el aumento de la velocidad de algunas máquinas y el uso de los buffers existentes (véase la figura 7.3b). El punto match-up se calcula para cada máquina, y seleccionando el máximo se obtiene el punto

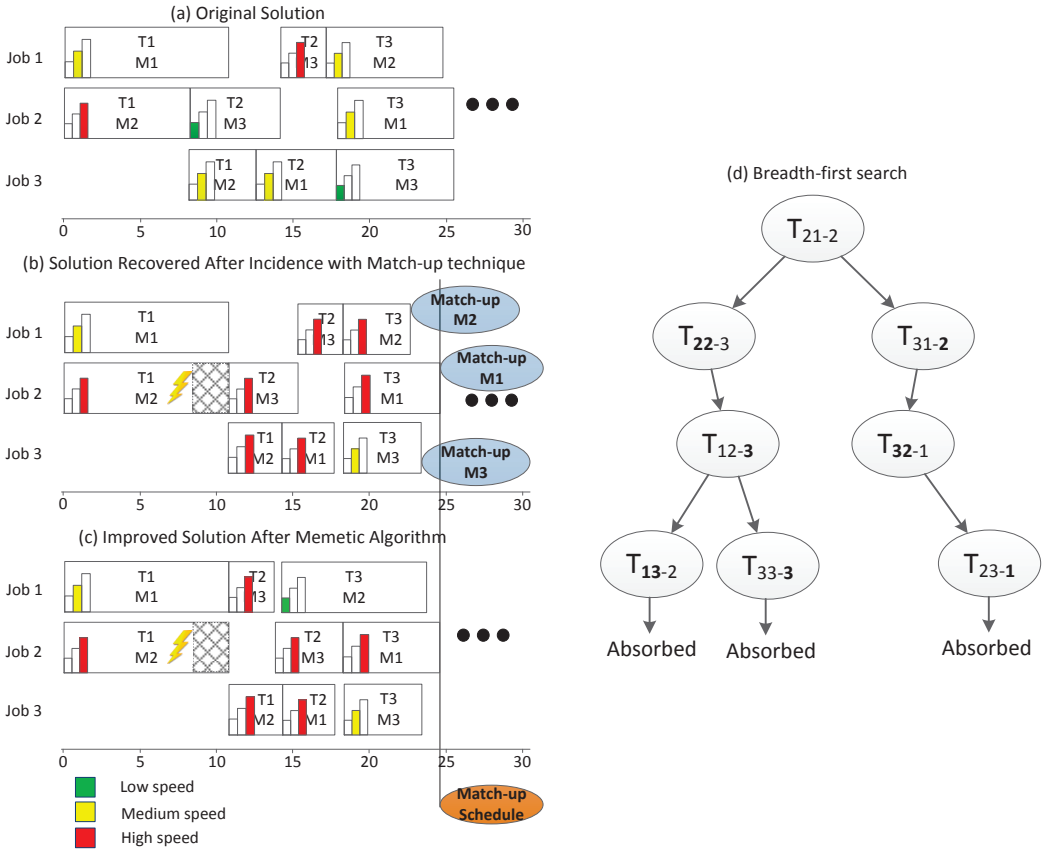


Figura 7.3: Solución original, solución recuperada, solución mejorada después de una incidencia y una búsqueda en anchura

match-up para el schedule. Además, el schedule original se recupera obteniendo una solución válida. Para ello, algunas de las tareas involucradas tuvieron que ser ejecutadas a mayor velocidad con el fin de reducir el punto match-up.

Por último, el algoritmo memético realiza mejoras entre el punto de interrupción y el punto de match-up obtenido (figura 7.3c). Se puede observar que la tarea 2 del trabajo 1 es ejecutada antes que la tarea 2 del trabajo 2 (ambas utilizando la máquina 3), por lo que la tarea 3 del trabajo 1 es ejecutada a la velocidad más baja y por lo tanto su tiempo de procesamiento se incrementa y el consumo de energía se reduce. Sin embargo, estos cambios no modifican el resto del schedule. La figura 7.3d muestra el árbol de búsqueda llevado a cabo por la técnica de match-up.

7.4 Evaluación

En esta sección se muestran los resultados obtenidos de llevar a cabo una evaluación de las técnicas propuestas. En primer lugar, se han simulado incidencias sobre las soluciones originales y se ha medido la robustez de estas. Cuando las máquinas que sufren las incidencias no pueden absorberlas (por robustez), se aplica el método de rescheduling propuesto. Estas incidencias se analizan con nuestra técnica match-up para determinar el intervalo temporal del schedule que tiene que ser replanificado (sub-problema). Así, el sub-problema se resuelve con el algoritmo memético propuesto para obtener un schedule donde se mejore la eficiencia energética. Para realizar esta evaluación se ha usado el benchmark de Lawrence (Lawrence 1984), donde se disponen de 5 instancias para cada tipo de problema.

7.4.1 Incidencias y Robustez

Una vez obtenida una solución para cada instancia, se simularon incidencias para medir la robustez de las soluciones. Por lo tanto, para cada solución, se generaron 100 incidencias (500 para cada grupo de instancias). Una vez que una incidencia ha sido asignada a una máquina, durante la ejecución de una tarea, se le asigna una duración de incidencia de forma aleatoria entre el 1 % y 30 % del tiempo de procesamiento máximo de cada tipo de instancia.

La figura 7.4 muestra que todas las instancias mantienen un comportamiento similar ante las incidencias. Cuando los valores de λ son bajos, el objetivo se centra principalmente en la minimización de la energía. Por lo tanto el makespan es mayor y no existen muchos buffers entre las tareas consecutivas. En este caso, muchas incidencias pueden ser absorbidas sin rescheduling (schedules robustos). Hay que tener en cuenta que para los casos con menos tareas (10x5, 15x5, 20x5), la robustez es menor (sobre todo en los valores altos de λ) porque hay menos variabilidad en la asignación de las tareas, por lo que el número de buffers es menor.

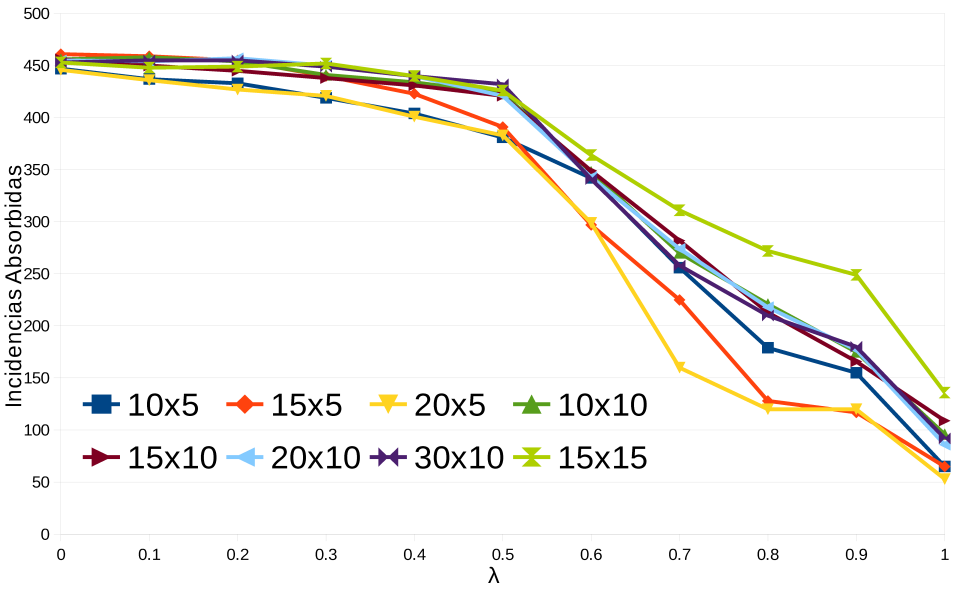


Figura 7.4: Incidencias absorbidas para los distintos tipos de instancias

7.4.2 Evaluando la técnica Match-up

Todas las incidencias que no fueron absorbidas por la propia robustez del schedule deben ser tratadas por el proceso de rescheduling. Para ello, se ha aplicado nuestra técnica match-up para recuperar en la mayor brevedad posible el schedule original. Con este fin, teniendo en cuenta una incidencia, el algoritmo busca el mejor punto de match-up y devuelve una solución o devuelve que se ha alcanzado el makespan del schedule original. Este último caso se produce cuando la incidencia se encuentra al final del schedule, por lo que no hay tiempo suficiente para recuperar la solución. La figura 7.5 muestra el tiempo de ejecución de la técnica match-up para diferentes grupos de instancias. Se representa el tiempo de ejecución promedio en milisegundos para resolver cada tipo de instancias sobre un procesador Intel Core 2 Duo con 1Gb de memoria RAM.

Las tablas 7.1 y 7.2 muestran, para cada grupo de instancias, la cantidad de incidencias (de un total de 500) que fueron absorbidas por la robustez (Rb) por nuestra técnica de match-up (MUP) y las incidencias que no se pudieron recuperar porque se alcanzó el makespan (Mk). A la luz de los resultados se puede observar la importancia del valor de λ. Dado que para los valores bajos de λ, el objetivo es reducir al mínimo el consumo de energía, por lo que, la mayoría de las máquinas funcionan a baja velocidad y como consecuencia el makespan aumenta. Esto provoca que muchos buffers aparezcan a lo largo del schedule y la mayoría de las incidencias sean absorbidas por el uso de estos buffers (soluciones robustas), por lo que no se necesita rescheduling. Sin embargo, se

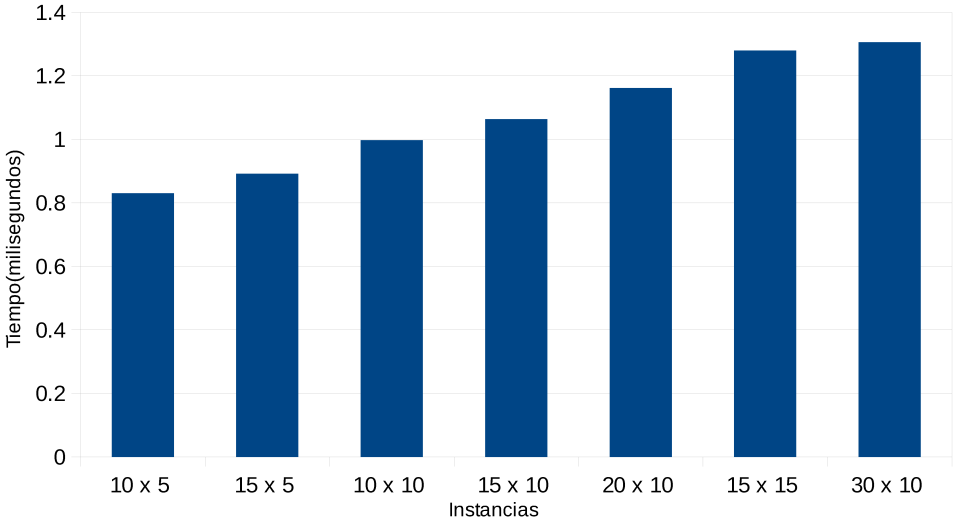


Figura 7.5: Tiempo de ejecución medio de la técnica Match-up para las diferentes instancias

Tabla 7.1: Número de incidencias absorbidas por robustez (Rb), por punto match-up (MUp) o solución no recuperada (Mk) para diferentes instancias

	10x5			15x5			20x5			10x10		
λ	Rb	MUp	Mk	Rb	MUp	Mk	Rb	MUp	Mk	Rb	MUp	Mk
0	447	40	13	461	35	4	446	53	1	456	27	17
0.1	437	55	8	459	40	1	436	61	3	458	24	18
0.2	433	56	11	455	45	0	427	66	7	454	33	13
0.3	419	67	14	440	56	4	421	72	7	441	42	17
0.4	404	70	26	423	71	6	401	95	4	434	48	18
0.5	381	95	24	391	100	9	383	112	5	423	57	20
0.6	342	113	45	297	170	33	299	169	32	348	97	55
0.7	256	165	79	225	217	58	160	252	88	270	128	102
0.8	179	170	151	128	206	166	120	204	176	221	120	159
0.9	155	137	208	117	209	174	120	199	181	175	83	242
1	65	141	294	65	235	200	53	177	270	96	109	295

Tabla 7.2: Número de incidencias absorbidas por robustez (Rb), por punto match-up (MUp) o solución no recuperada (Mk) para diferentes instancias

	15x10			20x10			30x10			15x15		
λ	Rb	MUp	Mk	Rb	MUp	Mk	Rb	MUp	Mk	Rb	MUp	Mk
0	456	38	6	455	38	7	453	45	2	453	28	19
0.1	450	42	8	454	44	2	455	45	0	448	27	25
0.2	445	46	9	457	40	3	455	44	1	449	29	22
0.3	438	54	8	450	48	2	449	50	1	452	29	19
0.4	431	64	5	440	49	11	440	55	5	440	43	17
0.5	421	63	16	421	79	0	432	63	5	426	49	25
0.6	349	114	37	343	140	17	341	146	13	364	78	58
0.7	282	177	41	274	193	33	258	214	28	311	119	70
0.8	213	222	65	218	230	52	210	253	37	272	145	83
0.9	166	196	138	177	237	86	180	271	49	249	128	123
1	109	171	220	86	180	234	92	234	174	136	138	226

puede observar en las tablas 7.1 y 7.2 que a medida que los valores λ aumentan, el número de incidencias absorbidas por la robustez (Rb) disminuye, mientras que, el número de incidencias absorbidas por nuestra técnica de match-up (MUP) aumenta. Cabe destacar que los valores más altos de MUP se alcanzan para los valores de λ igual a 0.7, 0.8 y 0.9, cuando el objetivo está centrado principalmente en minimizar el makespan, reduciéndose el número de buffers en el schedule original para absorber las incidencias por robustez, por lo que nuestra técnica match-up debe recuperar más schedules. Hay que tener en cuenta que para $\lambda = 1$, el objetivo solo se centra en minimizar el makespan, por lo que todas las máquinas están trabajando a la velocidad más alta y es menos probable recuperar una solución porque se alcanza el final del schedule sin encontrar prácticamente buffers.

7.4.3 Evaluando el algoritmo memético para rescheduling

En esta evaluación consideramos que la técnica de match-up ya ha sido aplicada, y por lo tanto se ha alcanzado un punto de match-up, obteniendo un schedule recuperado inicial. Por lo tanto, el objetivo del algoritmo memético es mejorar dicha solución obtenida en términos de eficiencia energética.

La tabla 7.3 muestra los resultados para las instancias 15x10 y 30x10. La columna (MUp) representa el número total de casos recuperados por la técnica de match-up, las columnas (Rec) y (% Rec) representan el número y el porcentaje de casos recuperados donde el algoritmo memético fue capaz de reducir el consumo de energía, respectivamente. Por último, las columnas EnRed y %EnRed representan la cantidad de energía y el porcentaje de energía que el algoritmo memético fue capaz de reducir en el schedule obtenido con respecto al schedule recuperado por la técnica de match-up.

Tabla 7.3: Replanificando los resultados obtenidos por la técnica de Match-up

	15x10					30x10				
λ	MUp	Rec	%Rec	EnRed	%EnRed	MUp	Rec	%Rec	EnRed	%EnRed
0	38	27	71.1 %	3960	35.62 %	45	30	66.7 %	5318	43.82 %
0.1	42	34	81.0 %	6512	39.05 %	45	38	84.4 %	5756	46.12 %
0.2	46	31	67.4 %	6050	41.74 %	44	31	70.5 %	7061	47.14 %
0.3	54	36	66.7 %	6672	38.89 %	50	32	64.0 %	6308	43.35 %
0.4	64	52	81.3 %	11599	43.48 %	55	40	72.7 %	7549	44.95 %
0.5	63	49	77.8 %	9472	39.27 %	63	46	73.0 %	10281	44.07 %
0.6	114	94	82.5 %	18413	30.11 %	146	105	71.9 %	18374	22.36 %
0.7	177	119	67.2 %	22903	19.92 %	214	103	48.1 %	16272	7.23 %
0.8	222	102	45.9 %	21782	7.78 %	253	86	34.0 %	15260	4.93 %
0.9	196	64	32.7 %	87961	25.36 %	271	55	20.3 %	24391	5.04 %
1	171	114	66.7 %	187027	74.52 %	234	177	75.6 %	739241	86.08 %

Se puede observar en la tabla 7.3 que el algoritmo memético fue capaz de reducir el consumo de energía en un número significativo de incidencias. Como se anticipó en la sección 7.4.2, para los valores bajos de λ, el número de veces que el algoritmo memético fue capaz de reducir el consumo de energía no varía significativamente (alrededor de 40). Sin embargo, cuando λ es igual a 0.6 y 0.7, el número de casos recuperados (Rec y % Rec) aumentan, y cuando λ es igual a 0.8 y 0.9 estos valores disminuyen. Este comportamiento es debido a las características del schedule original. Para los valores altos de λ, los valores MUp son altos porque la robustez es baja, de modo que hay más opciones para el rescheduling. Se han llevado a cabo más procesos de replanificación cuando λ es igual a 0.6 y 0.7, lo que demuestra que, el algoritmo memético es capaz de reducir el consumo de energía en un alto porcentaje de casos. Sin embargo, cuando λ es igual a 0.8 y 0.9 se reduce este porcentaje, ya que la función objetivo se centra principalmente en la minimización del makespan. Por lo tanto, la mayoría de las máquinas están trabajando a la velocidad más alta siendo muy complicado reducir la energía utilizada. Por último, cuando $\lambda = 1$, se considera un caso especial porque en la función objetivo solo se tiene en cuenta el makespan, por lo que muchas de las tareas que no están involucradas en la ruta crítica se pueden ejecutar a la velocidad más baja sin empeorar el makespan. Este comportamiento se puede ver representado en la figura 7.6 con la curva de tendencia (Rec).

La tabla 7.3 también muestra el ahorro total de energía tras aplicar la reducción de energía realizada por el algoritmo memético. Los valores de energía ahorrada (EnRed) aumentan cuando λ aumenta, debido a que para los valores altos de λ la energía utilizada es mayor, por lo que, es posible reducir más cantidad de energía. La columna (% EnRed) representa el porcentaje de energía ahorrada. Se puede observar que para los valores bajos de λ, el porcentaje de reducción de energía no varía significativamente. Sin embargo, cuando λ es igual a 0.6, 0.7 y 0.8, el porcentaje disminuye y cuando λ es igual a 0.9, el porcentaje aumenta de nuevo. Este comportamiento también se relaciona con la característica del schedule original porque cuando λ es igual a 0.6 y 0.7 el algoritmo memético es capaz de reducir el consumo de energía en un alto porcentaje de casos, pero no en gran cantidad.

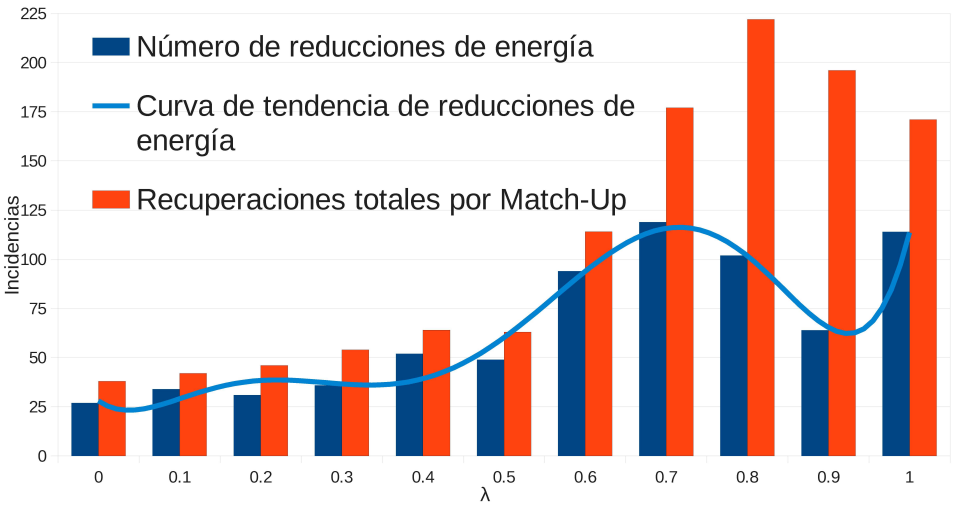


Figura 7.6: Comparativa entre MUp y Rec para las instancias 10x5

Para estos valores de λ, la energía utilizada no es demasiado alta. Por lo contrario para λ igual a 0.9, la técnica es capaz de ahorrar más cantidad de energía.

7.5 Conclusiones

En el sector industrial se enfrentan a un gran número de problemas de scheduling. La mayoría de estos problemas son dinámicos, y los recursos utilizados para realizar las distintas tareas pueden trabajar a distintos ratios. En los problemas de la vida real también es habitual enfrentarse con incidencias por lo que son necesarias técnicas de recuperación para restablecer el plan original en la mayor brevedad posible. En este capítulo, hemos propuesto dos técnicas diferentes para gestionar el rescheduling sobre una versión extendida del problema job-shop scheduling. Así dada una incidencia, se aplica una primera técnica, denominada técnica match-up, para determinar el punto del schedule donde se puede recuperar la solución original, y por lo tanto, obtener una solución válida tras la incidencia. A continuación, se propone un algoritmo memético para buscar una solución eficiente energéticamente en la zona de rescheduling establecida. Un extenso estudio se ha llevado a cabo para analizar el comportamiento de las técnicas propuestas. Con este fin, se simularon algunas incidencias sobre soluciones a instancias conocidas en la literatura pero adaptadas a nuestro problema. La técnica match-up propuesta mantiene un buen rendimiento recuperando en muchos casos la solución. Finalmente, la mayoría de las soluciones replanificadas se mejoraron para ahorrar más consumo de energía gracias al algoritmo memético.

Capítulo 8

Conclusiones y Trabajos Futuros

8.1 Conclusions

The main aim of this thesis is to develop several techniques to get solution in scheduling problems taking into account optimality, robustness and energy-efficiency. At the beginning of this work, the job shop scheduling problem has been analysed and different methods to solve it have been presented.

The classical job shop scheduling problem can be extended to other problems, in order to represent possible real life situations. In this case, we have used the problem $JSO(n, p)$, where each operation has to be assisted by one operator from a limited set of them. To solve this problem, we have developed a three steps technique with the aim of obtaining optimized and robust solutions. In the first step, an optimized solution has been obtained to optimize the makespan for the classical JSP. In the second step, this solution has been modified to take into account operators constraints to minimize the makespan and to maximize the number of buffers. Finally, in the third step, setting the previous makespan, the robustness has been maximized by redistributing buffers. In this way, given the solutions for the JSP, this procedure provides schedules, taking into account operators, in short time. These solutions are obtained according to client needs, maintaining their desired trade-off between optimality and robustness. The obtained solutions have been compared to the solutions obtained by the IBM ILOG CPLEX CP Optimizer tool (CP Optimizer) and the comparative study shows that losing some optimality, robust solutions are able to absorb more incidences.

In the traditional job shop scheduling problem different optimization objectives have been considered, such as processing time, cost and quality. However, energy consumption has

been treated recently. We have extended the classical JSP into a job-shop scheduling problem where machines can work at different speeds (JSMS). To solve the JSMS problem, a memetic algorithm has been developed and the results have been compared with the commercial tool CP Optimizer to compare the behaviour of both solvers. The evaluation shows that our algorithm has a similar behaviour than CP Optimizer for small instances. However, for large instances, our algorithm overcomes CP Optimizer. This is due to the fact that our algorithm is an adaptive method able to get solutions faster when the size of the instances is large and the time is limited. Our memetic algorithm is competitive to solve realistic instances of large scale scheduling problems.

As it has been described in the previous chapters, robustness can be related with many industrial processes. In industry, robustness can be profitable, because even if a perturbation appears in the schedule, it can remain valid, being not necessary to reschedule and therefore to interrupt the production process. We have developed a dual model to solve the JSMS problem in order to relate optimality criteria with energy consumption and robustness/stability. Our model is committed to protect dynamic tasks against further incidences so as to obtain robust and energy-aware solutions. The proposed dual model has been evaluated with our memetic algorithm to compare the behaviour against the original model. The evaluation shows that the dual model is able to obtain more robust solutions in all instances and more energy efficient solutions in most of them. This is due to the fact that the assigned buffer times to some tasks are used to protect them against incidences. Furthermore, if during execution, there is no incidence, the involved machine can use this buffer to work at lower speed, reducing the energy consumption. The combination of robustness and stability gives the proposal an added value because, although an incidence cannot be directly absorbed by the disrupted task, it can be repaired by involving only a small number of tasks. Thus, the original solution can be recovered in order to maintain feasibility along the rest of the previously obtained schedule.

In this work, it is shown the close relationship between Makespan, Energy consumption and Robustness. However, this relationship has not been analysed in the literature. We have analysed the relationship among these objectives and analytical formulas have been proposed to estimate the relationship between them. The results show the direct relationship between robustness and energy-efficiency; the trade-off between makespan and robustness and the direct relationship between robustness and makespan. As it has been explained, to reduce the makespan, the energy used has to be increased to process the tasks faster. Otherwise, when the energy used is low, it is because the speed of the machines is not the highest. Thus if an incidence appears, it can be increased in order to recover the time of the incidence. When the makespan is bigger, there are more gaps in the schedule, so the possible incidences can be absorbed by the buffers.

In some cases, when a disruption appears, it is possible to modify the solution to absorb the disruption. But in many cases, rescheduling is compulsory to minimize the effects of the disruption and recover the original solution as soon as possible. To this end, we propo-

se a technique to calculate the match-up point, so as to minimize the rescheduling zone. Then, a memetic algorithm for finding a schedule that minimizes the energy consumption within the rescheduling zone is proposed. The proposed match-up technique maintained a good performance and most of the instances were recovered in an efficient way. Finally, most of the rescheduling solutions were improved to save more energy consumption.

When we are working with real time problems and rescheduling is needed, the time needed to recover the solution is critical. Thus, efficient algorithms that re-establish the solution as soon as possible are mandatory. For this reason, in future works we have planned to work in rescheduling techniques to obtain solutions more efficiently and techniques that recover the solution as soon as possible to reduce changes in the original schedule. We are interested in other problems where energy efficiency is taken into account in order to represent more realistic problems.

8.2 Publications List

In this section, all publications related to this thesis are listed. They are classified in two groups: International journals listed in the Journal Citation Reports (JCR[1]) and international conferences listed in the Conference Ranking Exercise (CORE[2]).

Journals:

- *A Genetic Algorithm for Energy-Efficiency in Job-Shop Scheduling.* Miguel A. Salido, Joan Escamilla, Adriana Giret, Federico Barber. **The International Journal of Advanced Manufacturing Technology**. pp: 1-12, 2015. (JCR:1.458)
- *A Dual Scheduling Model for Optimizing Robustness and Energy Consumption in Manufacturing Systems.* **Journal of engineering manufacture**. Joan Escamilla, Miguel A. Salido. pp: 1-17, (2016). (JCR:0.954)
- *Energy efficiency, robustness, and makespan optimality in job-shop scheduling problems.* MA Salido, J Escamilla, F Barber, A Giret, D Tang, M Dai. **Artificial Intelligence for Engineering Design, Analysis and Manufacturing**, pp: 1-13, 2015. (JCR:0.604)
- *Solving the job shop scheduling problem with operators by depth-first heuristic search enhanced with global pruning rules.* Carlos Mencía, María R. Sierra, Miguel A. Salido, Joan Escamilla, Ramiro Varela. **AI Communications**. 28(2): 365-381 (2015). (JCR:0.547)

[1] Journal Citation Reports http://thomsonreuters.com/journal-citation-reports/
[2] Conference Ranking Exercise http://http://core.edu.au/index.php/conference-rankings

- *Metaheuristic Technique for Energy-Efficiency in Job-Shop Scheduling.* Joan Escamilla, Miguel A. Salido, Adriana Giret and Federico Barber. **Knowledge Engineering Review**, (Accepted), 2015. (JCR:0.766)
- *Rescheduling an Extension of Job-shop Problem with Match-up Scheduling Technique.* Joan Escamilla, Miguel A. Salido. **Journal of Cleaner Production**. (Under Review) (2015). (JCR:3.844)
- *A Hybrid Metaheuristic to optimize energy consumption in job shop Scheduling Problem with speed scaling.* Joan Escamilla, Miguel A. Salido. **Journal of Manufacturing Science and Engineering** (Under Review) (2015). (JCR:1.022)

Conferences:

- *Robust Solutions to Job-Shop Scheduling Problems with Operators.* Joan Escamilla, Mario Rodriguez-Molins, Miguel A Salido, Maria R Sierra, Carlos Mencia, Federico Barber. IEEE 24th International Conference on Tools with Artificial Intelligence (**ICTAI**), pp: 299-306, 2012. (CORE B)
- *Combining global pruning rules with depth-first search for the job shop scheduling problem with operators.* Mencía, C., Sierra, M. R., Salido, M. A., Escamilla, J., & Varela, R. (2012). In Proceedings of the 19th **RCRA** International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion.
- *Energy-aware parameters in job-shop scheduling problems.* Salido, M. A., Escamilla, J., Barber, F., Giret, A., Tang, D., & Dai, M. (2013). In **GREEN-COPLAS 2013**: IJCAI 2013 Workshop on Constraint Reasoning, Planning and Scheduling Problems for a Sustainable Future (pp. 44-53). (CORE B)
- *A Metaheuristic Technique for Energy-Efficiency in Job-Shop Scheduling.* Escamilla, J., Salido, M. A., Giret, A., & Barber, F. (2014). In (**COPLAS 2104**) Constraint Satisfaction Techniques for Planning and Scheduling, 42. (CORE B)

Referencias

Abbasi, B., S. Shadrokh y J. Arkat (2006). «Bi-objective resource-constrained project scheduling with robustness and makespan criteria». En: *Applied mathematics and computation* 180.1, págs. 146-152.

Abumaizar, R. J. y J. A. Svestka (1997). «Rescheduling job shops under random disruptions». En: *International Journal of Production Research* 35.7, págs. 2065-2082.

Adams, J., E. Balas y D. Zawack (1988). «The shifting bottleneck procedure for job shop scheduling». En: *Management Science* 34.3, págs. 391-401.

Administration, U. E. I. (2013). «International Energy Outlook 2013». En: *Citeseer* Report Number: DOE/EIA-0484(2013).

Agnetis, A. y col. (2011). «A job-shop problem with one additional resource type». En: *Journal of Scheduling* 14.3, págs. 225-237.

Ajanovski, V. V. (2013). «Integration of a course enrolment and class timetable scheduling in a student information system». En: *International Journal of Database Management Systems* 5.1, pág. 85.

Akturk, M. S. y E. Gorgulu (1999). «Match-up scheduling under a machine breakdown». En: *European journal of operational research* 112.1, págs. 81-97.

Applegate, D. y col. (1994). «The traveling salesman problem». En: *Technical report, DIMACS, Rutgers University, New Brunswick, NJ*, págs. 1-19.

Arnaout, J.-P. y G. Rabadi (2008). «Rescheduling of unrelated parallel machines under machine breakdowns». En: *International Journal of Applied Management Science* 1.1, págs. 75-89.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. John Wiley & Sons.

Balas, E. (1967). «Discrete programming by the filter method». En: *Operations Research* 15.5, págs. 915-957.

Barber, F. y M. A. Salido (2014). «Robustness, stability, recoverability, and reliability in constraint satisfaction problems». En: *Knowledge and Information Systems*, págs. 1-16.

Beasley, D., R. Martin y D. Bull (1993). «An overview of genetic algorithms: Part 1. Fundamentals». En: *University computing* 15, págs. 58-58.

Beasley, J. (1993). «Modern heuristic techniques for combinatorial problems». En: ed. por C. Reeves. Halsted Press. Cap. Lagrangean relaxation, págs. 243-303.

Beck, J. (1994). «A schema for constraint relaxation with instantiations for partial constraint satisfaction and schedule optimization». Tesis doct. University of Toronto.

Bellman, R. (1957). *Dynamic Programming*. Princeton, New Jersey: Princeton University Press.

Bierwirth, C. (1995). «A Generalized Permutation Approach to Jobshop Scheduling with Genetic Algorithms». En: *OR Spectrum* 17, págs. 87-92.

Billaut, J., A. Moukrim y E. Sanlaville (2008). «Flexibility and Robustness in Scheduling». En: Wiley.

Binato, S. y col. (2002). «A GRASP for job shop scheduling». En: *Essays and surveys in metaheuristics*. Springer, págs. 59-79.

Bitner, J. y E. Reingold (1975). «Backtrack programming techniques». En: *Communications of the ACM* 18.11, págs. 651-656.

Blazewicz, J. y col. (1986). «Scheduling under resource constraints-deterministic models». En: *Annals of Operations Research* 7, págs. 1-356.

BMWi (2009). «German Federal Ministry of Economics and Technology: Energy Statistics». En:

Brah, S. A. y J. L. Hunsucker (1991). «Branch and bound algorithm for the flow shop with multiple processors». En: *European Journal of Operational Research* 51.1, págs. 88-99.

Brucker, P., J. Hurink y col. (1997). «A branch & bound algorithm for the open-shop problem». En: *Discrete Applied Mathematics* 76.1, págs. 43-59.

Brucker, P., B. Jurisch y B. Sievers (1994). «A branch and bound algorithm for the job-shop scheduling problem». En: *Discrete applied mathematics* 49.1, págs. 107-127.

Bruzzone, A. y col. (2012). «Energy-aware scheduling for improving manufacturing process sustainability: a mathematical model for flexible flow shops». En: *CIRP Annals-Manufacturing Technology* 61.1, págs. 459-462.

Cambazard, H. y col. (2005). «Interactively solving school timetabling problems using extensions of constraint programming». En: *Practice and Theory of Automated Timetabling V*. Springer, págs. 190-207.

Caprara, A. y M. Fischetti (1997). «Annotated Bibliographies in Combinatorial Optimization». En: ed. por F. M. In M. Dell'Amico y S. Martello. John Wiley & Sons. Cap. Branch and Cut Algorithms, págs. 45-64.

Chang-tian, Y. e Y. Jiong (2012). «Energy-aware genetic algorithms for task scheduling in cloud computing». En: *ChinaGrid Annual Conference (ChinaGrid), 2012 Seventh*. IEEE, págs. 43-48.

Cheng, Y. y col. (2013). «Energy-aware resource service scheduling based on utility evaluation in cloud manufacturing system». En: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, pág. 0954405413492966.

Church, L. K. y R. Uzsoy (1992). «Analysis of periodic and event-driven rescheduling policies in dynamic shops». En: *International Journal of Computer Integrated Manufacturing* 5.3, págs. 153-163.

Chvátal, E. (1973). «Edmonds polytopes and a hierarchy of combinatorial problems». En: *Discrete Mathematics* 4, págs. 305-337.

Dahmus, J. y T. Gutowski (2004). «An environmental analysis of machining». En: *ASME International Mechanical Engineering Congress and RD&D Exposition, Anaheim, California, USA*.

Dasgupta, S., C. Papadimitriou y U. Vazirani (2006). *Algorithms*. McGraw Hill.

Davis, L. (1985). «Job shop scheduling with genetic algorithms». En: *Proceedings of an international conference on genetic algorithms and their applications*. Vol. 140. Carnegie-Mellon University Pittsburgh, PA.

Davis, W. J. y A. T. Jones (1988). «A real-time production scheduler for a stochastic manufacturing environment». En: *International Journal of Computer Integrated Manufacturing* 1.2, págs. 101-112.

Dechter, R. (2003). *Constraint Processing*. Ed. por E. S. (USA). Morgan Kaufmann.

Della Croce, F., R. Tadei y G. Volta (1995). «A genetic algorithm for the job shop problem». En: *Computers & Operations Research* 22.1, págs. 15-24.

Dorndorf, U., E. Pesch y T. Phan-Huy (2000). «Constraint propagation techniques for the disjunctive scheduling problem». En: *Artificial intelligence* 122.1, págs. 189-240.

Duflou, J. R. y col. (2012). «Towards energy and resource efficient manufacturing: A processes and systems approach». En: *CIRP Annals-Manufacturing Technology* 61.2, págs. 587-609.

EIA, U. (2013). «Annual energy outlook 2013». En: *US Energy Information Administration, Washington, DC*.

Energy, U. D. of (2015). «Energy Information Administration, Monthly Energy Review». En:

Escamilla, J., M. Rodriguez-Molins y col. (2012). «Robust solutions to job-shop scheduling problems with operators». En: *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*. Vol. 1. IEEE, págs. 299-306.

Escamilla, J. y M. A. Salido (2016). «A Dual Scheduling Model for Optimizing Robustness and Energy Consumption in Manufacturing Systems». En: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, págs. 1-17.

Escamilla, J., M. A. Salido y col. (2014). «A Metaheuristic Technique for Energy-Efficiency in Job-Shop Scheduling». En: *Workshop on Constraint Satisfaction Techniques (COPLAS) 2014: 24th International Conference on Automated Planning and Scheduling (ICAPS)*, págs. 42-50.

Fang, K. y col. (2011). «A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction». En: *Journal of Manufacturing Systems* 30.4, págs. 234-240.

Feo, T. y M. Resende (1995). «Greedy randomized adaptive search procedures». En: *Journal of Global Optimization* 6.2, págs. 109-133.

Fox, M. S. y N. M. Sadeh (1990). «Why is Scheduling Difficult? A CSP Perspective.» En: *ECAI*, págs. 754-767.

Frost, D. y R. Dechter (1994). «Dead-end driven learning». En: *Proceedings of the National Conference on Artificial Intelligence*. JOHN WILEY & SONS LTD, págs. 294-294.

Garey, M., D. Johnson y R. Sethi (1976). «The complexity of flowshop and job shop scheduling». En: *Mathematical Operation Research* 1, págs. 117-129.

Garrido, A. y col. (2000). «Heuristic methods for solving job-shop scheduling problems». En: *ECAI-2000 Workshop on New Results in Planning, Scheduling and Design, Berlín*, págs. 36-43.

Gaschig, J. (1979). *Performance measurement and analysis of certain search algorithms*. Inf. téc. DTIC Document.

Gaschnig, J. (1977). «A general backtrack algorithm that eliminates most redundant tests». En: *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, págs. 457-457.

Glover, F. y M. Laguna (1999). «Tabu Search». English. En: *Handbook of Combinatorial Optimization*. Ed. por D.-Z. Du y P. Pardalos. Springer US, págs. 2093-2229.

Goldberg, D. (1985). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Gomory, R. (1963). «Recent Advances in Mathematical Programming». En: ed. por R. Graves y P. Wolfe. McGraw-Hill, New York. Cap. An algorithm for integer solutions to linear programs, págs. 269-302.

Gonzalez, T. y S. Sahni (1976). «Open shop scheduling to minimize finish time». En: *Journal of the ACM (JACM)* 23.4, págs. 665-679.

Goodall, P., E. Rosamond y J. Harding (2014). «A review of the state of the art in tools and techniques used to evaluate remanufacturing feasibility». En: *Journal of Cleaner Production* 81, págs. 1-15.

Goren, S. e I. Sabuncuoglu (2008). «Robustness and stability measures for scheduling: single-machine environment». En: *IIE Transactions* 40.1, págs. 66-83.

Graham, R. L. (1966). «Bounds for certain multiprocessing anomalies». En: *Bell System Technical Journal* 45.9, págs. 1563-1581.

Grimes, D. y E. Hebrard (2015). «Solving variants of the job shop scheduling problem through conflict-directed search». En: *INFORMS Journal on Computing* 27.2, págs. 268-284.

Guide, V. D. R. (2000). «Production planning and control for remanufacturing: industry practice and research needs». En: *Journal of Operations Management* 18.4, págs. 467-483.

Hall, N. G. y C. N. Potts (2004). «Rescheduling for new orders». En: *Operations Research* 52.3, págs. 440-453.

Haralick, R. y G. Elliott (1980). «Increasing tree search efficiency for constraint satisfaction problems». En: *Artificial intelligence* 14.3, págs. 263-313.

Hastings, N. y C.-H. Yeh (1990). «Job oriented production scheduling». En: *European Journal of Operational Research* 47.1, págs. 35-48.

Held, M. y R. Karp (1970). «The Traveling Salesman problem and minimum spanning trees». En: *Operations Research* 18.6, págs. 1138-1162.

Held, M. y R. Karp (1971). «The Traveling Salesman problem and minimum spanning trees: Part II». En: *Mathematical Programming* 1.1, págs. 6-25.

Hoffman, K. L. y M. Padberg (1993). «Solving airline crew scheduling problems by branch-and-cut». En: *Management Science* 39.6, págs. 657-682.

Holthaus, O. y C. Rajendran (1997). «Efficient Dispatching RuIes for Scheduling in a Job Shop». En: *International Journal of Produclion Economics* 48.1, págs. 87-105.

Huang, K. y C. Liao (2008). «Ant colony optimization combined with taboo search for the job shop scheduling problem». En: *Computers & Operations Research* 35.4, págs. 1030-1046.

IBM (2009). «Modeling with IBM ILOG CP Optimizer - Practical Scheduling Examples». En: *IBM*.

ILOG, I. (2012). *http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/*.

Ingolotti, L. y col. (2006). «New heuristics to solve the ÇSOPrailway timetabling problem». En: *Advances in Applied Artificial Intelligence*. Springer, págs. 400-409.

Irohara, T. (2010). «Lagrangian relaxation algorithms for hybrid flow-shop scheduling problems with limited buffers». En: *Biomedical Soft Computing and Human Science* 15.1, págs. 21-28.

Jen, E. (2003). «Stable or Robust? Whats the difference?» En: *Complexity* 8.3, págs. 12-18.

Jünger, M., G. Reinelt y S. Thienel (1995). «Practical problem solving with cutting plane algorithms in combinatorial optimization». En: *In Combinatorial Optimization: DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 20, págs. 111-152.

Kaskavelis, C. A. y M. C. Caramanis (1998). «Efficient Lagrangian relaxation algorithms for industry size job-shop scheduling problems». En: *IIE transactions* 30.11, págs. 1085-1097.

Khemmoudj, M. O. I., M. Porcheron y H. Bennaceur (2006). «When constraint programming and local search solve the scheduling problem of electricite de france nuclear power plant outages». En: *Principles and Practice of Constraint Programming-CP 2006*. Springer, págs. 271-283.

Kirkpatrick, S., C. Gelatt y M. Vecchi (1983). «Optimization by Simulated Annealing, Science, Vol. 220». En: *Number* 4598, págs. 671-680.

Kitano, H. (2007). «Towards a theory of biological robustness». En: *Molecular Systems Biology* 3.137.

Laborie, P. (2009). «IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems». En: *Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR09)*, págs. 148-162.

Laborie, P. (2014). «Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results». En: *Sixth European Conference on Planning*.

Land, A. y A. Doig (1960). «An automatic method for solving discrete programming problems». En: *Econometrica* 28.3, págs. 497-520.

Lau, T. y E. Tsang (1998). «Solving the radio link frequency assignment problem with the guided genetic algorithm». En: *Proceedings, NATO Symposium on Radio Length Frequency Assignment, Sharing and Conservation Systems (Aerospace), Aalborg, Demark*.

Lawrence, S. (1984). *Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques*. Inf. téc. Graduate School of Industrial Administration, Camegie Mellon University, Pittsburgh, USA.

Lecoutre, C. y S. Tabary (2008). «Abscon 112: towards more robustness». En: *3rd International Constraint Solver Competition (CSC'08)*, págs. 41-48.

Lee, Y. C. y A. Y. Zomaya (2009). «Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling». En: *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*. IEEE, págs. 92-99.

Li, R.-K., Y.-T. Shyu y S. Adiga (1993). «A heuristic rescheduling algorithm for computer-based production scheduling systems». En: *The International Journal Of Production Research* 31.8, págs. 1815-1826.

Li, W., A. Zein y col. (2011). «An investigation into fixed energy consumption of machine tools». En: *Glocalized Solutions for Sustainability in Manufacturing*, págs. 268-273.

Mackworth, A. (1977). «Consistency in Networks of Relations». En: *Artificial Intelligence* 8.1, págs. 99-118.

Magnanti, T. y W. R.T. (1990). «Discrete Location Theory». En: ed. por P. B. Mirchandani y R. L. Francis. Wiley-Interscience. Cap. Decomposition Methods for Facility Location Problems, págs. 209-262.

Mangano, S. (1995). «Genetic algorithms solve seemingly intractabe problems». En: *Computer Design* 34.5, págs. 70-91.

Matsuo, H., C. J. Suh y R. S. Sullivan (1988). «A controlled search simulated annealing method for the general jobshop scheduling problem». En: *Austin, TX: Grad. School of Bus., Univ. of Texas*.

Mattfeld, D. C. (1995). *Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling*. Springer-Verlag.

Mencía, C. y col. (2012). «Combining Global Pruning Rules with Depth-First Search for the Job Shop Scheduling Problem with Operators». En: *19th RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*.

Mencía, C. y col. (2015). «Solving the job shop scheduling problem with operators by depth-first heuristic search enhanced with global pruning rules». En: *AI Communications* 28.2, págs. 365-381.

Mitchell, J. E. (2002). «Branch-and-cut algorithms for combinatorial optimization problems». En: *Handbook of Applied Optimization*. Ed. por P. M. Pardalos y M. G. C. Resende. Oxford University Press, págs. 65-77.

Moccia, L. y col. (2006). «A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal». En: *Naval Research Logistics (NRL)* 53.1, págs. 45-59.

Monette, J.-N., Y. Deville, P. Van Hentenryck y col. (2009). «Just-In-Time Scheduling with Constraint Programming.» En: *ICAPS*.

Montanari, U. (1974). «Networks of constraints: Fundamental properties and applications to picture processing». En: *Information sciences* 7, págs. 95-132.

Mouzon, G. y M. Yildirim (2008). «A framework to minimise total energy consumption and total tardiness on a single machine». En: *International Journal of Sustainable Engineering* 1.2, págs. 105-116.

Mouzon, G., M. Yildirim y J. Twomey (2007). «Operational methods for minimization of energy consumption of manufacturing equipment». En: *International Journal of Production Research* 45.18-19, págs. 4247-4271.

Neugebauer, R. y col. (2011). «Structure principles of energy efficient machine tools». En: *CIRP Journal of Manufacturing Science and Technology* 4.2, págs. 136-147.

Newman, S. T. y col. (2012). «Energy efficient process planning for CNC machining». En: *CIRP Journal of Manufacturing Science and Technology* 5.2, págs. 127-136.

Nowicki, E. y C. Smutnicki (1996). «A fast taboo search algorithm for the job shop scheduling problem». En: *Management Science* 42, págs. 797-813.

Pareto, V. y A. N. Page (1971). *Translation of Manuale di economia politica (Manual of political economy). AM Kelley*. Inf. téc. ISBN 978-0-678-00881-2.

Petroleum, B. (2014). *BP statistical review of world energy*.

Pinedo, M. (2005). *Planning and Scheduling in Manufacturing and Services*. Ed. por I. Springer Science + Business Media.

Pinedo, M. y M. Singer (1999). «A Shifting Bottleneck Heuristic for Minimizing the Total Weighted Tardiness in a Job Shop». En: ed. por N. R. Logistics. Vol. 46. 1. John Wiley & Sons, Inc., págs. 1-17.

Planeación Energética y Desarrollo Tecnológico, S. de (2009). «Balance Nacional de Energía 2009». En:

Ponnambalam, S., N. Jawahar y P. Aravindan (1999). «A simulated annealing algorithm for job shop scheduling». En: *Production Planning & Control* 10.8, págs. 767-777.

Poojari, C., G. Mitra y C. Siamitros (2005). «Revisiting Lagrange relaxation for processing large-scale mixed integer programming problems». En: *In Proceedings of IFORS 2005*.

Prosser, P. (1993). «Hybrid algorithms for the constraint satisfaction problem». En: *Computational intelligence* 9.3, págs. 268-299.

Pujawan, I. N. (2004). «Schedule nervousness in a manufacturing system: a case study». En: *Production planning & control* 15.5, págs. 515-524.

Qi, X., J. F. Bard y G. Yu (2006). «Disruption management for machine scheduling: the case of SPT schedules». En: *International Journal of Production Economics* 103.1, págs. 166-184.

Quan, G. y X. Hu (2001). «Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors». En: *Design Automation Conference, 2001. Proceedings*. IEEE, págs. 828-833.

Rizk, A., G. Batt, F. Fages y S. Solima (2009). «A general computational method for robustness analysis with applications to synthetic gene networks». En: *Bioinformatics* 25.12, págs. 168-179.

Rizk, A., G. Batt, F. Fages y S. Soliman (2009). «A general computational method for robustness analysis with applications to synthetic gene networks». En: *Bioinformatics* 25.12, págs. i169-i178.

Rodrigue, J.-P., C. Comtois y B. Slack (2013). *The geography of transport systems*. Routledge.

Roussel, O. y C. Lecoutre (2009). «Xml representation of constraint networks: Format xcsp 2.1». En: *arXiv preprint arXiv:0902.2362*.

Roy, B. y B. Sussman (1964). «Les problemes d'ordonnancements avec contraintes disjonctives». En: Notes DS no. 9 bis, SEMA, Paris.

Sadeh, N. y M. Fox (1996). «Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem». En: *Artificial Intelligence* 86.1, págs. 1-41.

Sadeh, N. (1991). *Look-ahead techniques for micro-opportunistic job shop scheduling*. Inf. téc. DTIC Document.

Sadeh, N., K. Sycara e Y. Xiong (1995). «Backtracking techniques for the job shop scheduling constraint satisfaction problem». En: *Artificial Intelligence* 76.1, págs. 455-480.

Saifullah, A. y col. (2013). «Multi-core real-time scheduling for generalized parallel task models». En: *Real-Time Systems* 49.4, págs. 404-435.

Salido, M. A., J. Escamilla, F. Barber y col. (2015). «Energy efficiency, robustness, and makespan optimality in job-shop scheduling problems». En: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, págs. 1-13.

Salido, M. A., J. Escamilla, A. Giret y F. Barber (2015). «A genetic algorithm for energy-efficiency in job-shop scheduling». En: *The International Journal of Advanced Manufacturing Technology*, págs. 1-12.

Salido, M. A., J. Escamilla, A. Giret, F. Barber y col. (2013). «Energy-aware Parameters in Job-shop Scheduling Problems». En: *GREEN-COPLAS 2013: IJCAI 2013 Workshop on Constraint Reasoning, Planning and Scheduling Problems for a Sustainable Future*, págs. 44-53.

Seo, E. y col. (2008). «Energy efficient scheduling of real-time tasks on multicore processors». En: *Parallel and Distributed Systems, IEEE Transactions on* 19.11, págs. 1540-1552.

Seow, Y. y S. Rahimifard (2011). «A framework for modelling energy consumption within manufacturing systems». En: *CIRP Journal of Manufacturing Science and Technology* 4.3, págs. 258-264.

Sierra, M., C. Mencía y R. Varela (2011). «Optimally scheduling a job-shop with operators and total flow time minimization». En: *Advances in Artificial Intelligence: 14th Conf. of the Spanish Association for Artificial Intelligence, Caepia 2011*. LNAI 7023. Springer, págs. 193-202.

Silva, F. A. M. da, A. C. Moretti y A. T. de Azevedo (2014). «A Scheduling Problem in the Baking Industry». En: *Journal of Applied Mathematics* 2014.

Srinivasan, V. (1971). «A hybrid algorithm for the one machine sequencing problem to minimize total tardiness». En: *Naval Research Logistics Quarterly* 18.3, págs. 317-327.

Steele, D. C. (1975). «The nervous MRP system: how to do battle». En: *Production and Inventory Management* 16.4, págs. 83-89.

Subramaniam, V. y A. S. Raheja (2003). «mAOR: A heuristic-based reactive repair mechanism for job shop schedules». En: *The International Journal of Advanced Manufacturing Technology* 22.9-10, págs. 669-680.

Szathmáry, E. (2006). «A robust approach». En: *Nature* 439.7072, págs. 19-20.

Taillard, E. (1993). «Benchmarks for basic scheduling problems». En: *european journal of operational research* 64.2, págs. 278-285.

Taillard, E. D. (1994). «Parallel taboo search techniques for the job shop scheduling problem». En: *ORSA journal on Computing* 6.2, págs. 108-117.

Tian, G. y col. (2014). «Technology innovation system and its integrated structure for automotive components remanufacturing industry development in China». En: *Journal of Cleaner Production* 85, págs. 419-432.

Van Laarhoven, P. J., E. H. Aarts y J. K. Lenstra (1992). «Job shop scheduling by simulated annealing». En: *Operations research* 40.1, págs. 113-125.

Varela, R., D. Serrano y M. Sierra (2005). «New codification schemas for scheduling with genetic algorithms». En: *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*. Springer, págs. 11-20.

Vepsalanen, A. y T. Morton (1987). «Priority rules for job shops with weighted tardiness costs». En: *Management Science* 33.8, págs. 1035-1047.

Vieira, G. E., J. W. Herrmann y E. Lin (2000). «Predicting the performance of rescheduling strategies for parallel machine systems». En: *Journal of Manufacturing Systems* 19.4, págs. 256-266.

Wang, X. y col. (2011). «A low-carbon production scheduling system considering renewable energy». En: *Service Operations, Logistics, and Informatics (SOLI), 2011 IEEE International Conference on*. IEEE, págs. 101-106.

Watson, J.-P. y col. (1999). «Algorithm Performance and Problem Structure for Flow-shop Scheduling.» En: *AAAI/IAAI*, págs. 688-695.

Webster, S. y M. Azizoglu (2001). «Dynamic programming algorithms for scheduling parallel machines with family setup times». En: *Computers & Operations Research* 28.2, págs. 127-137.

Weinert, N., S. Chiotellis y G. Seliger (2011). «Methodology for planning and operating energy-efficient production systems». En: *CIRP Annals-Manufacturing Technology* 60.1, págs. 41-44.

Wu, H.-H. y R.-K. Li (1995). «A new rescheduling method for computer based scheduling systems». En: *International journal of production research* 33.8, págs. 2097-2110.

Wu, S. D., R. H. Storer y C. Pei-Chann (1993). «One-machine rescheduling heuristics with efficiency and stability as criteria». En: *Computers & Operations Research* 20.1, págs. 1-14.

Xu, Y. y col. (2014). «A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues». En: *Information Sciences* 270, págs. 255-287.

Yan, J. y L. Li (2013). «Multi-objective optimization of milling parameters–the trade-offs between energy, production rate and cutting quality». En: *Journal of Cleaner Production* 52, págs. 462-471.

Zhong, J.-H. y col. (2013). «A differential evolution algorithm with dual populations for solving periodic railway timetable scheduling problem». En: *Evolutionary Computation, IEEE Transactions on* 17.4, págs. 512-527.