



UNIVERSIDAD
POLITECNICA
DE VALENCIA

**TÉCNICAS
METAHEURÍSTICAS PARA
LA PROGRAMACIÓN
FLEXIBLE DE LA
PRODUCCIÓN**

TESIS DOCTORAL

Presentada por:

D. Rubén Ruiz García

Dirigida por:

Dra. D^a. Concepción Maroto Álvarez

Valencia, 2003

RESUMEN

TÉCNICAS METAHEURÍSTICAS PARA LA PROGRAMACIÓN FLEXIBLE DE LA PRODUCCIÓN

La programación de la producción o la asignación de órdenes de producción a máquinas y su posterior secuenciación, representa un problema importante dentro de las industrias. Actualmente se tiende a ofrecer un catálogo de productos muy amplio donde además se pretende que estos productos estén diferenciados de la competencia. Los clientes son cada vez más exigentes en términos de precio, calidad y plazos de entrega y por otro lado, las economías emergentes de países en vías de desarrollo son una fuente de fuerte competencia a nivel de precios. Todos estos factores, unidos al hecho de que, por norma general, las empresas disponen de sistemas de producción poco flexibles y orientados hacia la producción en masa, hacen que sea necesaria una eficiente utilización de los recursos para poder mantener el nivel competitivo. Una excelente estrategia empresarial y una buena planificación de la producción necesitan también de una buena programación de la producción para conseguir los objetivos de rentabilidad y servicio.

Los primeros trabajos sobre programación de la producción aparecieron a mediados de los años 50 y contrariamente a lo que cabría esperar, después de 50 años de investigación los problemas de programación de la producción siguen, en la mayoría de los casos, sin una solución satisfactoria. Esto se hace evidente por el hecho de que existe un “gap” importante entre la teoría de la programación de la producción o “scheduling” y la práctica. La mayoría de los métodos propuestos se basan en hipótesis de partida que rara vez se dan en la práctica, de ahí que muchas

empresas sigan utilizando métodos manuales para realizar la programación de la producción.

El objetivo principal de esta Tesis Doctoral es precisamente el de desarrollar métodos para la programación flexible de la producción, abarcando desde los problemas más clásicos de tipo taller de flujo hasta problemas complejos extraídos del entorno productivo, concretamente del sector azulejero, como pueden ser los talleres de flujo híbrido con tiempos de cambio de partida dependientes de la secuencia. Estos problemas, por su naturaleza combinatoria, pertenecen a la clase de problemas \mathcal{NP} -Completos, o lo que es lo mismo, no existen técnicas generales para obtener una solución óptima en un tiempo aceptable.

En una primera fase se ha realizado una completa revisión del estado del arte para el problema del taller de flujo de permutación, junto con una exhaustiva evaluación comparativa de 25 métodos, tanto heurísticos como metaheurísticos, para este problema. Para las evaluaciones se ha utilizado el conjunto de 120 problemas estándar de Taillard (1993) y los resultados de la comparativa permiten identificar los mejores métodos para el problema considerado.

Posteriormente se proponen dos nuevos algoritmos genéticos para el taller de flujo que incorporan cuatro nuevos operadores de cruce, una inicialización de la población mediante el uso de heurísticas eficaces, un operador de reinicialización y un nuevo esquema generacional, así como una hibridación con búsqueda local. Para la calibración de los distintos operadores y parámetros de los algoritmos se utilizan amplios diseños de experimentos. Los resultados ponen de manifiesto que los dos algoritmos desarrollados son mejores, en términos de eficiencia y eficacia que el resto de métodos considerados, con unas diferencias que se sitúan entre el 9 % y el 49 %.

Se considera también un problema más general como es el taller de flujo de permutación con tiempos de cambio de partida dependientes de la secuencia. Para este problema, sensiblemente más difícil que el anterior, también se realiza una revisión del estado del arte así como una adaptación de los dos algoritmos genéticos anteriores. Los algoritmos genéticos adaptados se calibran de nuevo

con un conjunto de 480 problemas donde se consideran distintas magnitudes de los tiempos de cambio. Los algoritmos se comparan con 11 métodos entre los que tenemos adaptaciones de los mejores algoritmos para el taller de flujo estándar así como heurísticas diseñadas específicamente para la versión con tiempos de cambio de partida. Los resultados indican que ambos algoritmos genéticos propuestos mejoran todos los resultados anteriores entre un 50 % y un 488 %.

Otro tipo de problemas más realistas son los talleres de flujo híbridos que se dan en el sector azulejero. Para la correcta caracterización del sistema productivo en este sector, se presenta una completa descripción del proceso de producción del azulejo así como una revisión del estado del arte para este tipo de problemas. En este caso se propone también un algoritmo genético y un amplio diseño de experimentos para la calibración del mismo que comprende 12 experimentos distintos con 1320 problemas en total. También se presentan adaptaciones de varios de los mejores algoritmos evaluados. De nuevo, el algoritmo propuesto representa una mejora de entre el 53 % y el 135 %.

Por último se muestra una aplicación del algoritmo propuesto a un conjunto de datos reales extraídos de empresas del sector azulejero donde se contemplan todas las restricciones para la correcta programación de la producción. Los resultados indican que las secuencias obtenidas con el algoritmo propuesto suponen una mejora con respecto a las secuencias obtenidas por el responsable de la programación de la producción de entre un 2,32 % y un 16,95 %. El algoritmo propuesto se ha codificado dentro de un prototipo software que actualmente se está implantado en dos importantes empresas del sector.

RESUM

TÈCNIQUES METAHEURÍSTIQUES PER A LA PROGRAMACIÓ FLEXIBLE DE LA PRODUCCIÓ

La programació de la producció o l'assignació d'ordes de producció a màquines i la seu posterior seqüenciació, representa un problema important dins de les indústries. Actualment es tendix a oferir un catàleg de productes molt ampli on a més es pretén que estos productes estiguin diferenciatos de la competència. Els clients són cada vegada més exigents en termes de preu, qualitat i terminis d'entrega i d'altra banda, les economies emergents de països en vies de desenrotllament són una font de forta competència pel que fa als preus. Tots estos factors, units al fet, que per norma general, les empreses disposen de sistemes de producció poc flexibles i orientats cap a la producció en massa, fan que siga necessària una eficient utilització dels recursos per a poder mantindre el nivell competitiu. Una excel·lent estratègia empresarial i una bona planificació de la producció necessiten també d'una bona programació de la producció per a aconseguir els objectius de rendibilitat i servei.

Els primers treballs sobre programació de la producció van aparéixer a mitjan els anys 50 i contràriament al que cabria esperar, després de 50 anys d'investigació els problemes de programació de la producció seguixen, en la majoria dels casos, sense una solució satisfactòria. Açò es fa evident pel fet que hi ha un “gap” important entre la teoria de la programació de la producció o “*scheduling*” i la pràctica. La majoria dels mètodes proposats es basen en hipòtesis de partida que rara vegada es donen en la pràctica, d'ací que moltes empreses seguisquen

utilitzant mètodes manuals per a realitzar la programació de la producció.

L'objectiu principal d'esta Tesi Doctoral és precisament el de desenrotllar mètodes per a la programació flexible de la producció, abraçant des dels problemes més clàssics de tipus taller de flux fins a problemes complexos extrets de l'entorn productiu, concretament del sector del taulelet, com poden ser els tallers de flux híbrid amb temps de canvi de partida dependents de la seqüència. Estos problemes, per la seu naturalesa combinatòria, pertanyen a la classe de problemes \mathcal{NP} -Complets, o el que és el mateix, no existixen tècniques generals per a obtindre'n una solució òptima en un temps acceptable.

En una primera fase s'ha realitzat una completa revisió de l'estat de l'art per al problema del taller de flux de permutació, junt amb una exhaustiva evaluació comparativa de 25 mètodes, tant heurístics com metaheurístics, per a este problema. Per a les evaluacions s'ha utilitzat el conjunt de 120 problemes estàndard de Taillard (1993) i els resultats de la comparativa permeten identificar els millors mètodes per al problema considerat.

Posteriorment es proposen dos nous algoritmes genètics per al taller de flux que incorporen quatre nous operadors d'encreuament, una inicialització de la població per mitjà de l'ús d'heurístiques eficaces, un operador de reinicialització i un nou esquema generacional, així com una hibridació amb recerca local. Per al calibratge dels distints operadors i paràmetres dels algoritmes s'utilitzen amplis dissenys d'experiments. Els resultats posen de manifest que els dos algoritmes desenrotllats són millors, en termes d'eficiència i eficàcia que la resta de mètodes considerats, amb unes diferències que se situen entre el 9% i el 49%.

Es considera també un problema més general com és el taller de flux de permutació amb temps de canvi de partida dependents de la seqüència. Per a este problema, sensiblement més difícil que l'anterior, també es realitza una revisió de l'estat de l'art així com una adaptació dels dos algoritmes genètics anteriors. Els algoritmes genètics adaptats es calibren de nou amb un conjunt de 480 problemes on es consideren distintes magnituds dels temps de canvi. Els algoritmes es comparen amb 11 mètodes entre els quals tenim adaptacions dels millors algoritmes

per al taller de flux estàndard així com heurístiques dissenyades específicament per a la versió amb temps de canvi de partida. Els resultats indiquen que estos dos algoritmes genètics proposats milloren tots els resultats anteriors entre un 50% i un 488%.

Un altre tipus de problemes més realistes són els tallers de flux híbrids que es donen en el sector del taulell. Per a la correcta caracterització del sistema productiu en este sector, es presenta una completa descripció del procés de producció del taulell així com una revisió de l'estat de l'art per a este tipus de problemes. En este cas es proposa també un algoritme genètic i un ampli disseny d'experiments per al calibratge del mateix que comprén 12 experiments distints amb 1320 problemes en total. També es presenten adaptacions de diversos dels millors algoritmes avaluats. De nou, l'algoritme proposat representa una millora d'entre el 53% i el 135%.

Finalment es mostra una aplicació de l'algoritme proposat a un conjunt de dades reals extretes d'empreses del sector del taulell on es contemplen totes les restriccions per a la correcta programació de la producció. Els resultats indiquen que les seqüències obtingudes amb l'algoritme proposat suposen una millora respecte a les seqüències obtingudes pel responsable de la programació de la producció d'entre un 2,32% i un 16,95%. L'algoritme proposat s'ha codificat dins d'un prototip programari que actualment s'està implantat en dos importants empreses del sector.

ABSTRACT

METAHEURISTIC TECHNIQUES FOR FLEXIBLE PRODUCTION PROGRAMMING

Production programming, or the assignment of production orders to machines and the subsequent sequencing represents an important problem for industries. Today, industries aim to offer a wide catalog of products that must be in addition clearly different from those offered by competitors. Clients are more and more demanding in terms of price, quality and shipment date. Additionally, the emerging economies of developing countries are an important source of pressure concerning price levels. All these factors, together with the fact that usually the companies operate production systems which are rigid and oriented towards mass production, raise the necessity of efficiently using the available resources for maintaining the competitive edge. An excellent global strategy and a good production planning need also a good production programming in order to achieve the goals of profit and service.

The first papers about production programming were published in the 50s and contrary to what should be expected after 50 years of research, production programming problems remain, in the majority of cases, unsolved. Evidently, there is an important gap between the theory of scheduling and the practice. A big part of the methods proposed rely on strong hypotheses that are rarely seen in practice. This is the main cause of the lack of use of these methods by companies in which only manual methods are used for the production programming.

The main objective of this Doctoral Thesis is precisely to develop methods for flexible production programming, reaching from the most classical problems to complex real problems taken from the production environment. More precisely, from the ceramic tile sector. Problems like this can be hybrid flowshop problems with sequence-dependent set-up times. These problems, being combinatorial in nature, belong to the class of \mathcal{NP} -Complete problems, which means that there are no general algorithms for obtaining the optimal solution in an acceptable amount of time.

In the first stage of the study, we have conducted a complete state-of-the-art review of the permutation flowshop problem, along with an exhaustive comparative evaluation of 25 methods, both heuristic and metaheuristic for this problem. For the evaluations we have used the 120 problem set of Taillard (1993) and the results of the evaluation allow us to identify the best methods available for the problem considered.

Afterwards, we propose two new genetic algorithms for the flowshop problem that include four new crossover operators, an initialization of the population by effective heuristics, a restart operator and a new generational scheme, as well as a hybridization with local search. For the calibration of the different operators and parameters we have used broad designs of experiments. The results evidence that the two proposed algorithms are better, both in terms of effectiveness and efficiency than the other considered methods, with the differences being between 9% and 49%.

A more general problem is also considered, which is the permutation flowshop problem with sequence-dependent set-up times. For this problem, which is more difficult than the previous one, we have done a state-of-the-art review as well as an adaptation of the two previously presented genetic algorithms. This adapted algorithms are calibrated again with a set of 480 problems where different magnitudes of the set up times are considered. The algorithms are compared against 11 methods, taken from among the adaptations of the best algorithms for the regular flowshop as well as specific methods for the sequence-dependent set-up times version. Results indicate that both proposed genetic algorithms are better than all

other methods in a range from 50% to 488%.

Another more realistic type of problems are the hybrid flowshops found in the ceramic tile sector. For the correct characterization of the productive system in this sector, we present a complete description of the ceramic tile production process, as well as a state-of-the-art review of this type of problems. In this case, we also propose a new genetic algorithm and a broad experimental design that comprehends 12 different experiments with 1230 problems in total. We also present adaptations of the best methods evaluated. Once again, the new algorithm proposed represents and enhancement of between a 53% and a 135%.

Lastly, we show an application of the proposed algorithm to a set of real problems taken from companies from the ceramic tile sector where all constraints are taken into account for the correct production programming. The results obtained illustrate that the schedules obtained with the proposed algorithm can enhance the schedules obtained by the production manager of the company in a percentage from 2,32% to 16,95%. The proposed algorithm has been coded inside a software prototype that is currently being deployed at two important companies of the sector.

Agradecimientos

Son muchas las personas que directa o indirectamente han colaborado o contribuido en buena medida para la consecución de este trabajo. Voy a intentar dar las gracias a todas ellas aunque es posible que por olvido y no por omisión me deje alguna mención, por ello pido mis más sinceras disculpas.

Difícilmente este trabajo hubiese terminado sin la inestimable ayuda de mi compañero el Dr. D. Javier Alcaraz Soria, su constante dedicación y disponibilidad ha sido muy importantes en esta Tesis Doctoral y en varias ocasiones sus comentarios y consejos han sido el catalizador de ideas y desarrollos. Agradecer a la Dra. D^a Concepción Maroto Álvarez que a pesar de su intensa dedicación a la gestión universitaria siempre ha estado disponible y ha realizado una excelente y esmerada labor de dirección. También querría agradecer el apoyo recibido por parte de mis compañeros el Dr. D. Juan Carlos García Díaz, la Dra. D^a. Susana San Matías Izquierdo y D. José Miguel Carot Sierra, por soportar estoicamente mis constantes idas y venidas a sus respectivos despachos.

Querría agradecer también al profesor Christos Koulamas, del College of Business Administration de Florida, por su ayuda en la implementación de su algoritmo heurístico HFC. Al profesor Thomas Stützle, de la Technische Universität de Darmstadt, por sus indicaciones en su método de búsqueda local iterativa. Agradecer también al profesor Éric Taillard de la University of Applied Sciences of Western Switzerland, por proporcionarme información sobre las últimas versiones y cotas de su conocido banco de pruebas y a los profesores y profesionales Jean Paul Watson, de la Colorado State University, Colin Reeves de la Coventry University y Takeshi Yamada de la Nippon Telegraph and Telephone Corporation (NTT), por haber respondido desinteresadamente numerosos correos aclarando

dudas y proporcionando una gran ayuda. Me gustaría agradecer también a los compañeros y amigos del Grupo de Investigación Operativa GIO, en especial la profesora D^a. Eva Vallada Regalado y al becario D. Miguel Edo Aparicio por su colaboración en la implementación de los bancos de pruebas y aspectos de la aplicación de programación de la producción.

La colaboración por parte de las empresas también ha sido esencial, por ello me gustaría agradecer a D. José Romeo, de la empresa Cerypsa Cerámicas S.A., a D. Javier Marqués y a D. José Luís Salaj, de la empresa Halcón Cerámicas S.A. y a D. Fernando Soriano y a D. Luís Rodrigo, de la empresa Gres de Valls S.A., por todo el tiempo que han dedicado a este proyecto y por la gran cantidad de información y datos que nos han proporcionado.

Por último agradecer al Departamento de Estadística e Investigación Operativa Aplicadas y Calidad el excelente ambiente de trabajo, realmente los compañeros y el entorno han hecho un poco más fácil desarrollar esta Tesis Doctoral.

Muchas gracias a todos.

A Gema, por su compresión y apoyo, que
han sido indispensables para mí.

A mis padres, constante fuente de
motivación y estímulo.

ÍNDICE GENERAL

1. Introducción	1
1.1. Programación de la producción	4
1.2. Objetivos y Metodología	10
1.3. Esquema general de la Tesis Doctoral	12
2. Conceptos previos	15
2.1. Notación y clasificación de los problemas de programación de la producción	15
2.2. Complejidad computacional	26
3. El problema del taller de flujo	35
3.1. Métodos de programación de la producción para el taller de flujo .	41
3.1.1. Algoritmo clásico de Johnson	43
3.1.2. Reglas de prioridad	46
3.1.3. Métodos exactos	50
3.1.4. Técnicas heurísticas	58
3.1.4.1. Heurísticas constructivas	58
3.1.4.2. Heurísticas de mejora	73
3.1.5. Técnicas metaheurísticas	74
3.1.5.1. Recocido simulado (“ <i>Simulated Annealing</i> ”) .	75
3.1.5.2. Búsqueda Tabú (“ <i>Tabu Search</i> ”)	77
3.1.5.3. Algoritmos Genéticos	79
3.1.5.4. Otras técnicas metaheurísticas	84

3.2. Evaluación de heurísticas para el problema del taller de flujo	86
3.3. Conclusiones del capítulo	115
4. Nuevos algoritmos genéticos para el problema del taller de flujo 117	
4.1. Funcionamiento general de los algoritmos genéticos	118
4.2. Algoritmos genéticos y el problema del taller de flujo de permutación	122
4.2.1. Representación genética	122
4.2.2. Evaluación	124
4.2.3. Selección	125
4.2.4. Cruce	128
4.2.5. Mutación	140
4.3. Un nuevo algoritmo genético	143
4.3.1. Representación, evaluación e inicialización de la población	143
4.3.2. Selección	147
4.3.3. Cruce	147
4.3.4. Mutación y operador de reinicialización	158
4.3.5. Esquema generacional	162
4.3.6. Evaluación experimental	167
4.4. Nuevo algoritmo genético híbrido	186
4.5. Evaluación de los algoritmos genéticos propuestos	190
4.6. Conclusiones del capítulo	201
5. El taller de flujo con tiempos de cambio de partida 203	
5.1. La complejidad del Taller SDST	208
5.2. Métodos exactos para la programación de la producción en el taller SDST	209
5.3. Métodos heurísticos y metaheurísticos para la programación de la producción en el taller SDST	211
5.4. Métodos para otros problemas con tiempos de cambio de partida .	214
5.5. Adaptación del algoritmo genético al taller SDST	218
5.5.1. Representación genética, evaluación e inicialización . . .	219
5.5.2. Selección y operadores de cruce	220

5.5.3. Mutación, reinicialización y esquema generacional	221
5.5.4. Evaluación de los parámetros del algoritmo genético	222
5.6. Adaptación del HGA al taller SDST	242
5.7. Evaluación comparativa de heurísticas y metaheurísticas para el taller SDST	243
5.8. Conclusiones del capítulo	264
6. Taller de flujo híbrido con tiempos de cambio de partida: una aplicación al sector cerámico	265
6.1. El sector de producción de baldosas cerámicas	267
6.2. El azulejo o baldosa cerámica	269
6.3. Tipos de baldosas cerámicas	270
6.4. El proceso de producción de las baldosas cerámicas	272
6.4.1. Monococción porosa	275
6.4.2. Biccoción rápida o doble cocción rápida	275
6.4.3. Bicocción mixta o 2 ^a bicocción rápida	275
6.4.4. Fabricación de baldosas extrudidas	275
6.4.5. Preparación de las materias primas	280
6.4.6. Molienda o molturación	280
6.4.6.1. Molturación por vía húmeda	281
6.4.6.2. Secado por atomización	282
6.4.7. Humectación y amasado	284
6.4.8. Extrusión	284
6.4.9. Prensado	284
6.4.10. Secado	287
6.4.11. Esmaltado y serigrafía	289
6.4.12. Preparación de esmaltes	293
6.4.13. Cocción	294
6.4.14. Clasificación y embalaje	297
6.4.15. Otros aspectos de la producción de baldosas cerámicas .	299
6.5. El problema de la programación de la producción en el sector cerámico	301
6.5.1. Caracterización del sistema productivo	303

6.5.2.	El taller de flujo híbrido	307
6.5.3.	Métodos para la programación de la producción en talleres de flujo híbrido	311
6.5.4.	Algoritmos genéticos y el problema de la programación de la producción en el sector cerámico	320
6.5.4.1.	Representación genética y función de evaluación	320
6.5.4.2.	Inicialización de la población	325
6.5.4.3.	Otros parámetros y operadores del algoritmo genético	326
6.5.4.4.	Calibración de los parámetros y operadores del algoritmo genético	327
6.6.	Evaluación de técnicas para la programación de la producción en el sector cerámico	344
6.7.	Conclusiones del capítulo	364
7.	Conclusiones	367
REFERENCIAS		373
A.	Tests estadísticos	399
A.1.	Tests de comprobación de la hipótesis de homocedasticidad (ANOVA Capítulo 4)	399
A.2.	Experimentos y tablas ANOVA, Capítulo 5	402
A.2.1.	Tests de comprobación de la hipótesis de homocedasticidad, experimento SSD10	402
A.2.2.	Experimento SSD50	405
A.2.3.	Experimento SSD100	414
A.2.4.	Experimento SSD125	423
A.3.	Experimentos y tablas ANOVA, Capítulo 6	432
A.3.1.	Experimento SSD50_P13	432
A.3.2.	Experimento SSD100_P13	437
A.3.3.	Experimento SSD125_P13	441
A.3.4.	Experimento SSD10_P2	445
A.3.5.	Experimento SSD50_P2	449

A.3.6. Experimento SSD100_P2	453
A.3.7. Experimento SSD125_P2	457
A.3.8. Experimento SSD10_P3	461
A.3.9. Experimento SSD50_P3	465
A.3.10. Experimento SSD100_P3	469
A.3.11. Experimento SSD125_P3	473
B. Bancos de datos y problemas ejemplo	477
B.1. Problemas para el taller SDST	477
B.2. Problemas para el taller de flujo híbrido con tiempos de cambio de partida dependientes de la secuencia	485
B.3. Problemas reales de planificación de la producción	506
C. Prototipo de software: código fuente	507
D. Tipos de baldosas cerámicas	515
D.1. Azulejo	515
D.2. Pavimento de gres	516
D.3. Gres porcelánico	517
D.4. Baldosín catalán	518
D.5. Gres rústico	519
D.6. Barro cocido	520

ÍNDICE DE FIGURAS

1.1.	Problemas de decisión de los procesos productivos.	3
1.2.	Esquema simplificado del proceso de producción de baldosas cerámicas; un ejemplo de taller de flujo o “ <i>flow shop</i> ”.	7
3.1.	Esquema del problema de taller de flujo o “ <i>flowshop</i> ”.	36
3.2.	Esquema de un problema de tipo taller de flujo generalizado o “ <i>general flow shop</i> ”.	37
3.3.	Diferencia entre secuencias generales y secuencias de permutación.	40
3.4.	Diagrama de Gantt para la secuencia resultado de la aplicación del algoritmo basado en la regla de Johnson.	45
3.5.	Comparación entre la versión estándar y mejorada de las reglas de prioridad para el ejemplo: regla SPT.	50
3.6.	Diagrama de Gantt para el ejemplo de 4 máquinas y 5 trabajos donde se detallan las variables del modelo de Wagner.	52
3.7.	Solución óptima para el ejemplo de 4 máquinas y 5 trabajos resuelto con el modelo de Wagner.	55
3.8.	Paso 2 de la heurística NEH para el ejemplo.	64
3.9.	Paso 3 de la heurística NEH para el ejemplo, $k = 3$	65
3.10.	Paso 3 de la heurística NEH para el ejemplo, $k = 4$	67
3.11.	Paso 3 de la heurística NEH para el ejemplo, $k = n$, última iteración.	70

3.12.	Diagrama de dispersión del incremento porcentual total por encima del óptimo o mínima cota superior conocida (<i>IPSOM</i>) frente al tiempo de proceso para todos los métodos evaluados (criterio de parada 50.000 “ <i>makespans</i> ”).	110
3.13.	Evolución del incremento porcentual total por encima del óptimo o mínima cota superior conocida (<i>IPSOM</i>) frente al número de C_{max} evaluados para los métodos metaheurísticos.	112
3.14.	Incremento porcentual total por encima del óptimo o mínima cota superior conocida (<i>IPSOM</i>) de los mejores métodos de la comparativa para todos los grupos de problemas de Taillard (criterio de parada 50.000 C_{max}).	113
4.1.	Esquema general de funcionamiento de los algoritmos genéticos.	120
4.2.	Ejemplo de representación genética ordinal para el problema ta001 de Taillard (20 trabajos).	123
4.3.	Programa de producción a partir de la representación genética ordinal para el problema ta001 de Taillard.	124
4.4.	Operador de cruce de un punto estándar.	130
4.5.	Errores en el cruce de un punto estándar.	130
4.6.	Segunda fase del cruce de un punto por orden.	131
4.7.	Operador de cruce de dos puntos por orden.	132
4.8.	Operador de cruce “ <i>Partially Matched Crossover</i> ” (PMX).	134
4.9.	Fallos en el operador de cruce “ <i>Partially Matched Crossover</i> ” (PMX) estándar.	135
4.10.	Resultado de aplicar el operador PMX modificado.	136
4.11.	Operador de cruce “ <i>Order Crossover</i> ” (OX).	137
4.12.	Operador de cruce “ <i>Uniform Order Based Crossover</i> ” (UOB). . .	139
4.13.	Operador de cruce “ <i>Generalized Position Crossover</i> ” (GPX). . .	140
4.14.	Mutación por intercambio (“ <i>SWAP mutation</i> ”).	142
4.15.	Mutación por intercambio adyacente (“ <i>POSITION mutation</i> ”). .	142
4.16.	Mutación por desplazamiento (“ <i>SHIFT mutation</i> ”).	143
4.17.	Destrucción de los bloques en el operador de cruce “ <i>Order Crossover</i> ” o OX.	150

4.18.	Operador de cruce “ <i>Similar Job Order Crossover</i> ” (SJOX).	152
4.19.	Operador de cruce “ <i>Similar Block Order Crossover</i> ” (SBOX).	154
4.20.	Operador de cruce “ <i>Similar Job 2-Point Order Crossover</i> ” (SJ2OX).	156
4.21.	Operador de cruce “ <i>Similar Block 2-Point Order Crossover</i> ” (SB2OX).	158
4.22.	Evolución del mejor C_{max} (trazo rojo) y del C_{max} medio (trazo verde) de la población frente al número de generaciones. Problema ta001 de Taillard.	159
4.23.	Evolución del mejor C_{max} (trazo rojo) y del C_{max} medio (trazo verde) de la población frente al número de generaciones aplicando operador de reinicialización. Problema ta001 de Taillard.	161
4.24.	Histograma con la distribución de los distintos valores del C_{max} para el problema car7 de Carlier.	165
4.25.	Diagrama de flujo con el esquema generacional propuesto.	167
4.26.	Gráfico de probabilidad Normal para comprobar la hipótesis de normalidad en el ANOVA.	172
4.27.	Gráfico de los residuos frente a los niveles del factor probabilidad de cruce (Cross_Prob) para comprobar la hipótesis de homocedasticidad en el ANOVA.	173
4.28.	Gráfico de los residuos frente los valores previstos de IPSOM total para comprobar la hipótesis de homocedasticidad en el ANOVA.	174
4.29.	Gráfico de residuos frente al orden de ejecución de las pruebas para comprobar la hipótesis de independencia en el ANOVA.	175
4.30.	Gráfico de medias e intervalos LSD al 95 % para el factor tipo de selección (Select_Type).	178
4.31.	Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de mutación (Mut_Prob).	179
4.32.	Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores tipo de selección (Select_Type) y probabilidad de mutación (Mut_Prob).	180

4.33.	Gráfico de medias e intervalos LSD al 95 % para el factor operador de reinicialización (Restart)	181
4.34.	Gráfico de medias e intervalos LSD al 95 % para el factor tipo de cruce (Cross_Type)	182
4.35.	Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de cruce (Cross_Prob)	183
4.36.	Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores tamaño de la población (Pop_Size) y tipo de selección (Select_Type)	184
4.37.	Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores probabilidad de cruce (Cross_Prob) y tipo de selección (Select_Type)	185
4.38.	Evolución del <i>IPSOM</i> total frente al número de C_{max} evaluados para los mejores métodos metaheurísticos y los algoritmos genéticos propuestos.	200
5.1.	Diferencia entre tiempos de cambio de partida anticipativos y no anticipativos para un ejemplo de secuencia con 2 trabajos y 2 máquinas.	206
5.2.	Gráfico de probabilidad Normal para comprobar la hipótesis de normalidad en el ANOVA, experimento SSD10.	228
5.3.	Gráfico de los residuos frente a los niveles del factor Cross_Prob, experimento SSD10.	229
5.4.	Gráfico de los residuos frente los valores previstos de <i>IPSOM</i> total, experimento SSD10.	230
5.5.	Gráfico de residuos frente al orden de ejecución de las pruebas, experimento SSD10.	231
5.6.	Gráfico de medias e intervalos LSD al 95 % para el factor tipo de selección (Select_Type), experimento SSD10.	233
5.7.	Gráfico de medias e intervalos LSD al 95 % para el factor operador de reinicialización (Restart), experimento SSD10.	234
5.8.	Gráfico de medias e intervalos LSD al 95 % para el factor tipo de cruce (Cross_Type), experimento SSD10.	235

5.9.	Gráfico ampliado de medias e intervalos LSD al 95 % para el factor tipo de cruce (Cross_Type) donde se observan los mejores operadores, experimento SSD10.	236
5.10.	Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de mutación (Mut_Prob), experimento SSD10.	237
5.11.	Gráfico de medias e intervalos LSD al 95 % para el factor tamaño de población (Pop_Size), experimento SSD10.	238
5.12.	Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores tipo de selección (Select_Type) y probabilidad de mutación (Mut_Prob), experimento SSD10.	239
5.13.	Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de cruce (Cross_Prob), experimento SSD10.	240
5.14.	Evolución del <i>IPSOM</i> total frente a la magnitud de los tiempos de cambio para los mejores métodos metaheurísticos.	263
6.1.	Evolución de la producción y exportación del sector cerámico en España desde el año 1970 hasta el 2001.	269
6.2.	Procesos de fabricación de baldosas cerámicas.	274
6.3.	Fabricación de baldosas por monococción.	276
6.4.	Fabricación de baldosas por bicocción rápida.	277
6.5.	Fabricación de baldosas por bicocción mixta.	278
6.6.	Fabricación de baldosas por extrusión.	279
6.7.	Molino de bolas.	282
6.8.	Atomizador.	283
6.9.	Prensa hidráulica.	286
6.10.	Recogedor de la prensa.	287
6.11.	Secadero vertical.	288
6.12.	Secadero horizontal.	289
6.13.	Aplicación de esmaltes mediante campana.	290
6.14.	Rascador por vía seca.	291
6.15.	Girador de baldosas.	291
6.16.	Cabina aerógrafo.	292
6.17.	Máquina para cepillar vertical.	293

6.18.	Pantalla serigráfica o máquina de decorar.	293
6.19.	Horno, zona de precalentamiento.	295
6.20.	Horno, zona de cocción.	296
6.21.	Horno, zona de enfriamiento.	297
6.22.	Máquina paletizadora.	298
6.23.	Carros filoguiados.	299
6.24.	Compensador en línea.	300
6.25.	Sistema de clasificación automática.	301
6.26.	Etapas de producción en el proceso de monococción con preparación de materias primas por vía húmeda.	304
6.27.	Etapas de producción en el proceso simplificado de monococción con preparación de materias primas por vía húmeda.	305
6.28.	Esquema del problema del taller de flujo con múltiples procesadores o FSMP.	307
6.29.	Gráfico de medias e intervalos LSD al 95 % para el factor tipo de selección (<i>Select_Type</i>), experimento SSD10_P13.	334
6.30.	Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de mutación (<i>Mut_Prob</i>), experimento SSD10_P13.	335
6.31.	Gráfico de medias e intervalos LSD al 95 % para el factor tamaño de la población (<i>Pop_Size</i>), experimento SSD10_P13.	336
6.32.	Gráfico de medias e intervalos LSD al 95 % para para la interacción entre los factores tamaño de la población (<i>Pop_Size</i>) y tipo de selección (<i>Select_Type</i>), experimento SSD10_P13.	337
6.33.	Gráfico de medias e intervalos LSD al 95 % para para la interacción entre los factores probabilidad de mutación (<i>Mut_Prob</i>) y tipo de selección (<i>Select_Type</i>), experimento SSD10_P13.	338
6.34.	Gráfico de medias e intervalos LSD al 95 % para el factor tipo de cruce (<i>Cross_Type</i>), experimento SSD10_P13.	339
6.35.	Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de cruce (<i>Cross_Prob</i>), experimento SSD10_P13.	340

6.36. Gráfico de medias e intervalos LSD al 95 % para para la interacción entre los factores probabilidad de mutación (Mut_Prob) y operador de reinicialización (Restart), experimento SSD10_P13.	341
6.37. Detalle del gráfico de medias e intervalos LSD al 95 % para para la interacción entre los factores probabilidad de mutación (Mut_Prob) y operador de reinicialización (Restart), experimento SSD10_P13.	342
A.1. Gráfico de los residuos frente a los niveles del factor Cross_Type.	400
A.2. Gráfico de los residuos frente a los niveles del factor Mut_Prob.	400
A.3. Gráfico de los residuos frente a los niveles del factor Pop_Size.	401
A.4. Gráfico de los residuos frente a los niveles del factor Restart..	401
A.5. Gráfico de los residuos frente a los niveles del factor Select_Type.	402
A.6. Gráfico de los residuos frente a los niveles del factor Cross_Type, experimento SSD10.	403
A.7. Gráfico de los residuos frente a los niveles del factor Mut_Prob, experimento SSD10.	403
A.8. Gráfico de los residuos frente a los niveles del factor Pop_Size, experimento SSD10.	404
A.9. Gráfico de los residuos frente a los niveles del factor Restart, experimento SSD10.	404
A.10. Gráfico de los residuos frente a los niveles del factor Select_Type, experimento SSD10.	405
A.11. Gráfico de probabilidad Normal, experimento SSD50.	405
A.12. Gráfico de los residuos frente a los niveles del factor Cross_Prob, experimento SSD50.	406
A.13. Gráfico de los residuos frente a los niveles del factor Cross_Type, experimento SSD50.	406
A.14. Gráfico de los residuos frente a los niveles del factor Mut_Prob, experimento SSD50.	407

A.15.	Gráfico de los residuos frente a los niveles del factor Pop_Size, experimento SSD50.	407
A.16.	Gráfico de los residuos frente a los niveles del factor Restart, experimento SSD50.	408
A.17.	Gráfico de los residuos frente a los niveles del factor Select_Type, experimento SSD50.	408
A.18.	Gráfico de los residuos frente los valores previstos de <i>IPSOM</i> total, experimento SSD50.	409
A.19.	Gráfico de residuos frente al orden de ejecución de las pruebas, experimento SSD50.	409
A.20.	Gráfico de medias para el factor Select_Type, experimento SSD50.	411
A.21.	Gráfico de medias para el factor Restart, experimento SSD50.	411
A.22.	Gráfico de medias para el factor Mut_Prob, experimento SSD50.	412
A.23.	Gráfico de medias para el factor Cross_Type, experimento SSD50.	412
A.24.	Gráfico de medias para el factor Cross_Prob, experimento SSD50.	413
A.25.	Gráfico de medias para el factor Pop_Size, experimento SSD50.	413
A.26.	Gráfico de probabilidad Normal, experimento SSD100.	414
A.27.	Gráfico de los residuos frente a los niveles del factor Cross_Prob, experimento SSD100.	414
A.28.	Gráfico de los residuos frente a los niveles del factor Cross_Type, experimento SSD100.	415
A.29.	Gráfico de los residuos frente a los niveles del factor Mut_Prob, experimento SSD100.	415
A.30.	Gráfico de los residuos frente a los niveles del factor Pop_Size, experimento SSD100.	416
A.31.	Gráfico de los residuos frente a los niveles del factor Restart, experimento SSD100.	416
A.32.	Gráfico de los residuos frente a los niveles del factor Select_Type, experimento SSD100.	417

A.33. Gráfico de los residuos frente los valores previstos de <i>IPSOM</i> total, experimento SSD100.	417
A.34. Gráfico de residuos frente al orden de ejecución de las pruebas, experimento SSD100.	418
A.35. Gráfico de medias para el factor <code>Select_Type</code> , experimento SSD100.	420
A.36. Gráfico de medias para el factor <code>Restart</code> , experimento SSD100.	420
A.37. Gráfico de medias para el factor <code>Mut_Prob</code> , experimento SSD100.	421
A.38. Gráfico de medias para el factor <code>Cross_Type</code> , experimento SSD100.	421
A.39. Gráfico de medias para el factor <code>Cross_Prob</code> , experimento SSD100.	422
A.40. Gráfico de medias para el factor <code>Pop_Size</code> , experimento SSD100.	422
A.41. Gráfico de probabilidad Normal, experimento SSD125.	423
A.42. Gráfico de los residuos frente a los niveles del factor <code>Cross_Prob</code> , experimento SSD125.	423
A.43. Gráfico de los residuos frente a los niveles del factor <code>Cross_Type</code> , experimento SSD125.	424
A.44. Gráfico de los residuos frente a los niveles del factor <code>Mut_Prob</code> , experimento SSD125.	424
A.45. Gráfico de los residuos frente a los niveles del factor <code>Pop_Size</code> , experimento SSD125.	425
A.46. Gráfico de los residuos frente a los niveles del factor <code>Restart</code> , experimento SSD125.	425
A.47. Gráfico de los residuos frente a los niveles del factor <code>Select_Type</code> , experimento SSD125.	426
A.48. Gráfico de los residuos frente los valores previstos de <i>IPSOM</i> total, experimento SSD125.	426
A.49. Gráfico de residuos frente al orden de ejecución de las pruebas, experimento SSD125.	427

A.50. Gráfico de medias para el factor <code>Select_Type</code> , experimento SSD125.	429
A.51. Gráfico de medias para el factor <code>Restart</code> , experimento SSD125.	429
A.52. Gráfico de medias para el factor <code>Mut_Prob</code> , experimento SSD125.	430
A.53. Gráfico de medias para el factor <code>Cross_Type</code> , experimento SSD125.	430
A.54. Gráfico de medias para el factor <code>Cross_Prob</code> , experimento SSD125.	431
A.55. Gráfico de medias para el factor <code>Pop_Size</code> , experimento SSD125.	431
A.56. Gráfico de medias para el factor <code>Select_Type</code> , experimento SSD50_P13.	433
A.57. Gráfico de medias para el factor <code>Mut_Prob</code> , experimento SSD50_P13.	433
A.58. Gráfico de medias para el factor <code>Cross_Type</code> , experimento SSD50_P13.	434
A.59. Gráfico de medias para el factor <code>Pop_Size</code> , experimento SSD50_P13.	434
A.60. Gráfico de medias para el factor <code>Cross_Prob</code> , experimento SSD50_P13.	435
A.61. Gráfico de medias para la interacción entre los factores <code>Cross_Prob</code> y <code>Select_Type</code> , que sirve para clarificar el anterior gráfico, experimento SSD50_P13.	435
A.62. Gráfico de medias para el factor <code>Restart</code> , experimento SSD50_P13.	436
A.63. Gráfico de medias para el factor <code>Select_Type</code> , experimento SSD100_P13.	438
A.64. Gráfico de medias para el factor <code>Mut_Prob</code> , experimento SSD100_P13.	438
A.65. Gráfico de medias para el factor <code>Cross_Type</code> , experimento SSD100_P13.	439

A.66. Gráfico de medias para el factor Pop_Size, experimento SSD100_P13.	439
A.67. Gráfico de medias para el factor Cross_Prob, experimento SSD100_P13.	440
A.68. Gráfico de medias para el factor Restart, experimento SSD100_P13.	440
A.69. Gráfico de medias para el factor Select_Type, experimento SSD125_P13.	442
A.70. Gráfico de medias para el factor Mut_Prob, experimento SSD125_P13.	442
A.71. Gráfico de medias para el factor Cross_Type, experimento SSD125_P13.	443
A.72. Gráfico de medias para el factor Pop_Size, experimento SSD125_P13.	443
A.73. Gráfico de medias para el factor Cross_Prob, experimento SSD125_P13.	444
A.74. Gráfico de medias para el factor Restart, experimento SSD125_P13.	444
A.75. Gráfico de medias para el factor Mut_Prob, experimento SSD10_P2.	446
A.76. Gráfico de medias para el factor Select_Type, experimento SSD10_P2.	446
A.77. Gráfico de medias para el factor Cross_Prob, experimento SSD10_P2.	447
A.78. Gráfico de medias para la interacción entre los factores Pop_Size y Select_Type, experimento SSD10_P2.	447
A.79. Gráfico de medias para el factor Cross_Type, experimento SSD10_P2.	448
A.80. Gráfico de medias para el factor Restart, experimento SSD10_P2.	448
A.81. Gráfico de medias para el factor Mut_Prob, experimento SSD50_P2.	450

A.82. Gráfico de medias para el factor <i>Select_Type</i> , experimento SSD50_P2.	450
A.83. Gráfico de medias para el factor <i>Cross_Prob</i> , experimento SSD50_P2.	451
A.84. Gráfico de medias para la interacción entre los factores <i>Pop_Size</i> y <i>Select_Type</i> , experimento SSD50_P2.	451
A.85. Gráfico de medias para el factor <i>Cross_Type</i> , experimento SSD50_P2.	452
A.86. Gráfico de medias para el factor <i>Restart</i> , experimento SSD50_P2.	452
A.87. Gráfico de medias para el factor <i>Mut_Prob</i> , experimento SSD100_P2.	454
A.88. Gráfico de medias para el factor <i>Select_Type</i> , experimento SSD100_P2.	454
A.89. Gráfico de medias para el factor <i>Cross_Prob</i> , experimento SSD100_P2.	455
A.90. Gráfico de medias para el factor <i>Pop_Size</i> , experimento SSD100_P2.	455
A.91. Gráfico de medias para el factor <i>Cross_Type</i> , experimento SSD100_P2.	456
A.92. Gráfico de medias para el factor <i>Restart</i> , experimento SSD100_P2.	456
A.93. Gráfico de medias para el factor <i>Mut_Prob</i> , experimento SSD125_P2.	458
A.94. Gráfico de medias para el factor <i>Select_Type</i> , experimento SSD125_P2.	458
A.95. Gráfico de medias para el factor <i>Cross_Prob</i> , experimento SSD125_P2.	459
A.96. Gráfico de medias para el factor <i>Pop_Size</i> , experimento SSD125_P2.	459
A.97. Gráfico de medias para el factor <i>Cross_Type</i> , experimento SSD125_P2.	460

A.98. Gráfico de medias para el factor Restart, experimento SSD125_P2.	460
A.99. Gráfico de medias para el factor Mut_Prob, experimento SSD10_P3.	462
A.100. Gráfico de medias para la interacción entre los factores Pop_Size y Select_Type, experimento SSD10_P3.	462
A.101. Gráfico de medias para el factor Cross_Prob, experimento SSD10_P3.	463
A.102. Gráfico de medias para el factor Cross_Type, experimento SSD10_P3.	463
A.103. Gráfico de medias para la interacción entre los factores Restart y Select_Type, experimento SSD10_P3.	464
A.104. Gráfico de medias para el factor Mut_Prob, experimento SSD50_P3.	466
A.105. Gráfico de medias para el factor Select_Type, experimento SSD50_P3.	466
A.106. Gráfico de medias para el factor Pop_Size, experimento SSD50_P3.	467
A.107. Gráfico de medias para el factor Cross_Type, experimento SSD50_P3.	467
A.108. Gráfico de medias para el factor Cross_Prob, experimento SSD50_P3.	468
A.109. Gráfico de medias para el factor Restart, experimento SSD50_P3.	468
A.110. Gráfico de medias para el factor Mut_Prob, experimento SSD100_P3.	470
A.111. Gráfico de medias para el factor Pop_Size, experimento SSD100_P3.	470
A.112. Gráfico de medias para el factor Cross_Prob, experimento SSD100_P3.	471
A.113. Gráfico de medias para el factor Cross_Type, experimento SSD100_P3.	471

A.114. Gráfico de medias para la interacción entre los factores Pop_Size y Select_Type, experimento SSD100_P3.	472
A.115. Gráfico de medias para el factor Restart, experimento SSD100_P3.	472
A.116. Gráfico de medias para el factor Mut_Prob, experimento SSD125_P3.	474
A.117. Gráfico de medias para el factor Pop_Size, experimento SSD125_P3.	474
A.118. Gráfico de medias para el factor Cross_Type, experimento SSD125_P3.	475
A.119. Gráfico de medias para el factor Select_Type, experimento SSD125_P3.	475
A.120. Gráfico de medias para el factor Cross_Prob, experimento SSD125_P3.	476
A.121. Gráfico de medias para el factor Restart, experimento SSD125_P3.	476
C.1. Herramienta de programación de la producción ProdPlanner. Pantalla de gestión de problemas ejemplo y secuencias.	509
C.2. Herramienta de programación de la producción ProdPlanner. Pantalla de configuración y selección de criterio de parada.	510
C.3. Herramienta de programación de la producción ProdPlanner. Pantalla de selección de métodos de resolución.	511
C.4. Herramienta de programación de la producción ProdPlanner. Diagrama de Gantt con la programación de las tareas para un ejemplo.	512
D.1. Ejemplo de azulejo para revestimiento.	516
D.2. Ejemplo de pavimento de gres.	517
D.3. Ejemplo de pavimento de gres porcelánico.	518
D.4. Ejemplo de baldosín catalán.	519
D.5. Ejemplo de gres rústico.	520
D.6. Ejemplo de barro cocido (aplicación).	521

ÍNDICE DE TABLAS

2.1. Tiempos de proceso para un ejemplo de problema de taller de flujo con 4 trabajos y 3 máquinas.	27
3.1. Tiempos de proceso para un ejemplo de problema de taller de flujo con 6 trabajos y 2 máquinas.	44
3.2. Ejecución completa del algoritmo basado en la regla de Johnson para un problema con 6 trabajos.	45
3.3. Tiempos de proceso para el ejemplo de las reglas de prioridad. . .	48
3.4. Tiempos de proceso para un ejemplo de problema de taller de flujo con 4 máquinas y 5 trabajos.	51
3.5. Evaluación de las prestaciones del modelo de Wagner con LINGO v. 6.0.	56
3.6. Heurísticas constructivas y de mejora para el problema del taller de flujo.	89
3.7. Metaheurísticas para el problema del taller de flujo.	90
3.8. Número de trabajos (n), máquinas (m), máximas cotas inferiores (CI) y mínimas cotas superiores (CS) conocidas para el “ <i>benchmark</i> ” de Taillard.	95
3.9. Incremento porcentual medio por encima del óptimo o mínima cota superior conocida (<i>IPSONM</i>) de los métodos heurísticos constructivos y de mejora para los problemas de Taillard.	101
3.10. Tiempos de proceso (en segundos) utilizados por los métodos heurísticos constructivos y de mejora para los problemas de Taillard.	104

3.11. Incremento porcentual medio por encima del óptimo o mínima cota superior conocida (<i>IPSOM</i>) de los métodos metaheurísticos para los problemas de Taillard.	106
3.12. Comparación entre los métodos metaheurísticos ILS y GAReev para los problemas de Taillard.	108
3.13. Tiempos de proceso (en segundos) utilizados por los métodos metaheurísticos para los problemas de Taillard.	109
4.1. Estudio de las soluciones y valores de C_{max} para los problemas de Carlier (car1-car8).	164
4.2. Tabla ANOVA con los resultados del experimento del GA propuesto.	176
4.3. Incremento porcentual medio por encima del óptimo o mínima cota superior conocida (<i>IPSOM</i>) de los mejores métodos heurísticos, metaheurísticos y los algoritmos genéticos propuestos. . .	191
4.4. Comparación entre los métodos metaheurísticos ILS y GA para los problemas de Taillard.	192
4.5. Comparación entre los métodos metaheurísticos ILS y HGA para los problemas de Taillard.	193
4.6. Tiempos de proceso (en segundos) utilizados por los mejores métodos metaheurísticos y los algoritmos genéticos propuestos. .	194
4.7. <i>IPSOM</i> de los mejores métodos heurísticos, metaheurísticos y los algoritmos genéticos propuestos. Criterio de parada fijado a $(n \cdot 60000)/1000$ milisegundos.	196
4.8. Comparación entre los métodos metaheurísticos ILS y GA para los problemas de Taillard. Criterio de parada fijado a $(n \cdot 60000)/1000$ milisegundos.	197
4.9. Comparación entre los métodos metaheurísticos ILS y HGA para los problemas de Taillard. Criterio de parada fijado a $(n \cdot 60000)/1000$ milisegundos.	198
5.1. Métodos exactos, heurísticos y metaheurísticos para el problema del taller SDST.	217

5.2.	Tabla ANOVA con los resultados del experimento del algoritmo GA_{sd} propuesto, experimento SSD10.	232
5.3.	Resultado de los experimentos y de la parametrización del algoritmo GA_{sd} propuesto.	241
5.4.	<i>IPSOM</i> y tiempo de CPU en segundos (entre paréntesis) de los métodos metaheurísticos adaptados y los nuevos algoritmos genéticos propuestos para el taller SDST, grupo de problemas SSD10.	245
5.5.	<i>IPSOM</i> y tiempo de CPU en segundos (entre paréntesis) de los métodos heurísticos y metaheurísticos para el taller SDST, grupo de problemas SSD10.	246
5.6.	Comparación entre los métodos metaheurísticos ILS y GA_{sd} para el grupo de problemas SSD10.	249
5.7.	<i>IPSOM</i> y tiempo de CPU en segundos (entre paréntesis) de los métodos metaheurísticos adaptados y los algoritmos genéticos propuestos para el taller SDST, grupo de problemas SSD50. . . .	250
5.8.	<i>IPSOM</i> y tiempo de CPU en segundos (entre paréntesis) de los métodos heurísticos y metaheurísticos para el taller SDST, grupo de problemas SSD50.	251
5.9.	<i>IPSOM</i> y tiempo de CPU en segundos (entre paréntesis) de los métodos metaheurísticos adaptados y los algoritmos genéticos propuestos para el taller SDST, grupo de problemas SSD100. . . .	252
5.10.	<i>IPSOM</i> y tiempo de CPU en segundos (entre paréntesis) de los métodos heurísticos y metaheurísticos para el taller SDST, grupo de problemas SSD100.	253
5.11.	<i>IPSOM</i> y tiempo de CPU en segundos (entre paréntesis) de los métodos metaheurísticos adaptados y los algoritmos genéticos propuestos para el taller SDST, grupo de problemas SSD125. . . .	254
5.12.	<i>IPSOM</i> y tiempo de CPU en segundos (entre paréntesis) de los métodos heurísticos y metaheurísticos para el taller SDST, grupo de problemas SSD125.	255
5.13.	Resultados de los tres mejores métodos para los cuatro grupos de problemas.	256

5.14. <i>IPSOM</i> de los métodos metaheurísticos para el taller SDST, criterio de parada establecido en $(n \cdot 60000)/1000$ milisegundos, grupo de problemas SSD10.	258
5.15. <i>IPSOM</i> de los métodos metaheurísticos para el taller SDST, criterio de parada establecido en $(n \cdot 60000)/1000$ milisegundos, grupo de problemas SSD50.	259
5.16. <i>IPSOM</i> de los métodos metaheurísticos para el taller SDST, criterio de parada establecido en $(n \cdot 60000)/1000$ milisegundos, grupo de problemas SSD100.	260
5.17. <i>IPSOM</i> de los métodos metaheurísticos para el taller SDST, criterio de parada establecido en $(n \cdot 60000)/1000$ milisegundos, grupo de problemas SSD125.	261
5.18. Resultados de los tres mejores métodos para los cuatro grupos de problemas, criterio de parada establecido en $(n \cdot 60000)/1000$ milisegundos.	262
6.1. Tipos de baldosas producidos en España.	271
6.2. Métodos para el problema del taller de flujo híbrido.	319
6.3. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD10_P13.	333
6.4. Resultado de los 12 experimentos y de la calibración del algoritmo GA_H propuesto.	343
6.5. <i>IPSOM</i> de los métodos propuestos para el problema de la pro- gramación de la producción en el sector cerámico. Grupo de pro- blemas SSD10_P13.	346
6.6. <i>IPSOM</i> de los métodos propuestos para el problema de la pro- gramación de la producción en el sector cerámico. Grupo de pro- blemas SSD50_P13.	348
6.7. <i>IPSOM</i> de los métodos propuestos para el problema de la pro- gramación de la producción en el sector cerámico. Grupo de pro- blemas SSD100_P13.	349

6.8. <i>IPSONM</i> de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD125_P13.	350
6.9. <i>IPSONM</i> de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P2.	351
6.10. <i>IPSONM</i> de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD50_P2.	352
6.11. <i>IPSONM</i> de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD100_P2.	353
6.12. <i>IPSONM</i> de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD125_P2.	354
6.13. <i>IPSONM</i> de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P3.	355
6.14. <i>IPSONM</i> de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD50_P3.	356
6.15. <i>IPSONM</i> de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD100_P3.	357
6.16. <i>IPSONM</i> de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD125_P3.	358
6.17. Tiempos de proceso (en segundos) utilizados por los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P13.	359
6.18. Tiempos de proceso (en segundos) utilizados por los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P2.	360

6.19. Tiempos de proceso (en segundos) utilizados por los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P3.	361	
6.20. Resultados del algoritmo genético propuesto y la programación manual para los diez ejemplos reales de producción.	363	
A.1. Tabla ANOVA con los resultados del experimento para el algoritmo GA_{sd} , conjunto de datos SSD50.		410
A.2. Tabla ANOVA con los resultados del experimento para el algoritmo GA_{sd} , conjunto de datos SSD100.		419
A.3. Tabla ANOVA con los resultados del experimento para el algoritmo GA_{sd} , conjunto de datos SSD125.		428
A.4. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD50_P13.		432
A.5. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD100_P13.		437
A.6. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD125_P13.		441
A.7. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD10_P2.		445
A.8. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD50_P2.		449
A.9. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD100_P2.		453
A.10. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD125_P2.		457
A.11. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD10_P3.		461
A.12. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD50_P3.		465
A.13. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD100_P3.		469

A.14. Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD125_P3.	473
B.1. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD10.	479
B.2. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD50.	481
B.3. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD100.	483
B.4. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD125.	484
B.5. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD10_P13.	487
B.6. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD50_P13.	488
B.7. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD100_P13.	490
B.8. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD125_P13.	492
B.9. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD10_P2.	494
B.10. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD50_P2.	495
B.11. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD100_P2.	497
B.12. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD125_P2.	499
B.13. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD10_P3.	501
B.14. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD50_P3.	502
B.15. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD100_P3.	504

B.16. Número de trabajos (n), máquinas (m), y mejores soluciones para el grupo de problemas SSD125_P3.	506
C.1. Contenido del CD-ROM anexo.	513

ÍNDICE DE LISTADOS

3.1.	Programa en lenguaje LINGO para el modelo de Wagner.	54
3.2.	Pseudocódigo de la metaheurística general ILS.	84
3.3.	Problema ta001 de Taillard.	96
4.1.	Pseudoalgoritmo del operador de selección por ruleta o selección simple.	127
4.2.	Pseudoalgoritmo con el funcionamiento de la fase de cruce genético.	128
4.3.	Pseudoalgoritmo con el funcionamiento de la fase de mutación genética.	141
4.4.	Proceso de inicialización para el GA propuesto en forma de pseudoalgoritmo.	146
5.1.	Proceso de inicialización para el GA_{sd}	220
5.2.	Problema ta001 de Taillard aumentado con los tiempos de cambio de partida dependientes de la secuencia al 10 % de los tiempos de proceso. Problema ta001_SSD10.	224
6.1.	Proceso de inicialización para el GA_H	326
6.2.	Problema ta007_SSD10_P13.	329

1

CAPÍTULO

INTRODUCCIÓN

Actualmente existe un gran mercado internacional en el que todas las empresas, con independencia de su localización geográfica, compiten en un entorno altamente variable. Para mantener su posición, las organizaciones ofrecen unos extensos catálogos de productos con un ciclo de vida muy corto y con frecuentes cambios en las especificaciones. Asimismo, las exigencias cada vez más refinadas de los clientes derivan en pedidos que van disminuyendo en tamaño, pero aumentando en frecuencia y con unos tiempos de entrega muy reducidos. Adicionalmente, la fuerte presión de la competencia obliga a mantener unos precios muy ajustados así como una clara diferenciación de los productos y/o servicios ofrecidos por las empresas.

Esta situación requiere cada vez más de una mayor flexibilidad por parte de las organizaciones, lo que contrasta fuertemente con los sistemas productivos utilizados en la mayoría de las ocasiones, donde comúnmente existen líneas de fabricación de alto rendimiento, preparadas para fabricar grandes lotes de productos a bajo precio, pero con el inconveniente de que cambios en la producción requieren normalmente de ajustes en las líneas con la consiguiente pérdida de tiempo y

de capacidad e incrementando con ello los costes.

En este contexto de economía global, la empresa debe conjugar cuidadosamente un amplio catálogo de productos innovadores y diferenciados con un sistema de producción flexible que permita al mismo tiempo una rapidez en la entrega y unos costes bajos para asegurar la completa satisfacción del cliente. Es evidente que con estas premisas se hace necesario un uso racional y eficiente de los recursos productivos.

Es posible diferenciar dos tipos de sistemas de producción (Johnson y Montgomery, 1974). El primer tipo son los sistemas de producción en *continuo*, donde existen pocas o muy pocas familias de productos que además son parecidas entre sí y los volúmenes de producción son elevados. Tal es el ejemplo de producción de acero, o de hormigón. El segundo tipo son los sistemas de producción *intermitentes* donde normalmente hay un gran número de productos diferentes a producir y las líneas de producción deben ajustarse en cada caso, lo que representa una mayor dificultad de operación y de control. Ejemplos claros de sistemas de producción intermitentes los encontramos en el sector textil, del automóvil o el azulejero, entre otros.

En la industria, y con independencia del tipo de sistema productivo, existen una serie de decisiones que es necesario tomar (Johnson y Montgomery, 1974 y Piñedo, 2002) y que dependen del horizonte temporal en el que estas decisiones se tomen. A largo plazo una empresa debe decidir qué productos debe ofrecer y a qué segmento del mercado van dirigidos. Asimismo, debe determinar qué políticas de servicio va a seguir y qué capacidad productiva y de almacenamiento va a disponer para lograr los objetivos empresariales fijados. A medio plazo, la compañía deberá planificar mano de obra y materiales para poder hacer frente a los pedidos de los clientes. La empresa, en un proceso llamado Planificación de la Producción o “*Production Planning*” desarrollará un *Plan Maestro de Producción* o “*Master Plan Schedule*”. Este plan sirve para establecer una política de producción a medio plazo y, por tanto, dimensionar la plantilla fija, los turnos a realizar y las necesidades de recursos financieros para soportar stocks o invertir en maquinaria adicional. El plan Maestro de Producción plantea una hipótesis de trabajo que va refinándose en las decisiones que se toman a corto plazo. En este último paso y a partir del plan maestro, se desarrolla una Planificación de la Producción o “*Production*

Scheduling" que da como resultado un *programa* o "schedule". El programa se compone de una asignación de las órdenes de producción a los centros productivos y de un calendario de fabricación (Companys Pascual y Corominas Subias, 1996). Los distintos pedidos del plan se *secuencian*, es decir, se asignan a un centro de trabajo y se establece su orden de ejecución dentro del mismo. De manera casi simultánea se determinan los instantes de comienzo y de finalización para cada operación de fabricación. En la Figura 1.1 se muestra un diagrama de bloques con las etapas y decisiones que se toman en un proceso productivo.

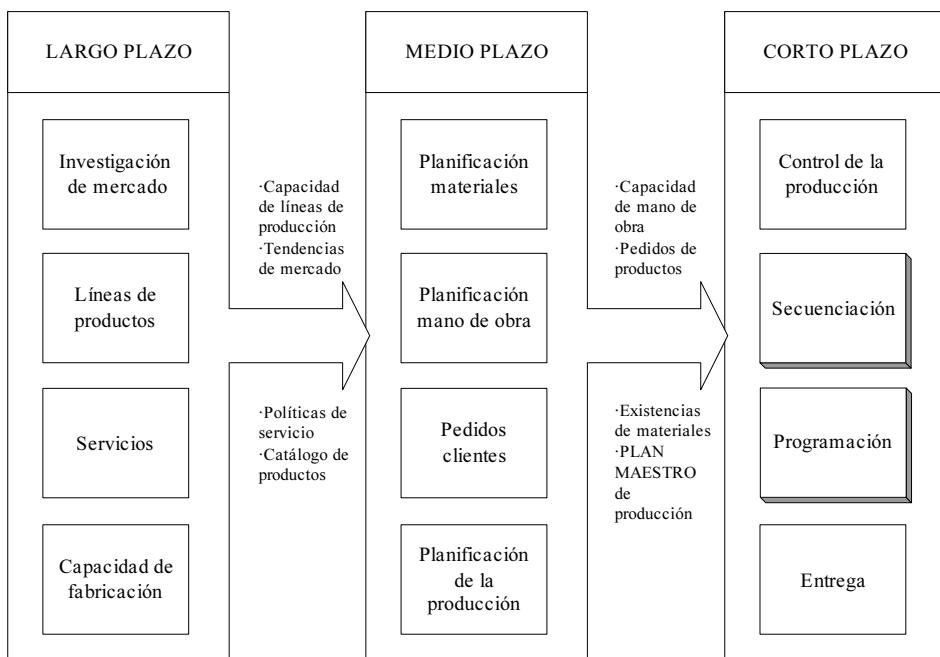


Figura 1.1 – Problemas de decisión de los procesos productivos.

Evidentemente, la planificación a largo plazo es esencial para la empresa y se puede ver como la estrategia de mercado. De igual manera, la planificación a medio plazo es de suma importancia, ya que, por poner un ejemplo, una mala previsión de materiales puede crear serios problemas a la hora de realizar la producción. De acuerdo con Thomas y McClain (1993), la secuenciación o "*sequencing*" y la

programación o “*scheduling*” son una parte muy importante de la organización y deben tenerse en cuenta explícitamente y de manera adecuada para la consecución de los objetivos. Estos dos problemas pertenecen a las decisiones que se toman al corto plazo y frecuentemente se han considerado menos importantes que los problemas de la planificación a largo o medio plazo. No obstante, y como se ha comentado, la reducción en los tamaños de lote y el amplio catálogo de productos que las empresas tienen hacen que realizar una programación de la producción eficiente sea cada vez más difícil, sobre todo en los entornos productivos de tipo intermitente. Actualmente existen varias metodologías de carácter general que resultan útiles para la planificación y programación de la producción, un ejemplo son los sistemas MRP o “*Material Requirement Planning*” y MRP II o “*Manufacturing Resource Planning*”. Estas aproximaciones han sido ampliamente criticadas por ser excesivamente generales y por no ser capaces de producir programas realistas (Artiba y Elmaghraby, 1997, Drexl y Kimms, 1998 y Moursli, 1999). Otro método más reciente, como es el paradigma “*Just In Time*” (JIT) requiere una demanda estable, productos estándar y una alta capacidad de producción, lo cual no siempre es posible, y por lo que hemos visto, no se ajusta a la situación actual del mercado. Otro sistema más elaborado es el “*Advanced Planner and Optimizer*” (APO) de la empresa SAP (<http://www.sap.com/>). Aunque muchos de estas técnicas y metodologías tienen excelentes capacidades desde el punto de vista de la planificación de la producción, son muy generalistas y con frecuencia proporcionan programaciones de la producción a capacidad infinita o no llegan siquiera a proporcionar secuencias de producción. De ahí que su implantación y utilización en la práctica no esté extendida.

1.1. Programación de la producción

La programación de la producción (“*Production Scheduling*”) engloba las fases de secuenciación y programación y por simplicidad se entiende por “*scheduling*” o secuenciación a toda esta etapa de programación de la producción. La secuenciación comporta la asignación de *recursos* (normalmente escasos) a lo largo del tiempo para la realización de un conjunto de *tareas*. Los recursos pueden ser máquinas en una línea de producción, mano de obra, procesadores en

un multicomputador o incluso salas de operación en un hospital. Las tareas pueden ser pedidos de los clientes para fabricación, operaciones a realizar en el hospital o programas de ordenador. La secuenciación supone que la planificación de la producción ya se ha realizado, es decir, se conocen con certeza los productos a producir y sus correspondientes cantidades así como la configuración, disposición y cantidad de recursos disponibles. Una adecuada secuenciación de la producción es vital para el funcionamiento y resultados de una empresa. Los objetivos que se persiguen con la secuenciación son muy variados, por ejemplo la minimización de la cantidad de producto en curso, o la minimización del número de pedidos que se finalizan después de la fecha de entrega pactada con el cliente. En el contexto de la presente Tesis Doctoral y sin pérdida de generalidad nos referiremos a los distintos recursos como *máquinas*. Una máquina se puede entender como tal o como un grupo de recursos que se modelizan como una unidad. Asimismo cada pedido de producción se denominará *trabajo*. Un trabajo se descompone en varias *tareas u operaciones* dependiendo de la configuración productiva y del número de máquinas disponibles. En este contexto podemos concretar la definición anterior de “*scheduling*” como la asignación de las tareas a máquinas (asignación), la ordenación de las tareas asignadas a cada máquina (secuenciación) y la posterior obtención de los tiempos de inicio y finalización de cada tarea (programación). Llegados a este punto es posible hacer una primera clasificación de los distintos problemas de secuenciación atendiendo a la configuración productiva (Portmann, 1997):

- Problemas de una máquina: en este caso sólo existe un único recurso o máquina y cada trabajo tiene una única operación. No existe el problema de asignación pero sí el de secuenciación.
- Problemas de máquinas paralelas: existen varias máquinas (> 1) dispuestas en paralelo donde los trabajos siguen teniendo una única operación. Si el número de máquinas es superior al número de trabajos no existe el problema de secuenciación pero sí el de asignación. En el caso general se tienen que asignar y secuenciar los trabajos en las máquinas. Los problemas de máquinas paralelas pueden asimismo dividirse atendiendo a la naturaleza de las distintas máquinas existentes:

- Máquinas idénticas (“*identical parallel machines*”): como el nombre indica, se trata de máquinas con iguales características productivas, es decir, procesan los trabajos a la misma velocidad.
 - Máquinas proporcionales o uniformes (“*uniform parallel machines*”): dentro del grupo de máquinas existen algunas que son capaces de procesar todos los trabajos a mayor o menor velocidad que las demás.
 - Máquinas no relacionadas (“*unrelated parallel machines*”): la velocidad de proceso depende de la máquina y de la tarea que se está procesando. Se trata de un caso más general que engloba a los dos anteriores.
- Problemas de tipo taller o “*shop*”. En estos problemas, que resultan ser más realistas, se dispone de varias máquinas (no paralelas) por lo que cada trabajo tiene (en general) tantas operaciones como máquinas. Adicionalmente existe una relación de precedencia entre las tareas para cada trabajo. A esta relación se la denomina *ruta*. Los problemas de tipo taller se subdividen de acuerdo a las características de las rutas en:
- Taller abierto o “*Open Shop*”: no existe ninguna restricción en la ruta. Las operaciones se pueden realizar en cualquier orden.
 - Taller de flujo o “*Flow Shop*”: todos los trabajos tienen la misma ruta en las máquinas. Es decir, existe una ordenación predeterminada de las tareas que además es la misma en todos los trabajos.
 - Taller de trabajo o “*Job Shop*”: cada trabajo tiene su propia ruta a seguir en las máquinas.

De los talleres anteriores, es frecuente el taller de flujo o “*Flow Shop*”. Un ejemplo se puede encontrar en el proceso de producción de azulejos o baldosas cerámicas. El proceso comienza con la preparación de materias primas y prensado de la arcilla, tras el cual se obtiene lo que se conoce como bizcocho verde, que es el azulejo “blando” sin esmaltar. El bizcocho se somete a una fase de secado para endurecerlo y después pasa por la línea de esmaltado donde se aplican las capas de color y las decoraciones. Una vez esmaltado, el azulejo se introduce en un horno

especial en lo que se conoce como fase de cocción. Una vez terminado el proceso, el azulejo se inspecciona visualmente o por medios automáticos para buscar defectos y se clasifica de acuerdo a calidades, tonos y calibres. Esta última fase se conoce como etapa de clasificación. Normalmente todos los azulejos siguen el mismo proceso productivo; las operaciones de prensado, secado, esmaltado, cocción y clasificación deben hacerse siempre en este orden, y el orden es el mismo para todos los tipos de azulejos. Un esquema simplificado puede verse en la Figura 1.2.

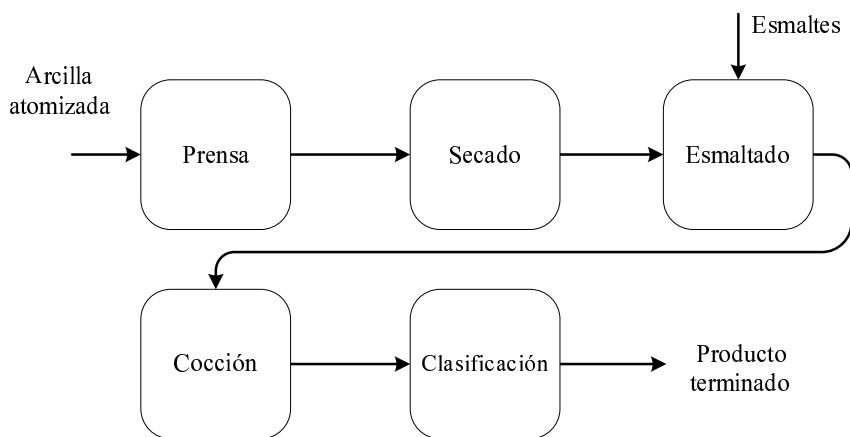


Figura 1.2 – Esquema simplificado del proceso de producción de baldosas cerámicas; un ejemplo de taller de flujo o “*flow shop*”.

Siguiendo este mismo ejemplo de producción de azulejos podemos tomar conciencia de la importancia de obtener programas o “*schedules*” de producción adecuados. El seguir una secuencia no óptima de producción en la prensa puede traducirse en la necesidad de cambiar varias veces el molde y perder tiempo realizando ajustes en la prensa. Sin embargo, distintos trabajos (lotes de azulejos) pueden tener el mismo tamaño (lo que se conoce como formato) por lo que no sería necesario cambiar el molde de la prensa si se mantienen juntos estos lotes en la secuencia de la prensa. De esta manera conseguiremos terminar la producción de los mismos mucho antes y malgastar menos recursos de mano de obra y de

maquinaria.

Podemos encontrar situaciones más complejas (y realistas) como puede ser el caso de un taller de flujo donde en alguna de las etapas productivas puede existir un grupo de máquinas paralelas, o siguiendo el ejemplo, para prensar los azulejos podríamos disponer de varias prensas dispuestas en paralelo. En estos contextos, por buena que sea la planificación de la producción, una programación ineficaz puede resultar en una mala utilización de los recursos productivos, elevados costes de producción, retrasos en las fechas de entrega y mal servicio al cliente.

El taller de flujo es, de los tres tipos de talleres anteriormente citados, el que más profusamente ha sido estudiado en la literatura (Dudek, Panwalkar y Smith, 1992). Sin embargo, este tipo de taller es un problema de naturaleza combinatoria que pertenece al conjunto de problemas \mathcal{NP} -Completo en sentido estricto (ver Garey, Johnson y Sethi, 1976), lo que quiere decir que no existen técnicas que proporcionen soluciones óptimas para el problema general en un tiempo “aceptable”.

El siguiente ejemplo puede servir para hacerse una idea de lo difícil que puede llegar a ser este problema. Supongamos que, siguiendo el esquema de producción de azulejos, tenemos que fabricar un total de 10 tipos de azulejos en las 5 máquinas de la Figura 1.2. El número total de posibles secuencias de producción diferentes para la prensa viene dado por todas las posibles ordenaciones o permutaciones de los tipos de azulejos en esta misma máquina, es decir, $10! = 3.628.800$. El mismo resultado es aplicable a todas las máquinas, es decir, hay $10!$ posibles secuencias para cada máquina, lo que nos da un total de $(10!)^5 = 6,29 \cdot 10^{32}$ posibles secuencias. Una técnica sencilla de obtener la solución óptima podría consistir en evaluar todas y cada una de las secuencias con algún objetivo prefijado y quedarnos con la mejor de todas. Supongamos que disponemos de un ordenador muy rápido que es capaz de calcular una posible secuencia en un nanosegundo (1^{-9} segundos), o 1000 millones de secuencias por segundo (un ordenador de este tipo no existe actualmente), luego evaluar todas las posibles secuencias nos llevaría $(10!)^5 \cdot 1^{-9} = 6,29 \cdot 10^{23}$ segundos, que son $1,99 \cdot 10^{16}$ años, o lo que es lo mismo ¡casi 20.000 billones de años! Recientemente, la NASA, utilizando el telescopio espacial Hubble, ha estimado la vida del universo con bastante precisión entre 13.000 y 14.000 millones de años, luego el tiempo que necesitaríamos para

resolver el sencillo problema propuesto necesitaría aproximadamente un millón y medio de “vidas del universo”. Es inmediato ver que para un número de trabajos y de máquinas mayor (por ejemplo 150 trabajos y 25 máquinas) ni siquiera es posible hacer el cálculo anterior.

A pesar de la enorme complejidad del problema, existen numerosas técnicas y trabajos publicados que aportan soluciones satisfactorias para el problema del taller de flujo. Uno de los primeros trabajos para este problema se debe a Johnson (1954) que propuso un algoritmo sencillo que da como resultado una secuenciación óptima para el flowshop, pero solo en el caso en que únicamente existen dos máquinas. En la década de los 60 y primeros 70 aparecieron numerosas contribuciones en el campo de técnicas exactas como la programación dinámica o los algoritmos de bifurcación y acotación. Estas aportaciones, como se verá en posteriores capítulos, tan solo son capaces de proporcionar soluciones para problemas muy concretos y/o ejemplos de tamaño reducido. Tras la generalización de las ideas de la teoría de la complejidad (uno de los primeros trabajos es el de Garey y Johnson, 1979), la comunidad científica reaccionó y el esfuerzo investigador internacional se centró en técnicas heurísticas. Hacia finales de los 70 y en la década de los 80 aparecieron varios algoritmos de carácter aproximado para resolver el problema del taller de flujo. Desde principios de los 90 hasta nuestros días la atención se ha centrado en técnicas más avanzadas, como son los algoritmos metaheurísticos; algoritmos genéticos, recocido simulado o “*simulated annealing*”, búsqueda tabú o “*tabu search*”, búsqueda local iterativa, GRASP y otros.

Como veremos, el taller de flujo estándar, aun siendo un problema de naturaleza combinatoria y muy difícil de resolver, resulta ser demasiado “simple” en el sentido en que se hacen muchas simplificaciones y supuestos que afectan a la aplicabilidad de los métodos disponibles en un entorno industrial real. Aunque se han desarrollado algoritmos para variaciones más realistas de este problema, las industrias siguen realizando una programación de la producción por métodos manuales y en todo caso, utilizando herramientas que no resuelven el problema en su forma general y completa.

1.2. Objetivos y Metodología

La problemática de la programación de la producción se lleva investigando desde hace muchos años y se han propuesto soluciones que van desde los sencillos diagramas que Henry L. Gantt propuso a principios del siglo 20 hasta las técnicas metaheurísticas más avanzadas y recientes.

La presente Tesis Doctoral se centra en un tipo de problemas muy comunes de programación de la producción, como son los talleres de flujo, que se dan en muchos sectores productivos de gran importancia económica y en particular en la Comunidad Valenciana. Estos problemas, como se ha comentado, se han estado investigando desde hace casi 50 años y aunque se han realizado avances importantes, se reconoce en muchos trabajos que los métodos y técnicas desarrollados no son adecuados para la aplicación en las industrias. Las razones más importantes que justifican esta carencia son las simplificaciones *fuertes* que con tanta frecuencia se realizan en los trabajos científicos y que muchas veces limitan enormemente el ámbito de las conclusiones así como el campo de aplicación de los resultados obtenidos. El objetivo principal de esta Tesis Doctoral es el de desarrollar métodos para la programación flexible de la producción, abarcando desde los problemas más clásicos hasta problemas complejos extraídos del entorno productivo. Con esto buscamos dar soluciones reales a sectores tan importantes como el cerámico o el textil.

En primer lugar se ha realizado una exhaustiva revisión del estado del arte sobre trabajos, técnicas, métodos y algoritmos propuestos para el problema del taller de flujo en su versión de permutación. Dada la gran cantidad de métodos propuestos nos planteamos como primer objetivo el realizar una extensa evaluación comparativa. Para llevar a cabo este objetivo hemos codificado un total de 25 algoritmos diferentes, utilizando el entorno de desarrollo Borland Delphi® versión 6.0. Para realizar la comparativa se utiliza el conocido banco de pruebas de Taillard (1993). Los resultados de la comparativa servirán para identificar los métodos más adecuados para este problema, así como para descartar aquéllos que, por una baja eficacia o eficiencia no resultan aptos para el problema en cuestión.

El segundo objetivo es proponer nuevos métodos para el problema del taller de flujo. Concretamente algoritmos genéticos robustos que incluyan nuevos operadores de cruce, así como nuevas técnicas y características que los hagan capaces de competir con los mejores métodos propuestos en términos de eficacia y eficiencia. Estos nuevos algoritmos se codifican también en Delphi[®] y una característica metodológica importante es la utilización del diseño de experimentos y análisis de la varianza (ANOVA) para la calibración de los operadores y parámetros de los algoritmos genéticos propuestos. Concretamente se propone realizar amplios experimentos donde se contemplen múltiples niveles y/o variantes de los parámetros y operadores, junto con un cuidadoso análisis de los resultados para así conseguir algoritmos más eficaces.

El tercer objetivo es considerar un aspecto muy importante de los problemas de programación de la producción y que se da con frecuencia en las industrias: la existencia de tiempos de cambio de partida o de ajuste en las máquinas que además son dependientes de la secuencia de fabricación. Se verá que este problema no es simplemente una extensión del anterior, sino un problema mucho más complejo y difícil de resolver. Para alcanzar este objetivo se lleva a cabo una completa revisión del estado de la cuestión así como una evaluación comparativa de varios de los métodos propuestos para este problema. Asimismo se proponen dos nuevos algoritmos genéticos implementados también en Delphi[®] y que se calibran también mediante diseño de experimentos. Para la realización de la comparativa se utiliza un conjunto de 480 problemas ejemplo específicamente diseñados para este caso.

El cuarto objetivo es considerar problemas más generales de programación de la producción que contemplan aspectos mucho más complejos como es la existencia de máquinas paralelas no relacionadas en cada etapa de producción, tiempos de cambio de partida dependientes de la secuencia y restricciones de uso de máquinas. Como veremos, estos problemas no son versiones más restringidas de los anteriores, sino más generales ya que incluyen muchas más situaciones que son comunes en las industrias. La extensa revisión del estado del arte realizada ha puesto de manifiesto que este tipo de problemas no han sido considerados ante-

riormente. Proponemos varias adaptaciones de métodos metaheurísticos así como un nuevo algoritmo genético para dar solución a este problema y una evaluación de más de 1300 problemas de ejemplo junto a un amplio diseño experimental encaminado a obtener una precisa parametrización del algoritmo propuesto.

Finalmente, se propone aplicar los métodos desarrollados a entornos reales de producción, concretamente en el sector cerámico. Como veremos, la creciente diversificación y diferenciación de productos y la fuerte competencia han hecho que las empresas de este sector no sean capaces de vender todo lo que producen como ocurría anteriormente. Para asegurar las ventas deben ofrecer un mejor servicio al cliente y un precio muy competitivo, lo que pasa por realizar una eficaz programación de la producción. En este contexto un objetivo que nos planteamos es el diseño de un prototipo de software para la programación de la producción que incluye todos los algoritmos desarrollados en la presente Tesis Doctoral. La finalidad del software es que sea capaz de obtener programas realistas, factibles y al mismo tiempo que los tiempos de cálculo sean reducidos.

1.3. Esquema general de la Tesis Doctoral

El Capítulo 2 presenta la notación para problemas de programación de la producción y unas nociones sobre la teoría de la complejidad para ubicar los problemas objeto de estudio de la presente Tesis Doctoral. En el Capítulo 3 se plantea la problemática del taller de flujo estándar y se proporciona una extensa y completa revisión del estado del arte para este problema. También se presenta una exhaustiva comparativa de métodos de programación de la producción que abarca tanto los métodos heurísticos clásicos como las más recientes técnicas metaheurísticas.

En el Capítulo 4 se desarrollan dos nuevos algoritmos genéticos para el problema del taller de flujo. Asimismo se presenta un amplio diseño experimental y análisis de la varianza para calibrar el conjunto de operadores y parámetros. Finalmente se comparan los algoritmos obtenidos con los mejores métodos y procedimientos presentados en el Capítulo 3.

El Capítulo 5 trata una ampliación del citado problema que considera la existencia

de tiempos de cambio de partida dependientes de la secuencia. Este capítulo muestra una revisión del estado del arte para este caso, y también se presentan dos nuevos algoritmos genéticos basados en los anteriores. Los algoritmos propuestos se calibran igualmente utilizando un diseño experimental. Para concluir el capítulo se realiza una completa evaluación de los métodos existentes para este problema. Un problema más realista se presenta en el Capítulo 6. En este caso, se consideran talleres de flujo híbridos con tiempos de cambio de partida y una serie de restricciones adicionales que se encuentran en los problemas de programación de la producción del sector azulejero. Tras caracterizar el problema de producción del sector y realizar una extensa revisión bibliográfica se propone un nuevo algoritmo genético para resolver el problema. Asimismo también se proponen una serie de adaptaciones de los mejores métodos metaheurísticos evaluados en los Capítulos 3 y 5. Finalmente, en el Capítulo 7 se presentan las conclusiones de la presente Tesis Doctoral.

CAPÍTULO

2

CONCEPTOS PREVIOS

2.1. Notación y clasificación de los problemas de programación de la producción

Un problema de programación de la producción viene determinado por el número de trabajos y operaciones a procesar, por el número y tipo de máquinas que hay disponibles, por el patrón de flujo de las operaciones en las máquinas y por el criterio(s) de optimización con los que se evalúa una secuencia de producción.

Normalmente, en los problemas de programación de la producción se asume un número de trabajos y de máquinas finito y determinista. De acuerdo con Baker (1974) y con Pinedo (2002), en un problema de producción tenemos un conjunto N de trabajos, donde $N = (1, 2, \dots, n)$, que hay que procesar en un conjunto M de máquinas, $M = (1, 2, \dots, m)$. Utilizaremos los subíndices j y k para referirnos a los trabajos y el subíndice i para las máquinas. De esta manera podemos definir los siguientes conceptos y parámetros:

- Tarea u operación (O_{ij}). Cada trabajo $j \in N$ tiene tantas operaciones como máquinas existen (m), luego O_{ij} denota la operación en la máquina i del trabajo j . En otros casos la operación O_{ij} se denota simplemente por (i, j) . Cada operación tiene un instante de comienzo y de finalización como resultado de la programación. Nos referiremos a estos instantes como $O_{ij}s$ y $O_{ij}f$.
- Tiempo de proceso (p_{ij}). Es la cantidad de tiempo que se necesita para procesar el trabajo j en la máquina i , es decir, el tiempo que necesita la operación O_{ij} . Normalmente, después de la programación se debe cumplir que $p_{ij} = O_{ij}f - O_{ij}s$.
- Instantes o fechas de entrada (r_j). Son conocidos como “*release dates*” o “*ready dates*”. Son los tiempos de entrada de los trabajos al taller. Esto implica que $\forall j \in N, O_{ij}s \geq r_j$, donde $i = (1, 2 \dots, m)$. Es decir, ninguna operación del trabajo puede comenzar antes de la fecha de entrada.
- Fechas de finalización (d_j). Son las fechas de compromiso de finalización para cada trabajo. Se permite una finalización posterior, aunque con una penalización. En la literatura se conocen como “*due dates*”.
- Fechas de finalización obligada o “*deadlines*” (\bar{d}_j). En este caso es un compromiso firme de finalización de los trabajos y por tanto $\forall j \in N, O_{ij}f \leq \bar{d}_j$, donde $i = (1, 2 \dots, m)$.
- Prioridades (w_j). La prioridad o peso de un trabajo es una manera de establecer una jerarquía relativa de los trabajos en N . La prioridad se puede establecer en función del coste o beneficio del trabajo, tipo de cliente etc...

Dada la gran variedad de problemas de programación de la producción, motivada en gran parte por las posibles configuraciones productivas, se han propuesto numerosas notaciones que ayudan a clasificar los diferentes problemas atendiendo a sus características. Una de las primeras clasificaciones aparecidas es la de Conway, Maxwell y Miller (1967) que se basa en la expresión cuádruple $A/B/C/D$ donde:

- A indica el número de trabajos a procesar, es decir, el valor concreto de n para un problema dado.
- B indica el número de máquinas o el valor de m para un problema.
- C describe el patrón de flujo de los trabajos dentro del taller. Cuando $C = F$ se trata de un taller de flujo, $C = J$ especifica un “*Job Shop*”, cuando $C = O$ un “*Open Shop*” y cuando $C = P$ se trata de un problema con máquinas paralelas. Si el problema a considerar tiene una única máquina este campo se omite.
- D proporciona el criterio de optimización considerado.

Es fácil observar que la anterior notación no permite especificar muchos de los problemas de programación de la producción, como por ejemplo un taller con máquinas paralelas uniformes o no relacionadas. Debido a esto existen notaciones más elaboradas, como la presentada en Rinnooy Kan (1976) y más tarde la notación basada en la tripleta $\alpha/\beta/\gamma$ utilizada en varias ocasiones (Graham et al., 1979, Błazewicz, Lenstra y Rinnooy Kan, 1983, Lawler, Lenstra y Rinnooy Kan, 1993, Błazewicz et al., 1996, Portmann, 1997 y Pinedo, 2002). A continuación vamos a hacer una síntesis de los aspectos más importantes de este tipo de notación, incluyendo las aportaciones que a la misma han hecho los distintos trabajos citados:

- El campo α define el entorno de producción y se divide a su vez en dos campos $\alpha_1\alpha_2$ de manera que:

- $\alpha_1 = \emptyset$: una sola máquina, a veces también se representa por $\alpha_1 = 1$.
- $= P$: máquinas paralelas idénticas. En este caso, al tener cada trabajo una única operación y ser todas las máquinas iguales tenemos que $p_{ij} = p_j$.
- $= Q$: máquinas paralelas proporcionales o uniformes. Se proporciona, para cada máquina i un factor de velocidad v_i . La existencia de máquinas paralelas de diferente velocidad, pero uniformes hace necesaria la siguiente transformación: $p_{ij} = p_j/v_i$.
- $= R$: máquinas no relacionadas.
- $= O$: taller abierto o “*Open Shop*”.
- $= F$: taller de flujo o “*Flow Shop*”.
- $= J$: taller de trabajo o “*Job Shop*”.
- $\alpha_2 = 1, 2, \dots, m$: número de máquinas fijado al número dado.
- $= \emptyset$: el número de máquinas depende del ejemplo a resolver. A veces también se puede encontrar como m .

- El campo β indica las características atribuibles a los trabajos. Estas características son muy numerosas y normalmente los autores dividen este campo en varios subcampos. Hemos recogido un total de 12 posibles casos a partir de las diversas notaciones propuestas ($\beta_1\beta_2, \dots, \beta_{12}$):

- $\beta_1 = pmtn$: interrupción de operaciones o “*preemption*”. Indica que una operación O_{ij} , una vez comenzada en una máquina no es necesario que se complete totalmente y puede ser interrumpida. De producirse la interrupción, puede introducirse otra tarea diferente en la máquina. El tiempo ya procesado para la operación no se pierde, el tiempo restante puede procesarse en la misma o en otra máquina. Según el trabajo consultado a veces este campo se denota como *prmp*.
- $= \emptyset$: no se permite la interrupción de operaciones.
- $\beta_2 = prec$: relaciones de precedencia. Con este campo se especifica que existen relaciones de precedencia o “*precedence constraints*” entre los trabajos. Si un trabajo tiene un predecesor, no podrá comenzar hasta que éste haya terminado. Existen distintos tipos de relaciones de precedencia que se expresan haciendo $\beta_2 = chains$, $\beta_2 = intree$, $\beta_2 = outtree$ ó $\beta_2 = tree$.
- $= \emptyset$: no existen relaciones de precedencia.
- $\beta_3 = r_j$: existen fechas de entrada.
- $= \emptyset$: no existen fechas de entrada, o lo que es lo mismo $r_j = 0, \forall j \in N$.

- $\beta_4 = S_{nsd}$: existen tiempos de cambio de partida o “*set-up times*” para los trabajos en las máquinas que no dependen de la secuencia. Esto es, es necesario hacer ajustes en la máquina antes de procesar el trabajo y estos ajustes no dependen del trabajo procesado anteriormente en la misma máquina. Los tiempos de cambio se denotan con S_{ij} . Normalmente los ajustes son “anticipativos”, es decir, los ajustes en la máquina se pueden hacer antes de que el trabajo llegue a la máquina.
- $= S_{sd}$: existen tiempos de cambio de partida pero dependen de la secuencia (“*sequence-dependent set-up times*”). Para cada máquina i se define una matriz S_{ijk} con número de filas y número de columnas igual a n . De esta manera se representa un ajuste que hay que hacer en la máquina antes de procesar un trabajo y que este ajuste depende del trabajo que se había procesado con anterioridad en dicha máquina.
- $= \emptyset$: los tiempos de cambio de partida no existen o son muy pequeños y se incluyen en los tiempos de proceso.
- $\beta_5 = d_j$: existen fechas de finalización o “*due dates*”, las fechas son independientes para cada trabajo.
- $= d_j = d$: las fechas de finalización son homogéneas para todos los trabajos (“*common due dates*”).
- $= \bar{d}_j$: las fechas de finalización son de obligado cumplimiento (“*deadlines*”), en algunas notaciones esto se representa como Δ_j .
- $= \emptyset$: no existen fechas de entrega. Es interesante comentar que algunos autores no contemplan este campo de manera explícita ya que el criterio de optimización (el campo γ , que veremos más adelante) ya indica la existencia de fechas de entrega.

- $\beta_6 = prmu$: sólo aplicable cuando $\alpha_1 = F$. Indica que el orden de entrada de los trabajos es el mismo para todas las máquinas (la secuencia es idéntica para todas las máquinas del taller). En este caso existen $n!$ posibles secuencias.
- $= \emptyset$: indica que la secuencia de entrada en cada máquina puede variar. Es lo mismo que permitir “*job passing*”. Un trabajo j se puede procesar en tercer lugar en la máquina 3 y un trabajo k en cuarto lugar en la misma máquina. Si en la máquina 4 el trabajo k se procesa en tercer lugar se dice que el trabajo k “pasa” o “adelanta” al trabajo j en la máquina 4. En esta situación tenemos $(n!)^m$ posibles secuencias.
- $\beta_7 = brkdw$: las máquinas están sujetas a averías o a períodos donde no pueden procesar trabajos (“*breakdowns*”).
- $= \emptyset$: las máquinas se suponen disponibles en todo momento.
- $\beta_8 = nwt$: sin esperas o “*no-wait*”. Esta condición impone la restricción de que un trabajo no puede esperar entre dos máquinas, es decir, $O_{ij}f = O_{i+1,j}s, \forall j \in N, i = (1, \dots, m-1)$. Un ejemplo claro de este tipo de restricciones se da en la producción de acero; una barra de acero en proceso de producción no puede esperar entre máquinas dado que se enfriaría. Esto a veces requiere retrasar el inicio de las operaciones de un trabajo para asegurarse de que no se producen esperas y que el trabajo se puede procesar “de una tirada”. En algunos trabajos, a esta restricción se la conoce como “*zero-buffer*”. Aunque realmente el caso de “*no-wait*” no coincide con el “*zero-buffer*” en el caso en que $m \geq 2$ y $\alpha_1 = F$ (ver Norman, 1999). Además, cuando $\alpha_1 = F$ la existencia de esta restricción implica obviamente $\beta_6 = prmu$.
- $= \emptyset$: los trabajos pueden esperar por tiempo indefinido entre las máquinas.

- $\beta_9 = \text{block}$: bloqueo o “*blocking*”. Indica que existe una capacidad finita entre las máquinas para almacenar el producto en curso, a esta capacidad se la conoce como “*buffer*”. Si el buffer entre las máquinas i e $i + 1$ se llena, el trabajo que esté siendo procesado en la máquina i al terminarse “bloqueará” la máquina i , quedándose en la misma y no permitiendo que otros trabajos puedan procesarse en i . En este caso se dice que tanto el trabajo como la máquina están bloqueados. El caso extremo se da cuando no hay buffer entre las máquinas (“*zero buffers*”).
- $= \emptyset$: existe un buffer ilimitado entre las máquinas y por tanto ningún trabajo o máquina puede quedar bloqueado.
- $\beta_{10} = \text{recrc}$: recirculación o “*recirculation*”. Puede darse cuando $\alpha_1 = O, F, J$. Indica que un trabajo (o todos en general) pueden necesitar ser procesados más de una vez en alguna o todas las máquinas, o lo que es lo mismo $\exists j \in N$ tal que $|O_{ij}| > m$.
- $= \emptyset$: los trabajos sólo se procesan una vez en cada máquina.
- $\beta_{11} = M_j$: restricción de uso de máquinas o “*machine eligibility constraints*”. Sólo puede darse cuando $\alpha_1 = P$ o en problemas mixtos más complejos. En este caso, para cada trabajo $j \in N$ tenemos un conjunto M_j , tal que $|M_j| \leq m$ que indica el subconjunto de máquinas en las que j puede procesarse.
- $= \emptyset$: los trabajos pueden procesarse en cualquiera de las máquinas disponibles.
- $\beta_{12} = p_{ij} = 1$: todos los tiempos de proceso de los trabajos son iguales a 1.
- $= p_{ij} = p$: todos los tiempos de proceso de los trabajos son iguales a p .
- $= \emptyset$: los tiempos de proceso son cantidades no negativas arbitrarias.
- Por último, el campo γ proporciona información sobre el criterio de optimización utilizado. Antes de pasar a comentar este campo, es necesario introducir una serie de conceptos y parámetros adicionales que se basan en

la notación anterior y en los parámetros de los trabajos y que nos permitirán definir las medidas de optimización de una manera más sucinta (French, 1982):

- a_j : período permitido para el proceso del trabajo j (“allowance”). Es el lapso de tiempo entre la fecha de entrada y la de finalización o “deadline”; $a_j = d_j - r_j$ ó $a_j = \bar{d}_j - r_j$. En el caso de no existir fechas de entrada ($r_j = 0, \forall j \in N$) entonces $a_j = d_j$ o $a_j = \bar{d}_j, \forall j \in N$. Si no existen ni fechas de entrega, ni de finalización o “deadline” entonces $a_j = \infty, \forall j \in N$.
- W_{ij} : es el tiempo que el trabajo j espera antes de empezar la operación correspondiente en la máquina i . A partir de la definición de las operaciones tenemos que $W_{ij} = O_{ijs} - O_{i-1,j}f$. Claramente, en un entorno “no-wait” tenemos que $W_{ij} = 0, \forall j \in N, i = (2, 3, \dots, m)$.
- W_j : tiempo total de espera del trabajo j . Se calcula a partir de $W_j = \sum_{i=1}^m W_{ij}$.
- C_j : tiempo o fecha en que finaliza el proceso del trabajo j en el taller (“completion time”). A partir de los parámetros anteriores se puede calcular como $C_j = r_j + \sum_{i=1}^m W_{ij} + p_{ij}$ (siempre que no existan tiempos de cambio de partida). De igual manera $W_j = C_j - r_j - \sum_{i=1}^m p_{ij}$.
- C_{max} : máxima fecha de finalización. $C_{max} = \max\{C_1, C_2, \dots, C_n\}$. Equivale a $O_{mj}f$ cuando j es el último trabajo que se procesa en el taller en algunas configuraciones productivas, por ejemplo cuando $\alpha_1 = F$.
- I_i : tiempo ocioso o “idle time” de la máquina i . Se refiere al tiempo que la máquina i pasa sin procesar trabajos, se puede calcular como $I_i = C_{max} - \sum_{j=1}^n p_{ij}$. Esta relación ilustra que un C_{max} bajo implica una alta utilización de las máquinas y de los recursos del taller (Pinedo, 2002).
- F_j : tiempo de flujo o “flow time”. Es la cantidad de tiempo que un trabajo j permanece en el taller. Claramente $F_j = C_j - r_j$.

- L_j : holgura del trabajo frente a su fecha de finalización o “*deadline*”. $L_j = C_j - d_j$ ó $L_j = C_j - \bar{d}_j$. En la literatura anglosajona a este término se le da el nombre de “*lateness*”, que puede inducir a confusión. Si $L_j = 0$ se dice que el trabajo termina a tiempo “*on-time*”, en cambio si $L_j < 0$ diremos que el trabajo ha terminado antes de lo previsto “*early*” y si $L_j > 0$ entonces el trabajo termina después de la fecha máxima y se dice que se ha retrasado (“*tardy*”).
- T_j : retraso o “*tardiness*” de un trabajo j . Se calcula como $T_j = \max\{L_j, 0\}$.
- E_j : adelanto o “*earliness*” de un trabajo j . Se calcula como $E_j = \max\{-L_j, 0\}$.

A partir de los parámetros y conceptos anteriormente especificados, las secuencias se evalúan a partir de la información agregada que proporcionan los n trabajos a procesar. Normalmente se consideran medidas de eficacia unidimensionales y regulares. Una medida de eficacia Z es regular cuando es una función no decreciente de (C_1, C_2, \dots, C_n) , o lo que es lo mismo, si buscamos minimizar Z , el valor de Z sólo aumentará cuando lo haga algún $C_j, j \in N$.

De esta manera el campo γ nos indica la medida de eficacia utilizada o criterio de optimización. Todas las medidas que se detallan son regulares salvo que se indique lo contrario:

- $\gamma = C_{max}$: minimización de la fecha máxima de finalización o “*makeSpan*”. Como veremos en próximos capítulos, este criterio de optimización es el más utilizado en la literatura.
- $= \bar{C}$: minimización del tiempo medio de finalización. Se calcula como $\bar{C} = \frac{1}{n} \cdot \sum_{j=1}^n C_j$.
- $= F_{max}$: minimización del máximo tiempo de flujo, $F_{max} = \max\{F_1, F_2, \dots, F_n\}$. Es importante notar que si no existen fechas entrada, esto es, si $r_j = 0, \forall j \in N$, tenemos que $C_{max} = F_{max}$.
- $= \bar{F}$: minimización del tiempo medio de flujo, $\bar{F} = \frac{1}{n} \cdot \sum_{j=1}^n F_j$. Es interesante remarcar que minimizar \bar{C} es equivalente a minimizar \bar{F} en algunos tipos de problemas (French, 1982).

- = L_{max} : minimización de la máxima holgura, $L_{max} = \max\{L_1, L_2, \dots, L_n\}$. Con esto se busca minimizar el máximo incumplimiento de la fecha de finalización o “*deadline*”.
- = \bar{L} : minimización de la holgura media. $\bar{L} = \frac{1}{n} \cdot \sum_{j=1}^n L_j$.
- = T_{max} : minimización del retraso máximo. $T_{max} = \max\{T_1, T_2, \dots, T_n\}$. Con esto se busca minimizar el máximo retraso con respecto a la fecha de finalización o “*deadline*”.
- = \bar{T} : minimización del retraso medio. $\bar{T} = \frac{1}{n} \cdot \sum_{j=1}^n T_j$.
- = E_{max} : minimización del adelanto máximo. $E_{max} = \max\{E_1, E_2, \dots, E_n\}$.
- = \bar{E} : minimización del adelanto medio. $\bar{E} = \frac{1}{n} \cdot \sum_{j=1}^n E_j$.
- = N_T : minimización del número de trabajos retrasados. Se calcula a partir de la siguiente expresión: $N_T = \sum_{j=1}^n U_j$, donde $U_j = \begin{cases} 1 & \text{si } C_j > d_j \\ 0 & \text{en caso contrario} \end{cases}$

Después de definir este grupo de medidas de eficacia regulares sería posible añadir otras más generales añadiendo ponderaciones:

- $\gamma = \sum_{j=1}^n w_j C_j$: minimización de la suma ponderada de fechas máximas de finalización.
- = $\{\sum_{j=1}^n w_j F_j, \sum_{j=1}^n w_j T_j, \sum_{j=1}^n w_j L_j,$
 $\sum_{j=1}^n w_j E_j, \sum_{j=1}^n w_j U_j, \frac{1}{n} \cdot \sum_{j=1}^n w_j F_j,$
 $\frac{1}{n} \cdot \sum_{j=1}^n w_j T_j, \frac{1}{n} \cdot \sum_{j=1}^n w_j L_j, \frac{1}{n} \cdot \sum_{j=1}^n w_j E_j,$
 $\frac{1}{n} \cdot \sum_{j=1}^n w_j U_j\}$.

En cuanto a medidas de eficacia no regulares tenemos:

- $\gamma = \sum_{j=1}^n E_j + T_j$: minimización de la suma de adelantos y retrasos.
- = $\sum_{j=1}^n w_j E_j + w_j T_j$: minimización de la suma ponderada de adelantos y retrasos. En este caso se podría definir otro grupo de pesos para poder penalizar más los retrasos o los adelantos:
 $\sum_{j=1}^n w_j E_j + w'_j T_j$.

Existen medidas adicionales de eficacia, por ejemplo medidas relacionadas con las máquinas (minimizar $\sum_{i=1}^m I_i$) pero no son muy utilizadas en la práctica.

Una vez detallada la notación es posible dar unos sencillos ejemplos: un taller con varias máquinas uniformes dispuestas en paralelo con fechas de finalización y con el objetivo de minimizar el número de trabajos retrasados se denotaría por $P/d_j/N_T$. Otro ejemplo sería un taller de trabajo con recirculación, tiempos de cambio de partida independientes de la secuencia con el objetivo de minimizar el “makespan” y vendría expresado como $J/recrc, S_{nsd}/C_{max}$. Como vemos las combinaciones de tipos de problemas son muy numerosas, y aun así en la vida real podremos encontrar fácilmente configuraciones productivas que no son posibles de expresar con la notación anterior y para las cuales sería necesario dar condiciones adicionales, en forma de campos β autoexplicativos. Es importante también fijarse en que no todas las combinaciones anteriores son posibles. Un problema del tipo $P/nwt/C_{max}$ no tiene sentido, ya que con el nwt decimos que un trabajo no puede esperar entre dos máquinas, cuando en un entorno P cada trabajo sólo se procesa una vez en una máquina.

2.2. Complejidad computacional

En la sección anterior se ha proporcionado una clasificación para los distintos problemas de máquinas y como hemos visto, podemos encontrarnos con una gran variedad. Evidentemente, el objetivo que nos planteamos ante uno de estos problemas es resolverlo de manera eficaz, es decir, obtener una secuenciación de tal manera que no exista ninguna otra secuencia posible con un mejor valor de la función objetivo considerada. Para dar una visión más formal de qué se considera como solución óptima podemos considerar el problema de taller de flujo de permutación ($F/prmu/C_{max}$) para el cual, como ya se ha comentado, existen un total de $n!$ posibles secuencias que vienen dadas por todas las posibles permutaciones en n . Si llamamos Π a este conjunto de posibles permutaciones, π a un elemento de este conjunto y denotamos por $C_{max}(\pi)$ al valor del “makespan” para la secuencia π , la solución óptima o π^* será aquella que cumpla la siguiente condición:

$$C_{max}(\pi^*) \leq C_{max}(\pi), \forall \pi \in \Pi$$

Si bien el concepto de solución óptima es sencillo, no lo es tanto diseñar un método que obtenga esta solución en un tiempo razonable. Ya hemos visto en la Sección 1.1 como encontrar π^* para el problema del taller de flujo ($F//C_{max}$) mediante una evaluación de todas las posibles secuencias puede resultar impráctico ¿Quiere decir esto que no existe un método que encuentre π^* de manera eficiente? Para responder a esta pregunta existe una rama de la informática teórica llamada Teoría de la Complejidad.

De acuerdo a esta teoría, los problemas se asignan a clases de complejidad. El que un problema esté en una clase o en otra determinará la facilidad para diseñar un algoritmo que encuentre π^* de manera eficiente o que tal algoritmo sea muy poco probable que exista, por lo que invertir tiempo intentando encontrarlo puede ser totalmente inútil. Vamos a definir unos conceptos iniciales:

- problema (P_r): en nuestro caso cualquier problema de programación de la producción, por ejemplo un problema puede ser $F/prmu/C_{max}$.
- instancia (I): cualquier entrada de datos para el problema dado. Siguiendo el ejemplo una instancia concreta podría ser $m = 3$ (número de máquinas), $n = 4$ (número de trabajos) y los p_{ij} (tiempos de proceso) de acuerdo a la Tabla 2.1:

máquinas (i)	trabajos (j)			
	1	2	3	4
1	3	4	9	2
2	5	2	1	3
3	6	7	4	6

Tabla 2.1 – Tiempos de proceso para un ejemplo de problema de taller de flujo con 4 trabajos y 3 máquinas.

- tamaño (η): es una transformación de los datos de entrada (I) a una cantidad que representa el tamaño de la instancia o la longitud de la entrada de datos. En el ejemplo podríamos representar η como la cadena formada

por las tripletas (i, j, p_{ij}) o lo que es lo mismo, las distintas operaciones acompañadas de sus respectivos tiempos de proceso:

- $(1,1,3), (1,2,4), (1,3,9), (1,4,2), (2,1,5), (2,2,2), (2,3,1), (2,4,3), (3,1,6), (3,2,7), (3,3,4), (3,4,6)$. Con lo que $\eta = 36$, sin contar los separadores.

Otra opción es representar los datos mediante un alfabeto binario de ceros y unos:

- $(01,01,11), (01,10,100), (01,11,1001), (01,100,10), (10,01,101), (10,10,10), (10,11,01), (01,100,11), (11,01,110), (11,01,111), (11,11,100), (11,100,110)$. En este caso tendríamos $\eta = 83$.

La tercera opción es utilizar un alfabeto unario, donde cada dato de entrada se representa con un número de unos, por ejemplo $5 = 11111$, luego:

- $(1,1,111), (1,11,1111), \dots, (111,1111,111111)$. Con un η de 106.

Evidentemente, el alfabeto unario es el que necesita un mayor valor de η y por tanto el más ineficiente.

Con estas definiciones podemos hacer referencia a lo que entendemos como *complejidad temporal*¹. Definimos una *función de complejidad temporal* o $f(\eta)$ de un método o algoritmo al número de pasos que se requieren para resolver una instancia I de tamaño η de un problema cualquiera P_r . Por ejemplo, si tenemos un sencillo problema P'_r para el que, en función del tamaño de la instancia, existe un método que obtiene la solución óptima en $f(1200 \cdot \eta + 10\eta^2)$ pasos diremos que el método tiene una *complejidad polinomial*, porque f en este caso es un polinomio cualquiera. Realmente no es necesario expresar la complejidad polinomial mediante un polinomio. Normalmente diremos que P'_r es del orden $\mathcal{O}(\eta^2)$. Esto puede parecer un error, puesto que los polinomios $1200 \cdot \eta + 10\eta^2$ y

¹Hace años se hacía una distinción entre complejidad espacial y temporal, el espacio de procesamiento que necesitaba un método era muy importante al tener los ordenadores una memoria finita. Actualmente el interés se centra en la complejidad temporal al ser la memoria un recurso barato. Puede pensarse que ambas complejidades están muy relacionadas, sin embargo, podemos encontrar métodos que utilizan estructuras de datos compactas y más tiempo de proceso, así como métodos que hacen uso de estructuras muy extensas para ahorrar procesamiento.

η^2 no dan el mismo número de pasos, especialmente para valores pequeños de η . No obstante, cuando $\eta \rightarrow \infty$, el término que domina en ambos polinomios es el de mayor grado. De esta manera diremos que dos polinomios $f(\eta)$ y $g(\eta)$ son del mismo orden y se expresará como “ $f(\eta)$ es $\mathcal{O}(g(\eta))$ ” cuando:

$$\frac{f(\eta)}{g(\eta)} \rightarrow c, \text{ cuando } \eta \rightarrow \infty$$

Para el ejemplo, si hacemos $\eta = 500$ obtenemos un c de 3,4. Si $\eta = 5000$ la anterior expresión da un valor de c de 1,24, si hacemos $\eta = 50000$ tendremos c de 1,024.

Así, de manera general diremos que un método o algoritmo que resuelve un problema P_r tiene *complejidad polinomial* cuando su función de complejidad temporal asociada $f(\eta)$ es $\mathcal{O}(p(\eta))$ para cualquier polinomio $p(\eta)$. De otra manera diremos que el método tiene una *complejidad exponencial*. Ejemplos de métodos exponenciales pueden ser aquellos con una función de complejidad $\mathcal{O}(\eta!)$ o $\mathcal{O}(2^\eta)$. Es obvio que el número de pasos en función de η crece mucho más rápidamente en una función de complejidad exponencial que en una función de complejidad polinomial.

Al principio de la sección hablamos de clases de problemas, con las definiciones anteriores es posible hacer una primera clasificación:

- La clase \mathcal{P} está formada por todos aquellos problemas para los que existe un método o algoritmo de complejidad polinomial. Esta definición es informal, una definición más completa sería la clase formada por todos aquellos problemas para los que existe una máquina de Turing determinista que acepta el lenguaje generado por el problema en un tiempo polinomial dependiendo de η . Para comprender esta definición más rigurosa sería necesario introducir los conceptos de *alfabeto, lenguaje y máquina de Turing*, algo que se sale del ámbito de esta Tesis. En Garey y Johnson (1979) o en Kelley (1995) podemos encontrar estas definiciones.
- La clase \mathcal{NP} está formada por todos aquellos problemas para los que sólo se han encontrado métodos o algoritmos de complejidad exponencial. Una

definición intuitiva es la clase formada por aquellos problemas para los que no se ha encontrado un algoritmo de complejidad polinómica pero para los cuales existe un método para *verificar* una solución obtenida en tiempo polinomial. Esta última definición requiere hacer la distinción entre problemas de optimización y problemas de decisión o verificación. Con cada problema de optimización tenemos un problema de decisión asociado. Un problema de decisión da una respuesta del tipo “sí-no”, por ejemplo, para el problema $F//C_{max}$ el problema de optimización sería encontrar el mínimo “*makespan*”, mientras que un problema de decisión sería verificar si una secuencia concreta para el problema es menor que una constante c dada. Dado que no se conoce ningún algoritmo polinomial para encontrar la solución óptima para el problema $F//C_{max}$ pero una solución dada sí puede verificarse en tiempo polinomial (el valor del C_{max} se puede calcular en $n^2 \cdot m$ pasos) podemos decir que el problema $F//C_{max}$ pertenece a la clase \mathcal{NP} .

Como hemos visto, que un problema pertenezca a la clase \mathcal{NP} tan sólo quiere decir que no se conoce un algoritmo que lo resuelva en tiempo polinomial. Ahora, un resultado fundamental es que $\mathcal{P} \subseteq \mathcal{NP}$. De todas formas la incapacidad de la comunidad científica para encontrar algoritmos de complejidad polinómica para los problemas en \mathcal{NP} no implica que $\mathcal{P} \neq \mathcal{NP}$, este último extremo no ha podido ser demostrado aunque se cree que es cierto.

No todos los problemas en \mathcal{NP} son igualmente difíciles. y existen otras clases de complejidad. Para poder comentarlas vamos a introducir el concepto de *reducción*. Decimos que un problema P_r se reduce a otro problema P'_r si para cada instancia I de P_r podemos construir una instancia equivalente para P'_r . Más concretamente, se dice que un problema P_r se *reduce polinomialmente* a otro problema P'_r (se expresa como $P_r \leq P'_r$) si toda instancia I de P_r se puede transformar en una instancia equivalente de P'_r mediante un algoritmo de complejidad polinomial. O lo que es lo mismo, para una instancia I de tamaño η de P_r se puede construir una instancia de P'_r en un tiempo $f(\eta)$ donde f es un polinomio. Desde esta perspectiva, y de una manera intuitiva tenemos que el problema P'_r es más “sencillo” que el problema P_r . De esta manera y suponiendo que $P_r \leq P'_r$, si

tenemos que $P'_r \in \mathcal{P}$ entonces también tendremos que $P_r \in \mathcal{P}$, de igual manera si $P_r \propto P'_r$ y $P_r \notin \mathcal{P}$ tendremos que $P'_r \notin \mathcal{P}$.

Con las definiciones anteriores podemos introducir la definición de \mathcal{NP} -Difícil²:

- Decimos que un problema P_r es \mathcal{NP} -Difícil si todos los problemas en \mathcal{NP} son reducibles polinómicamente a P_r , esto es:

$$P'_r \propto P_r, \forall P'_r \in \mathcal{NP}$$

De igual manera, podemos definir la clase de problemas \mathcal{NP} -Completos o clase \mathcal{NPC} :

- Un problema P_r es \mathcal{NP} -Completo cuando es \mathcal{NP} -Difícil y además $P_r \in \mathcal{NP}$.

Un resultado muy interesante es que si encontráramos un algoritmo o método con complejidad polinomial para un problema en \mathcal{NPC} todos los problemas en \mathcal{NP} tendrían respuesta también en tiempo polinomial, o lo que es lo mismo $\mathcal{P} = \mathcal{NP}$. Como hemos visto, \mathcal{NP} -Completo $\neq \mathcal{NP}$ -Difícil. En otros casos se da la siguiente definición de \mathcal{NP} -Difícil:

- Cuando el problema de decisión asociado a un problema de optimización es \mathcal{NP} -Completo, entonces el problema de optimización es \mathcal{NP} -Difícil.

Lo cual claramente no ayuda para distinguir entre un tipo y otro de problemas. Lo que sí está claro (Garey y Johnson, 1979 y French, 1982) es que los problemas \mathcal{NP} -Complejos son los más difíciles de resolver de \mathcal{NP} . y que todos los problemas \mathcal{NP} -Complejos son también \mathcal{NP} -Difíciles.

Es posible afinar más haciendo una distinción (de manera muy intuitiva) entre los problemas \mathcal{NP} -Complejos en sentido estricto o \mathcal{NP} -Complejos unarios y los problemas \mathcal{NP} -Complejos ordinarios o \mathcal{NP} -Complejos binarios (también llamados \mathcal{NP} -Complejos débiles). Para un problema P_r y un polinomio $g(\eta)$ en función del tamaño de la instancia η , definimos un problema P_{rg} con la restricción

²Los problemas “ \mathcal{NP} -Hard” se han traducido erróneamente al castellano en algunos trabajos como \mathcal{NP} -Duros cuando claramente, en este contexto “hard” se refiere a difícil, no a duro.

de que todos los datos de entrada están acotados por $g(\eta)$. Diremos entonces que P_r es \mathcal{NP} -Completo en sentido estricto si para algún polinomio $g(\eta)$, P_{r_g} es \mathcal{NP} -Completo. En caso contrario P_r será \mathcal{NP} -Completo en sentido ordinario. De igual manera se podría definir el concepto de \mathcal{NP} -Difícil en sentido estricto si $P_r \notin \mathcal{NP}$. Los problemas \mathcal{NP} -Completos en sentido estricto son los más difíciles de resolver, y además es muy probable que no existan algoritmos de complejidad polinomial para poder resolverlos a no ser que $\mathcal{P} = \mathcal{NP}$.

La anterior clasificación no es rigurosa ni mucho menos exhaustiva. De hecho existen 327 diferentes clases de complejidad y este número no para de aumentar (<http://www.cs.berkeley.edu/~aaronson/zoo.html>) aunque en lo relacionado con la presente Tesis Doctoral no es necesaria una visión más completa. Más información sobre la teoría de la complejidad puede encontrarse en Garey y Johnson (1979) y más recientemente en Papadimitriou (1994).

Como hemos visto, la teoría de la complejidad es “compleja”, pero los resultados de complejidad de un problema son un claro indicativo de qué podemos hacer para resolverlos. Como se vio en la Sección 1.1, el problema del taller de flujo estándar ($F//C_{max}$) es \mathcal{NP} -Completo en sentido estricto (Garey, Johnson y Sethi, 1976), cuando $m \geq 3$. La presente Tesis Doctoral parte de este problema como base para obtener métodos eficaces para la programación flexible de la producción.

Existen pocos problemas de programación de la producción que sean fáciles de resolver, esto es, que pertenezcan a \mathcal{P} . Por ejemplo, el problema $F2//C_{max}$ se puede resolver en tiempo polinómico utilizando la regla de Johnson (1954). Otros problemas que pertenecen a \mathcal{P} son casos muy simplificados y casi siempre problemas de una única máquina o a lo sumo dos. Aun así es frecuente encontrar problemas de una máquina en la clase \mathcal{NP} -Difícil, como por ejemplo el problema $1//\sum T_i$ (ver Graham et al., 1979). Por lo general, problemas de m máquinas son en muchas ocasiones \mathcal{NP} -Difíciles o incluso \mathcal{NP} -Completos. En esta Tesis Doctoral trataremos los problemas del taller de flujo con y sin tiempos de cambio de partida dependientes de la secuencia y versiones más realistas de los mismos que contemplan la existencia de varias máquinas en cada etapa de producción. Como se verá en capítulos posteriores, todos estos problemas son \mathcal{NP} -Completos y la mayoría en sentido estricto. Por tanto, nos centraremos en algoritmos de tipo

heurístico, que nos asegurarán obtener buenas soluciones en un tiempo reducido, aunque no sean en muchos casos las óptimas.



CAPÍTULO

EL PROBLEMA DEL TALLER DE FLUJO

La configuración productiva más frecuente cuando existen varias máquinas y éstas no están dispuestas en paralelo es la de un taller de flujo. En el Capítulo 1 vimos un ejemplo real de taller de flujo, concretamente del proceso de producción en una industria azulejera, otro sector que presenta problemas de este tipo es la industria textil. A continuación vamos a dar una definición formal de este problema.

En un taller de flujo o “flowshop” tenemos un conjunto N de trabajos, $N = (1, 2, \dots, n)$, que hay que procesar en un conjunto M de máquinas, $M = (1, 2, \dots, m)$. En principio cada trabajo debe procesarse en todas y cada una de las máquinas, con la particularidad de que la secuencia de proceso es la misma para todos los trabajos. Es decir, existe un “flujo” común de los trabajos en las máquinas. De esta manera tendremos un total de $n \cdot m$ operaciones. Las máquinas del taller se pueden disponer en serie y numerarlas de tal manera que la primera máquina en la que se deben procesar los trabajos sea la máquina 1 y la última máquina sea la m .

El flujo de los trabajos en las máquinas implica una relación de precedencia en

las operaciones, es decir, para cualquier trabajo j , el orden de procesamiento de sus m operaciones es el siguiente: $O_{1j}, O_{2j}, \dots, O_{mj}$. O lo que es lo mismo, las tareas entre la 2 y la $m - 1$ tienen una única tarea precedente y una única tarea sucesora. La tarea O_{1j} tiene una única sucesora y la tarea O_{mj} sólo tiene una precedente. Centrándonos en los tiempos de inicio y finalización de cada tarea podemos formalizar esta relación de precedencia de la siguiente manera:

$$\forall j \in N, O_{ij}s \geq O_{i-1,j}f, i = (2, \dots, m) \quad (3.1)$$

La Figura 3.1 representa un ejemplo de taller de flujo de manera esquemática.

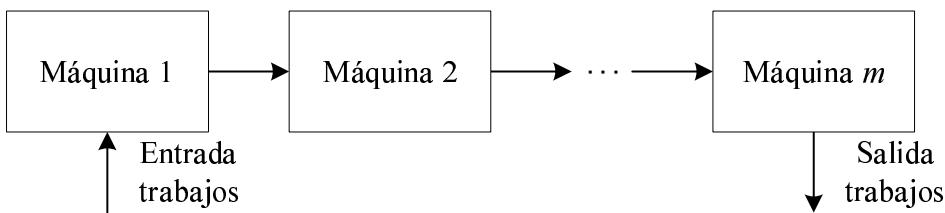


Figura 3.1 – Esquema del problema de taller de flujo o “*flowshop*”.

Existe una versión generalizada del problema de taller de flujo que se da cuando no es necesario que un trabajo se procese en las m máquinas. De esta manera, la primera operación no tiene por qué hacerse en la primera máquina y la última tampoco se hace obligatoriamente en la última máquina. Es decir, algunos trabajos se pueden “saltar” algunas máquinas. Por tanto la expresión 3.1 no se cumple para todo i y j , aunque para las operaciones de los trabajos sigue habiendo un orden. Este tipo de talleres de flujo se conocen como Taller de Flujo Generalizado o “*General Flow Shop*” (Baker, 1974). Un ejemplo esquemático de este tipo de talleres aparece en la Figura 3.2.

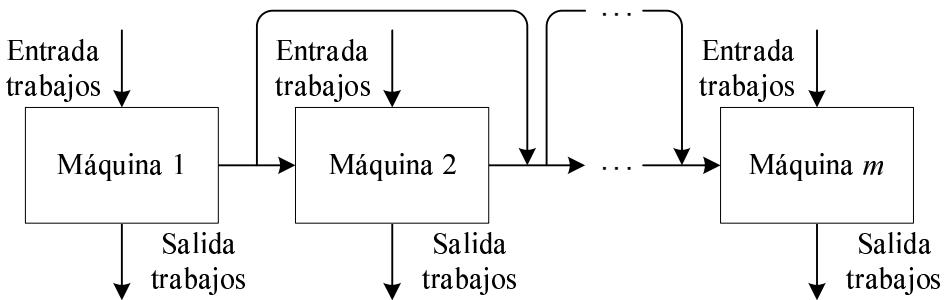


Figura 3.2 – Esquema de un problema de tipo taller de flujo generalizado o “general flow shop”.

Este último problema en algunos casos se puede modelizar haciendo $p_{ij} = 0$ y por tanto $O_{ij}s = O_{ij}f$ para aquellos trabajos j que no se procesan en alguna máquina i .

El criterio de optimización más utilizado en la literatura para este problema es la minimización de la fecha máxima de finalización o “makespan” (C_{max}). Este objetivo consigue una alta utilización de los recursos productivos, en nuestro caso máquinas, al tiempo que permite una entrega rápida del producto finalizado. A lo largo de la presente Tesis Doctoral éste será el objetivo considerado para la programación de la producción.

Como vimos en el Capítulo 2, el taller de flujo con la minimización de C_{max} como objetivo se denota por $n/m/F/F_{max}$ de acuerdo con la notación de Conway, Maxwell y Miller (1967) y como $F//C_{max}$ siguiendo la notación propuesta en la Sección 2.1 que está basada en la notación de Graham et al. (1979). Otros autores, como Pinedo (2002), utilizan una notación ligeramente distinta y denominan el problema como $Fm//C_{max}$.

Como se puede observar, el campo β de la notación está vacío, esto tiene muchas implicaciones, además de las propias de un taller de flujo y que pasamos a comentar:

- Existen n de trabajos a procesar en las m máquinas, siendo un taller de flujo “regular”. El entorno es estático, es decir, el conjunto N de trabajos

no varía.

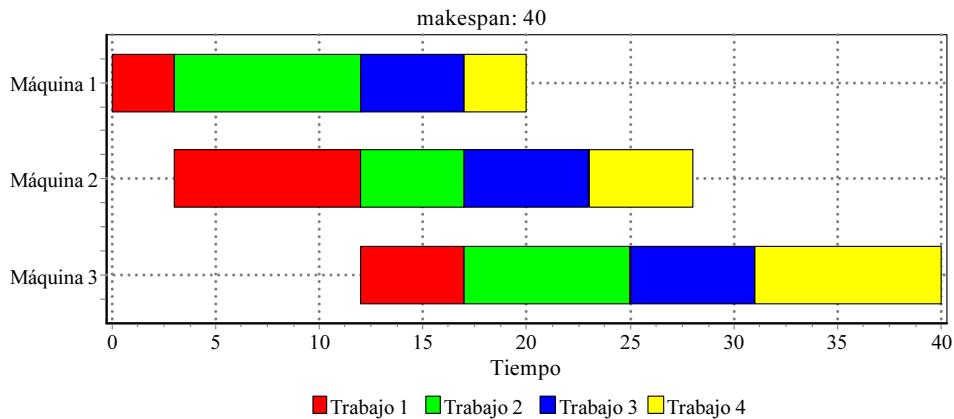
- Todos los trabajos están disponibles para procesarse en tiempo cero. Esto es, no hay fechas de entrada, $r_j = 0, \forall j \in N$. De ahí que $C_{max} = F_{max}$.
- Cada trabajo j solo puede procesarse al mismo tiempo en una máquina i .
- Cada máquina i solo puede procesar un trabajo j al mismo tiempo.
- No se permite la interrupción de las operaciones. Una vez comienza una operación de un trabajo j en una máquina i , ésta debe procesarse hasta su finalización por un tiempo p_{ij} .
- Los trabajos son independientes entre sí (que no las operaciones), es decir, no hay relaciones de precedencia de ningún tipo.
- Las máquinas están continuamente disponibles desde el instante cero.
- Los tiempos de cambio de partida o son muy pequeños y se pueden despreciar, o son independientes de la secuencia y no separables de los tiempos de proceso y por tanto van incluidos en los p_{ij} .
- Se asume una capacidad de almacenamiento ilimitada entre las máquinas. Nunca un trabajo queda bloqueado entre dos máquinas.
- Los tiempos de proceso p_{ij} de los trabajos en las máquinas se conocen de antemano, son deterministas y permanecen constantes durante todo el proceso.
- Se asume que los tiempos de transporte de los trabajos entre las máquinas son despreciables.

Como vemos, la cantidad de supuestos que se asumen hace que el problema tenga un relativo interés práctico. Desde el punto de vista industrial es común que no todos los materiales estén disponibles, que las máquinas tengan un estricto mantenimiento y puedan en algún momento no estar operativas o que aparezcan nuevos pedidos urgentes o cambios en la producción, de igual manera que los tiempos de producción no son fijos y deterministas. Aun así, la consideración de tan solo

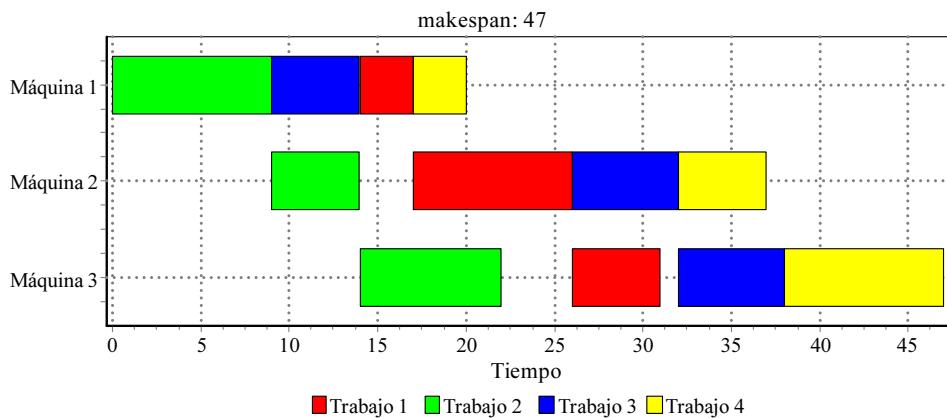
este problema de programación la producción simplificado y la obtención de una secuencia óptima de fabricación en un taller de flujo constituye un problema formidable. Como ya vimos en el Capítulo 1, el taller de flujo tiene un total de $(n!)^m$ posibles secuencias de fabricación por lo que se hace inviable la obtención de soluciones óptimas para problemas donde tengamos algo más de unos pocos trabajos y máquinas. También se ha comentado ya que este problema es \mathcal{NP} -Completo en sentido estricto (Garey, Johnson y Sethi, 1976). No obstante, algunas pequeñas simplificaciones son posibles a partir de las siguientes propiedades (ver Conway, Maxwell y Miller, 1967, Baker, 1974 y French, 1982):

- Para el taller de flujo con cualquier medida de eficacia regular como criterio de optimización, sólo es necesario considerar secuencias donde la entrada de los trabajos en las máquinas 1 y 2 es idéntica. Luego en general sólo es necesario considerar $(n!)^{m-1}$ secuencias.
- En el caso concreto de minimización del “makespan” (C_{max}), sólo es necesario considerar secuencias donde la entrada de los trabajos en las máquinas $m - 1$ y m es idéntica. En este caso podemos tener la misma secuencia en las máquinas 1 y 2 (la minimización del “makespan” también es una medida de eficacia regular) y la misma secuencia en las máquinas $m - 1$ y m , por lo que sólo es necesario considerar $(n!)^{m-2}$ posibles ordenaciones.

Definimos una secuencia de permutación de trabajos como aquella secuencia en la que todos los trabajos siguen el mismo orden de entrada en las máquinas, es decir, el orden de proceso de los trabajos en la máquina 1 es el mismo que la máquina 2, 3 y así hasta la m . La Figura 3.3 muestra dos diagramas de Gantt para un sencillo problema de 3 máquinas y 4 trabajos.



(a) Secuencia de permutación.



(b) Secuencia general.

Figura 3.3 – Diferencia entre secuencias generales y secuencias de permutación.

El diagrama de la parte superior se corresponde con una secuencia de permutación, mientras que en el diagrama inferior podemos observar como el trabajo 1 adelanta al trabajo 3 en la segunda máquina (esto se conoce como “*job-passing*”). Las máquinas 1 y 2 no tienen la misma secuencia (aunque la 2 y la 3 sí) y por

tanto no se puede hablar de una secuencia de permutación.

La anterior definición junto con las dos anteriores simplificaciones tienen implicaciones en cuanto a los problemas de tipo taller de flujo. Para el problema $F2//X$, siendo X cualquier medida de eficacia regular, las secuencias de permutación dominan a las secuencias generales, esto es, examinando sólo las secuencias de permutación tenemos la seguridad de poder encontrar la solución óptima. De esta manera, el problema $F2//X$ es equivalente al problema $F2/prmu/X$ y sólo hay que examinar $n!$ posibles soluciones.

En el caso de los problemas $F2//C_{max}$ y $F3//C_{max}$ la situación es similar, con examinar $n!$ secuencias es suficiente. Ahora bien, en el caso general con m máquinas, $m > 3$, las secuencias de permutación ya no dominan a las secuencias generales y se hace necesario evaluar $(n!)^{m-2}$ secuencias. Por ejemplo Potts, Shmoys y Williamson (1991) demostraron como la solución óptima de un problema $F/prmu/C_{max}$ es como mínimo un $1/2 \cdot \sqrt{m}$ peor (en %) que la solución de un $F//C_{max}$, aunque este resultado sólo se pudo constatar para un grupo concreto de ejemplos con unas características muy especiales. Que nosotros sepamos, no existen estudios posteriores que permitan medir el impacto de utilizar secuencias de permutación frente a secuencias generales.

Pese a esta situación, la literatura existente para el problema del taller de flujo se limita casi exclusivamente, como veremos, al taller de flujo de permutación o $F/prmu/C_{max}$ donde tan solo se consideran secuencias de permutación. Desgraciadamente, pese a esta enorme simplificación de pasar de $(n!)^{m-2}$ posibles secuencias a $n!$, la complejidad computacional del problema no cambia, o lo que es lo mismo, sigue siendo muy difícil de resolver en la práctica. Rinnooy Kan (1976) demostró que este problema también es \mathcal{NP} -Completo.

3.1. Métodos de programación de la producción para el taller de flujo

Inicialmente, y antes de exponer los métodos existentes para la programación de la producción en un taller de flujo, vamos a detallar cómo se calcula el “makespan” o C_{max} . Supongamos que tenemos una permutación de n trabajos

a la que denominamos π . Al trabajo que ocupa la posición k de la permutación (o secuencia) lo denotamos por $\pi_{(k)}$. De esta manera podemos calcular los instantes de comienzo y finalización de todas las operaciones de los trabajos de la siguiente manera:

$$\begin{aligned} O_{i,\pi_{(j)}} s &= \max \left\{ O_{i,\pi_{(j-1)}} f; O_{i-1,\pi_{(j)}} f \right\} \\ O_{i,\pi_{(j)}} f &= O_{i,\pi_{(j)}} s + p_{i,\pi_{(j)}}, \quad i = (1, \dots, m), j = (1, \dots, n) \end{aligned}$$

Adicionalmente, tenemos que $O_{0j} f = 0, j = (1, \dots, n)$ y que $O_{i0} f = 0, i = (1, \dots, m)$. Una vez calculados instantes de comienzo y finalización para cada operación, es posible calcular los tiempos de finalización de cada trabajo de la siguiente forma:

$$C_{\pi_{(j)}} = O_{m,\pi_{(j)}} f, \quad j = (1, \dots, n)$$

Y el “makespan” (C_{max}) se calcula de la siguiente manera:

$$C_{max} = \max \left\{ C_{\pi_{(1)}}, C_{\pi_{(2)}}, \dots, C_{\pi_{(n)}} \right\}$$

En el caso del taller de flujo, este último paso de cálculo del C_{max} se puede calcular de una manera más sencilla:

$$C_{max} = O_{m,\pi_{(n)}} f$$

Es necesario recalcar que las anteriores expresiones sólo son válidas en las condiciones en las que se ha definido el taller de flujo, teniendo además en cuenta de que se trata de un taller de flujo de permutación. Si por ejemplo tuviésemos fechas de entrada (r_j) las anteriores fórmulas devolverían un valor incorrecto para el C_{max} .

Vamos ahora a hacer una revisión de los métodos existentes para el problema del taller de flujo. Primero veremos un algoritmo clásico y las conocidas reglas de prioridad y después una breve revisión de los métodos exactos para luego ver más en detalle las distintas propuestas heurísticas y metaheurísticas.

3.1.1. Algoritmo clásico de Johnson

El primer método conocido para la programación del taller de flujo es la famosa regla de Johnson (1954). El algoritmo resuelve de manera óptima el problema $F2//C_{max}$. Algunos autores (Conway, Maxwell y Miller, 1967) atribuyen el uso generalizado del “makespan” como objetivo de optimización en la literatura al trabajo de Johnson. Otros autores, como por ejemplo Gupta y Dudek (1971), han criticado el uso del C_{max} como criterio de optimización y han propuesto criterios alternativos. De todas formas, el C_{max} ha sido y sigue siendo el criterio más estudiado. Este trabajo de Johnson ha sido reconocido como el origen de la investigación en “scheduling” y por su importancia, vamos a describirlo a continuación.

La regla de Johnson determina que un trabajo j precede en la secuencia a un trabajo k si:

$$\min\{p_{1j}, p_{2k}\} < \min\{p_{1k}, p_{2j}\}$$

De esta manera, los trabajos con un tiempo de proceso corto en la primera máquina van primero en la secuencia (así terminan antes) y los trabajos con un tiempo corto en la segunda máquina se procesan al final para así evitar tiempos muertos en la programación. Para hacer operativa la regla de Johnson se aplica un sencillo algoritmo:

1. Hacer $Ini = 1$, $Fin = n$. Crear una lista ℓ de trabajos ya secuenciados y una lista de trabajos no secuenciados ℓ' . Inicializar $\ell = \emptyset$ y $\ell' = \{1, \dots, n\}$
2. Para todos los trabajos de ℓ' , buscar el trabajo con el mínimo tiempo de proceso en la máquina 1 y el trabajo con el mínimo tiempo de proceso en la máquina 2. Hacer $min_1 = \min_{j \in \ell'}(p_{1j})$, $h_1 = \ell'(j)$ y $min_2 = \min_{k \in \ell'}\{p_{2k}\}$, $h_2 = \ell'(k)$
3. Si $min_1 < min_2$ entonces ir al paso 4. Si $min_1 > min_2$ ir al paso 5. Si $min_1 = min_2$ ir aleatoriamente o al paso 4 o al 5
4. Insertar el trabajo h_1 en la posición Ini de ℓ , $\ell(Ini) = h_1$. Hacer $Ini = Ini + 1$. Ir al paso 6

5. Insertar el trabajo h_2 en la posición Fin de ℓ , $\ell(Fin) = h_2$. Hacer $Fin = Fin - 1$. Ir al paso 7
6. Eliminar h_1 de ℓ' . Si $\ell' = \emptyset$ entonces fin, sino ir al paso 2
7. Eliminar h_2 de ℓ' . Si $\ell' = \emptyset$ entonces fin, sino ir al paso 2

Vamos a aplicar el algoritmo a un sencillo ejemplo. Resolveremos un ejemplo con seis trabajos cuyos tiempos de proceso (p_{ij}) se detallan en la Tabla 3.1.

máquinas (i)	trabajos (j)					
	1	2	3	4	5	6
1	8	4	10	5	3	12
2	12	1	6	5	2	8

Tabla 3.1 – Tiempos de proceso para un ejemplo de problema de taller de flujo con 6 trabajos y 2 máquinas.

Tras aplicar el primer paso tendremos $Ini = 1$, $Fin = 6$, $\ell = \emptyset$ y $\ell' = \{1, 2, 3, 4, 5, 6\}$. En el segundo paso tenemos que $\min_1 = 3$, $h_1 = 5$ y que $\min_2 = 1$, $h_2 = 2$, por tanto $\min_1 > \min_2$ y saltamos al paso 5. Hacemos $\ell(Fin) = 2$, $Fin = 5$ y vamos al punto 7. En este punto tenemos $\ell = \{\cdot, \cdot, \cdot, \cdot, 2\}$ y $\ell' = \{1, 3, 4, 5, 6\}$. Con esto habríamos terminado la primera iteración. En la segunda iteración $\min_1 = 3$, $h_1 = 5$ y que $\min_2 = 2$, $h_2 = 5$, por tanto $\min_1 > \min_2$ por lo que iríamos al paso 5. Hacemos $\ell(Fin) = 5$, $Fin = 4$ e iríamos al paso 6 donde tendríamos $\ell = \{\cdot, \cdot, \cdot, \cdot, 5, 2\}$ y $\ell' = \{1, 3, 4, 6\}$. Continuaríamos iterando hasta alcanzar la condición de parada. La Tabla 3.2 muestra detalladamente todas las iteraciones por las que pasa el algoritmo.

Etapa	ℓ'	min_1, h_1	min_2, h_2	ℓ
1	{1, 2, 3, 4, 5, 6}	3, 5	1, 2	{., ., ., ., ., 2}
2	{1, 3, 4, 5, 6}	3, 5	2, 5	{., ., ., ., 5, 2}
3	{1, 3, 4, 6}	5, 4	5, 4	{4, ., ., ., 5, 2} ¹
4	{1, 3, 6}	8, 1	6, 3	{4, ., ., 3, 5, 2}
5	{1, 6}	8, 1	8, 6	{4, ., 6, 3, 5, 2} ²
6	{1}	8, 1	12, 1	{4, 1, 6, 3, 5, 2}

¹En este paso hay un empate dado que $p_{14} = p_{24} = 5$. Hemos asignado el trabajo 4 según el paso 4, pero podríamos igualmente haberlo asignado según el paso 5.

²En este otro caso hay un empate, dado que $p_{11} = p_{26} = 8$. Aquí hemos asignado según el paso 5 el trabajo 6. Pero podríamos haber asignado según el paso 4 el trabajo 1.

Tabla 3.2 – Ejecución completa del algoritmo basado en la regla de Johnson para un problema con 6 trabajos.

En la Figura 3.4 podemos ver el diagrama de Gantt correspondiente a la secuenciación de la solución $\ell = \{4, 1, 6, 3, 5, 2\}$ obtenida tras aplicar el algoritmo para la regla de Johnson.

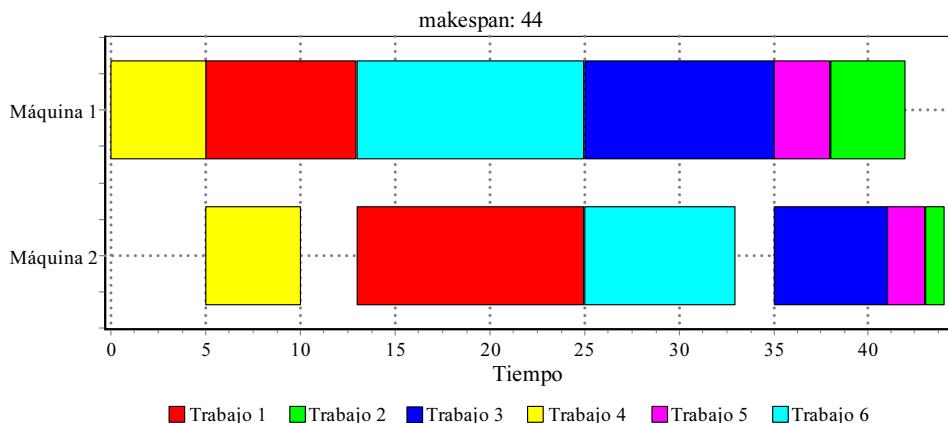


Figura 3.4 – Diagrama de Gantt para la secuencia resultado de la aplicación del algoritmo basado en la regla de Johnson.

Como hemos visto, la regla de Johnson sólo sirve para el caso concreto en el que $m = 2$. Para el caso general con m máquinas no es directamente aplicable. Algunos autores, como por ejemplo Campbell, Dudek y Smith (1970) aplican la regla de Johnson de forma heurística para m máquinas creando un problema con dos máquinas “virtuales”. Una manera simple de construir este problema virtual es definiendo como p'_{1j} y p'_{2j} a los tiempos en las dos máquinas del problema virtual que se calculan como:

$$\begin{aligned} p'_{1j} &= \sum_{i=1}^{\lceil \frac{m}{2} \rceil} p_{ij} \\ p'_{2j} &= \sum_{i=\lceil \frac{m}{2} \rceil + 1}^m p_{ij} \\ j &= (1, \dots, n) \end{aligned}$$

Resolviendo este nuevo problema con el algoritmo propuesto para la regla de Johnson podremos obtener una secuenciación de los trabajos para el problema original, aunque en este caso la solución obtenida puede ser que se desvíe considerablemente de la solución óptima, como veremos más adelante.

3.1.2. Reglas de prioridad

Existe un conjunto de métodos clásicos, muy frecuentemente utilizados en la práctica, que son las conocidas reglas de prioridad. No es fácil atribuir este tipo de métodos a ningún autor en concreto, pero es sencillo conocer su funcionamiento básico. En un algoritmo que aplica una regla de prioridad se mantiene una lista de operaciones que se pueden “lanzar” al taller, es decir, operaciones cuyas precedentes ya están en el taller, a esta lista la llamaremos lista de elegibles y la denotaremos por φ . De una forma sucinta, se puede decir que las reglas de prioridad secuencian el orden de lanzamiento de estas operaciones al taller ateniéndose a varios criterios. Estas reglas se pueden aplicar a diversos tipos de problemas de programación de la producción, no sólo al taller de flujo. Vamos a comentar algunas de las más conocidas:

- Regla “*First Come First Served*” (FCFS): se secuencia la operación más antigua de la lista de elegibles φ . Es equivalente a ordenar las tareas en orden “*First In First Out*” (FIFO).
- Regla “*Last Come First Served*” (LCFS): se secuencia la tarea más reciente de la lista de elegibles. De manera análoga a la regla FCFS, equivale a ordenar las tareas en orden “*Last In First Out*” (LIFO).
- Regla “*Shortest Processing Time*” (SPT): se secuencia la operación de la lista de elegibles que tiene el tiempo de proceso más corto. Es decir, se secuencia la tarea k tal que:

$$p_{ik} = \min_{j=1}^{|\varphi|} \{p_{ij}\}$$

- Regla “*Longest Processing Time*” (LPT): se secuencia la operación de la lista de elegibles que tiene el tiempo de proceso más largo, o lo que es lo mismo, aquella tarea k que cumpla:

$$p_{ik} = \max_{j=1}^{|\varphi|} \{p_{ij}\}$$

- Regla “*Earliest Due Date*” (EDD): entra en el taller la tarea de la lista de elegibles que pertenezca al trabajo que menor fecha de finalización o “*deadline*” tenga. Se secuencia la tarea k tal que:

$$d_j = \min_{j=1}^{|\varphi|} \{d_j\} \vee \bar{d}_j = \min_{j=1}^{|\varphi|} \{\bar{d}_j\}$$

En realidad existen muchas más reglas de prioridad que las que aquí se han comentado. Algunos autores han realizado estudios comparativos de este tipo de métodos, como por ejemplo Panwalkar y Iskander (1977) o Blackstone, Phillips y Hogg (1982). No obstante, la versatilidad de las reglas de prioridad contrasta con su bajo rendimiento en el taller de flujo en comparación con otros algoritmos heurísticos y metaheurísticos especialmente diseñados para este problema.

Adicionalmente, es posible hacer un refinamiento de las reglas de prioridad

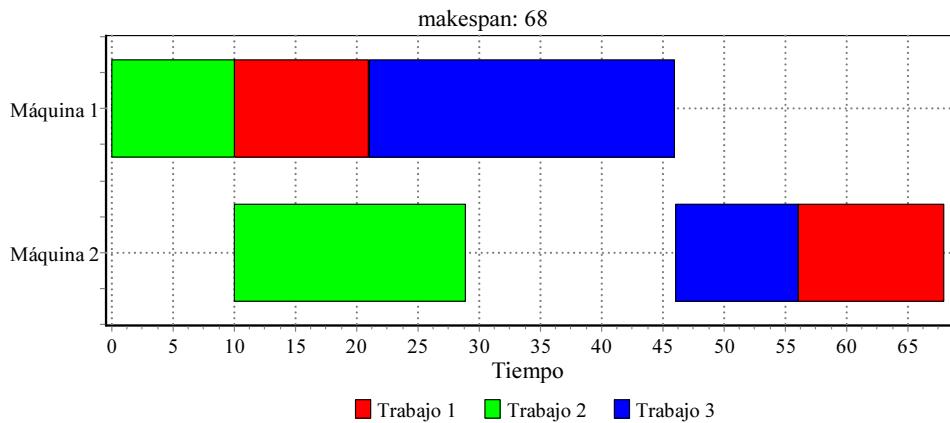
anteriormente expuestas y obtener una versión más “evolucionada” de las reglas de prioridad SPT y LPT que evita un problema común de este tipo de métodos. El problema viene motivado por la situación que se detalla a continuación; en las reglas de prioridad primero se busca la primera máquina que queda libre en el taller y luego se aplica la regla para ver qué tarea se secuencia para esa máquina. Bien, pues puede ocurrir que se secuencie una tarea cuya precedente todavía no ha terminado, y por tanto el instante de inicio para la tarea a secuenciar puede ser superior incluso que el tiempo de finalización para alguna tarea de la lista de elegibles que no ha sido elegida por la regla de prioridad. Para entender mejor esta situación vamos a plantear el siguiente ejemplo. Tenemos un sencillo problema con $n = 3$ y $m = 2$, y que aplicamos la regla SPT, los tiempos de proceso para los tres trabajos se muestran a continuación en la Tabla 3.3.

máquinas (i)	trabajos (j)		
	1	2	3
1	11	10	25
2	12	19	10

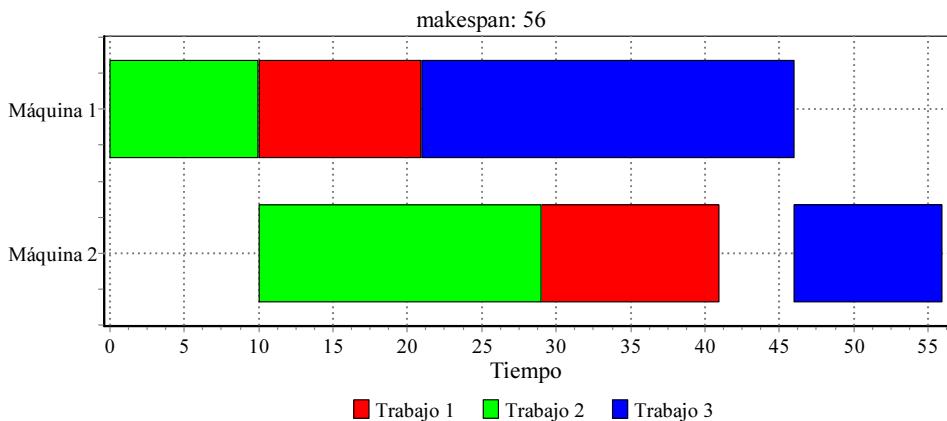
Tabla 3.3 – Tiempos de proceso para el ejemplo de las reglas de prioridad.

Inicialmente la lista de elegibles está formada por las tareas que no tienen precedentes, o sea, $\wp = \{O_{11}, O_{12}, O_{13}\}$. A partir de los p_{ij} vemos que la tarea O_{12} es la que se secuenciaría en primer lugar, de manera que $O_{12}s = 0$ y $O_{12}f = 10$ y actualizarnos la lista de elegibles con las sucesoras de la tarea O_{12} , es decir $\wp = \{O_{11}, O_{13}, O_{22}\}$. En la siguiente iteración tenemos que la máquina 2 es la que primero está libre (dado que la máquina 1 se libera en el instante 10) por lo que secuenciamos la tarea O_{22} al no haber más tareas en \wp para esta máquina. Tendremos que $O_{22}s = 10$ y $O_{22}f = 29$. O_{22} no tiene sucesoras, luego $\wp = \{O_{11}, O_{13}\}$. Aplicamos de nuevo la regla de prioridad escogiendo la máquina 1 que es la que primero queda libre y secuenciamos la tarea O_{11} haciendo $O_{11}s = 10$ y $O_{11}f = 21$, hacemos $\wp = \{O_{13}, O_{21}\}$. En una nueva iteración asignamos de nuevo a la máquina 1, dado que la máquina 2 se

libera más tarde. En este caso tenemos ya una única tarea para esta máquina en \wp , secuenciamos la tarea O_{13} haciendo $O_{13}s = 21$ y $O_{13}f = 46$ e incluimos sus sucesoras haciendo $\wp = \{O_{21}, O_{23}\}$. En este momento es cuando aplicamos la mejora a las reglas de prioridad, esta mejora está basada parcialmente en los resultados de Companys Pascual y Corominas Subias (1996). En una nueva iteración escogemos la máquina 2 por ser la que primero queda libre, aplicando directamente la regla de prioridad SPT la tarea con menor p_{ij} de \wp es O_{23} , y tendríamos que hacer $O_{23}s = 46$ y $O_{23}f = 56$, para luego secuenciar en la última iteración la única tarea restante, O_{21} , haciendo $O_{21}s = 56$ y $O_{21}f = 68$. Sin embargo, O_{21} puede secuenciarse antes que O_{23} porque realmente O_{21} puede terminar antes de que empiece O_{23} . Luego en cada iteración, y antes de aplicar la regla de prioridad buscaremos, para la máquina en la que vamos a secuenciar, aquella tarea que antes puede terminar en dicha máquina. Todas aquellas tareas secuenciables que puedan empezar como mínimo más tarde que esta tarea se marcarán como “dominadas”. Al aplicar la regla de prioridad impondremos la restricción de que no se contemplarán tareas dominadas. Podemos observar en la Figura 3.5 el resultado de aplicar ambas versiones de la regla SPT para el ejemplo de la Tabla 3.3.



(a) Aplicación de la regla SPT estándar.



(b) Aplicación de la regla SPT mejorada.

Figura 3.5 – Comparación entre la versión estándar y mejorada de las reglas de prioridad para el ejemplo: regla SPT.

Es inmediato observar que las reglas de prioridad mejoradas siempre obtendrán un menor valor de C_{max} .

3.1.3. Métodos exactos

A partir del trabajo pionero de Johnson varios autores aplicaron técnicas exactas para resolver el problema general con m máquinas. Uno de los primeros trabajos es el modelo de programación entera mixta de Wagner (1959), que apareció simultáneamente al modelo de Bowman (1959). Ya por aquel entonces, el propio autor indicó que el modelo era de escaso interés computacional y sólo válido para situaciones con muy pocos trabajos y máquinas. Vamos a presentar una modelización basada en este trabajo y en las modificaciones posteriores de Pinedo (2002).

Se definen tres tipos de variables:

$$X_{jk} = \begin{cases} 1 & \text{si el trabajo } j \text{ ocupa la posición } k\text{-ésima de la secuencia} \\ 0 & \text{en caso contrario} \end{cases}$$

En la anterior definición de X_{jk} tenemos que $j, k = (1, \dots, n)$, por lo que necesitaremos n^2 variables de tipo binario. Los otros dos grupos de variables son:

$$\begin{aligned} I_{ik} &= \text{Tiempo ocioso de la máquina } i \text{ entre los trabajos } k \text{ y } k+1 \\ i &= (1, \dots, m), k = (1, \dots, n-1) \end{aligned}$$

$$\begin{aligned} W_{ik} &= \text{Tiempo que espera el trabajo } k \text{ entre las máquinas } i \text{ e } i+1 \\ i &= (1, \dots, m-1), k = (1, \dots, n) \end{aligned}$$

Para poder explicar esta definición de variables vamos a servirnos de un ejemplo en el que tenemos 4 máquinas y 5 trabajos, los tiempos de proceso se detallan en la Tabla 3.4.

máquinas (i)	trabajos (j)				
	1	2	3	4	5
1	31	19	23	13	33
2	41	55	42	22	5
3	25	3	27	14	57
4	30	34	6	13	19

Tabla 3.4 – Tiempos de proceso para un ejemplo de problema de taller de flujo con 4 máquinas y 5 trabajos.

Realizando una sencilla secuenciación por orden de índice de los trabajos, es decir, haciendo una secuenciación $\pi = \{1, 2, 3, 4, 5\}$, obtenemos el diagrama de Gantt que aparece en la Figura 3.6. En la figura se muestran algunos ejemplos de las variables del modelo.

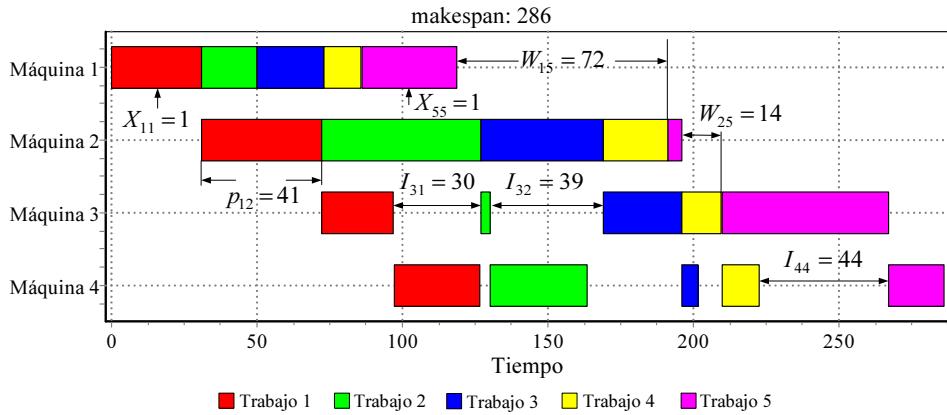


Figura 3.6 – Diagrama de Gantt para el ejemplo de 4 máquinas y 5 trabajos donde se detallan las variables del modelo de Wagner.

En general, nunca tendremos tiempos ociosos en la primera máquina, por lo que $I_{1k} = 0$, $k = (1, \dots, n - 1)$. De la misma manera, si las máquinas están siempre disponibles, el primer trabajo de la secuencia no tendrá que esperar en ninguna máquina, luego $W_{i1} = 0$, $i = (1, \dots, m - 1)$. De esta manera, al número de variables binarias anteriormente especificado, tendremos que añadirle un total de $2mn - 2m - 2n + 2$ variables continuas.

Una vez definidas las variables podemos definir la función objetivo, que en este caso es la minimización del C_{max} . El C_{max} en el taller de flujo se puede calcular, como se ha visto, como el tiempo en el que el último trabajo abandona la última máquina ($O_{m,\pi(n)} f$). Esta cantidad se puede expresar como el instante en el que la máquina m empieza a procesar el primer trabajo más el tiempo de proceso de todos los trabajos y el tiempo ocioso en la máquina m . Dado que el tiempo de proceso de todos los trabajos en la máquina m es constante, se puede eliminar con lo que la función objetivo queda:

$$C_{max} = \min \left\{ \sum_{i=1}^{m-1} \sum_{j=1}^n X_{ji} \cdot p_{ij} + \sum_{j=1}^{n-1} I_{mj} \right\}$$

El primer término calcula el tiempo que transcurre desde que se empieza el primer trabajo en la primera máquina hasta que se termina el primer trabajo en la máquina $m - 1$, o lo que es lo mismo, el tiempo ocioso que existe en la máquina m antes de empezar a procesar el primer trabajo de la secuencia. El segundo término calcula el tiempo ocioso en la máquina m cuando ya se están procesando los trabajos.

A partir de la definición de variables, y para cumplir los supuestos del taller de flujo, detallamos las siguientes restricciones:

- Cada trabajo debe asignarse a una única posición de la secuencia:

$$\sum_{j=1}^n X_{jk} = 1, \quad k = (1, \dots, n)$$

- Cada posición de la secuencia tiene asignado un único trabajo:

$$\sum_{k=1}^n X_{jk} = 1, \quad j = (1, \dots, n)$$

- Los trabajos no pueden solaparse en las máquinas. Además, mientras la máquina i termina el trabajo en la posición $k + 1$, la máquina $i + 1$ debe terminar el trabajo en la posición k o habrá retrasos, es decir, se debe cumplir la siguiente relación: $I_{ik} + \sum_{j=1}^n X_{j,k+1} \cdot p_{ij} + W_{i,k+1} = W_{ik} + \sum_{j=1}^n X_{j,k} \cdot p_{i+1,j} + I_{i+1,k}$, por lo que añadiremos un grupo de restricciones de la forma:

$$I_{ik} + \sum_{j=1}^n X_{j,k+1} \cdot p_{ij} + W_{i,k+1} - W_{ik} - \sum_{j=1}^n X_{j,k} \cdot p_{i+1,j} - I_{i+1,k} = 0$$

para todo $k = (1, \dots, n - 1)$ e $i = (1, \dots, m - 1)$.

De esta manera, en general tendremos un total de $mn + n - m + 1$ restricciones. Como podemos observar, el modelo no es demasiado complejo, otros problemas de programación de la producción se pueden también modelizar de manera sencilla (ver Błazewicz, Dror y Węglarz, 1991).

Para resolver el ejemplo anterior hemos programado este modelo en el lenguaje de

modelización de LINGO (Schrage, 1998 y Lindo Systems Inc., 1999). El listado del modelo se muestra a continuación:

```

!Modelo FlowShop simple;
!Basado en "Scheduling, Theory, Algorithms and Systems", M. Pinedo 2002;

MODEL:

TITLE: FLOW_SHOP;

! Parámetros fijos del modelo, ajustables según la instancia;
DATA:
  NJOBS=5;
  NMACHINES=4;
ENDDATA

SETS:
  JOBS/1..NJOBS/; ! TRABAJOS;
  MACHINES/1..NMACHINES/; ! MÁQUINAS;
  ASSIGN(JOBS,JOBS): X; ! Variables de asignación;
  IDLE(MACHINES,JOBS)|&2 #LT# NJOBS: I; !tiempos ociosos en las máquinas;
  WAIT(MACHINES,JOBS)|&1 #LT# NMACHINES: W; !tiempos de espera de los
    trabajos;
  PROC(MACHINES,JOBS): P; ! Tiempos de proceso de los trabajos en las
    máquinas;
ENDSETS

! p_ij, tiempos de proceso, ajustables según la instancia;
DATA:
  P=
  31      19      23      13      33
  41      55      42      22       5
  25       3      27      14      57
  30      34       6      13     19;
ENDDATA

! MINIMIZAR MakeSpan (Cmax);
[OBJ] MIN = CMAX;

CMAX=@SUM ( ASSIGN(I,J)| I #LT# NMACHINES: X(J,1)*P(I,J))
  +@SUM( IDLE(J,K)| J #EQ# NMACHINES: I(J,K))
  +@SUM( ASSIGN(L,J): X(J,L)*P(NMACHINES,J));

! posición como mucho con un trabajo asignado;
@FOR( JOBS( K): [POS]
  @SUM( JOBS( J): X( J, K)) = 1;
);

! Cada trabajo asignado como mucho a una posición;
@FOR( JOBS( J): [JOB]
  @SUM( JOBS( K): X( J, K)) = 1;
);

! Relación entre las variables de decisión y las restricciones físicas;
@FOR( JOBS(K)| K #LT# NJOBS:

```

```

@FOR( MACHINES(L) | L #LT# NMACHINES: [MACH]
      I(L,K) + @SUM( JOBS(J): X(J,K+1)*P(L,J)) + W(L,K+1)
      - W(L,K) - @SUM( JOBS(J): X(J,K)*P(L+1,J)) - I(L+1,K)=0;
   );
);

! No puede haber tiempo ocioso en la máquina 1;
@FOR(IDLE(L,K) | L #EQ# 1: [EQ] I(L,K)=0 );

! El trabajo 1 no tiene por qué esperar;
@FOR(WAIT(L,K) | K #EQ# 1: [EQ2] W(L,K)=0 );

! Las variables de asignación son binarias;
@FOR( ASSIGN: @BIN(X));

END

```

Listado 3.1 – Programa en lenguaje LINGO para el modelo de Wagner.

Como podemos observar en el listado, al utilizar el lenguaje de modelización de LINGO es posible resolver cualquier ejemplo sin necesidad de cambiar el modelo. En el listado aparecen, dentro de las secciones de DATA, los datos para el ejemplo que venimos utilizando. Una vez resuelto este ejemplo, obtenemos la solución mostrada en la Figura 3.7.

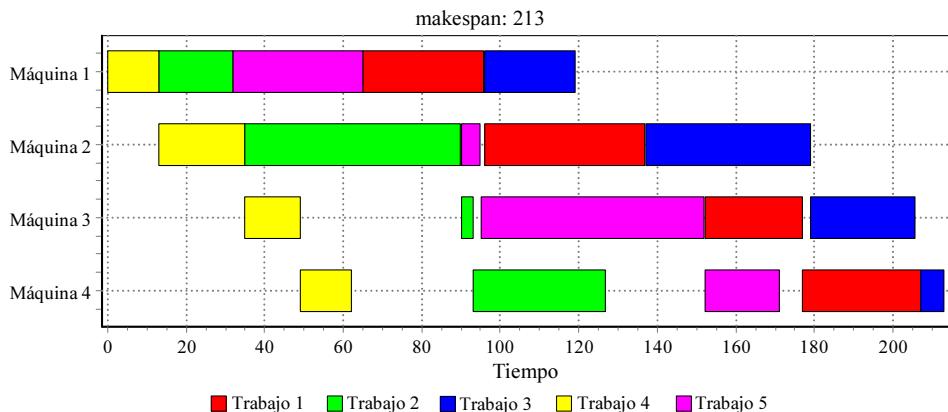


Figura 3.7 – Solución óptima para el ejemplo de 4 máquinas y 5 trabajos resuelto con el modelo de Wagner.

Podemos ver que la solución óptima tiene un C_{max} de 218, mientras que la solución inicial de la Figura 3.6 tenía un “makespan” de 286, luego la solución óptima es un 31,2 % mejor. El problema de este modelo es que el tiempo de cálculo necesario crece exponencialmente con el tamaño del problema. Se ha realizado un experimento con 12 ejemplos de tamaño muy reducido que van desde 10 trabajos y 5 máquinas hasta 15 trabajos y 10 máquinas. Para cada uno de estos problemas se ha construido el modelo de acuerdo con el Listado 3.1 y se ha resuelto con el programa LINGO versión 6.0. Los resultados se muestran en la Tabla 3.5.

Problemas ($n \times m$)	Variables totales	Variables enteras	Restricciones	Iteraciones realizadas	Tiempo (hh:mm:ss) ¹
Test(10x5)	173	100	58	60.262	00:00:09
Test(11x5)	202	121	64	252.509	00:00:32
Test(12x5)	233	144	70	959.268	00:02:16
Test(13x5)	266	169	76	7.128.305	00:18:36
Test(14x5)	301	196	82	13.411.107	00:35:39
Test(15x5)	338	225	88	240.814.154	20:28:47
Test(10x10)	263	100	103	250.437	00:00:43
Test(11x10)	302	121	114	1.524.179	00:04:33
Test(12x10)	343	144	125	3.191.230	00:10:31
Test(13x10)	386	169	136	5.110.551	00:18:57
Test(14x10)	431	196	147	72.917.322	04:42:32
Test(15x10)	478	225	158	466.407.767	94:13:57 ²

Tabla 3.5 – Evaluación de las prestaciones del modelo de Wagner con LINGO v. 6.0.

Como podemos observar, resulta muy llamativo cómo pasar de un problema con 14 trabajos y 5 máquinas a un problema de 15 trabajos y 5 máquinas conlleva

¹Las pruebas se realizaron en un ordenador tipo PC/AT con un procesador AMD Athlon 2000+XP (1666 MHz) y 512 Mbytes de memoria RAM.

²Lingo v. 6.0. abortó sin obtener la solución óptima en este ejemplo tras agotar toda la memoria disponible.

un aumento de casi 20 horas de tiempo de proceso. También, resolver un problema con el doble de máquinas no cuesta el doble de tiempo, sino muchas veces más. Como conclusión vemos que el tiempo necesario para obtener una solución óptima crece muy rápido con el tamaño del problema pese a haber utilizado un ordenador de última generación. De hecho, el ejemplo de 15 trabajos y 10 máquinas no pudo resolverse dado que el ordenador se quedó sin memoria disponible después de casi cuatro días de cálculos. Luego existe el problema añadido de la memoria física disponible además del tiempo de proceso.

Pese a estos problemas, se han venido desarrollando numerosos algoritmos exactos para el taller de flujo de permutación y algunas variantes. Los métodos de Bifurcación y Acotación (“*Branch & Bound*”) son un claro ejemplo. Los dos primeros algoritmos de este tipo que se publicaron son los trabajos de Ignall y Schrage (1965) y de Lomnicki (1965). Otros trabajos en la línea de los algoritmos B&B son los artículos de Brown y Lomnicki (1966), McMahon y Burton (1967), Nabeshima (1967), Bestwick y Hastings (1976) y Lageweg, Lenstra y Rinnooy Kan (1978). En todos estos trabajos la situación obtenida en el experimento anterior se repite y los autores no consiguen resolver problemas de más de siete u ocho trabajos y cinco o seis máquinas. La investigación en técnicas exactas para el taller de flujo se ralentizó considerablemente cuando se publicaron los resultados de la Teoría de la Complejidad, momento en el cual la comunidad científica internacional reorientó sus esfuerzos hacia técnicas aproximadas. No obstante, algunos trabajos sobre técnicas exactas aparecieron con posterioridad, por ejemplo Potts (1980) propuso una nueva regla de ramificación para un algoritmo Branch & Bound y Selen y Hott (1986) mostraron una formulación de programación entera mixta y por metas. Frieze y Yadegar (1989) publicaron una nueva formulación para el problema del flowshop basada en programación entera. Más recientemente, Carlier y Rebaï (1996) han publicado dos algoritmos tipo Branch & Bound para el taller de flujo de permutación. Aún así, estos últimos resultados, aunque contienen numerosas aportaciones, siguen sin poder resolver problemas de más de nueve trabajos en un tiempo razonable.

3.1.4. Técnicas heurísticas

Como veremos, desde el trabajo original de Johnson se han propuesto un elevado número de algoritmos heurísticos para resolver el problema del taller de flujo. La motivación principal ha sido los problemas de las técnicas exactas comentados en la sección anterior.

En una primera clasificación, y de acuerdo con los trabajos publicados, las técnicas heurísticas se dividen en *heurísticas constructivas* y *heurísticas de mejora*. Las primeras construyen una secuencia (permutación) partiendo de una secuencia vacía y las segundas “mejoran” una secuencia generada previamente, normalmente a través de intercambios de trabajos o aplicando conocimiento específico del problema. Todas las heurísticas de mejora que se comentan en esta sección son deterministas y no tienen un criterio de parada fijo, es decir, se aplica un algoritmo y siempre se obtiene el mismo resultado, de ahí que las diferenciamos de las técnicas metaheurísticas que se verán más adelante.

3.1.4.1. Heurísticas constructivas

Page (1961) propuso tres métodos heurísticos para el $F/prmu/C_{max}$ basados en técnicas de ordenación. La idea es ir ordenando parejas o grupos de trabajos buscando un mejor valor de la función objetivo. Los tres algoritmos de ordenación que propone son “pairing”, “merging” y “exchanging”. A partir de las pruebas computacionales parece que el método “merging” supera ligeramente a los otros dos, aunque es el más costoso de los tres. El tamaño de los ejemplos utilizados en su evaluación (hasta 32 trabajos y 8 máquinas) es demasiado pequeño como para poder extrapolar los resultados.

Dudek y Teuton (1964) desarrollaron unas reglas de secuenciación para el taller de flujo donde se busca minimizar el tiempo ocioso en la última máquina. El trabajo está basado en las ideas de Johnson pero extendido al caso de m máquinas. Otra heurística muy citada en la literatura es la de Palmer (1965). En este método se calcula un índice de “pendiente” o “slope index” para cada trabajo y la secuencia se obtiene ordenando los trabajos por orden decreciente del índice. Concretamente, se busca dar prioridad a los trabajos que tienen tendencia a tener tiempos cortos de proceso en las primeras etapas productivas y tiempos largos de proceso en las

últimas etapas del proceso. Es interesante observar cómo se calcula este índice, que llamamos SI :

$$SI_j = \sum_{i=1}^m \left(\frac{2i - m - 1}{2} \right) p_{ij}, \quad j = (1, \dots, n)$$

Un algoritmo muy extendido y citado es la heurística de Campbell, Dudek y Smith (1970). Básicamente, este algoritmo, conocido como CDS, aplica las ideas de la regla de Johnson y la extensión a m máquinas que se vio en la Sección 3.1.1. Concretamente, se ejecuta la regla de Johnson un total de $m - 1$ veces, y una forma especial de agrupar las m máquinas para formar un problema “virtual” de 2 máquinas. Por ejemplo, si tenemos que $m = 5$ podremos agrupar estas cinco máquinas en dos máquinas virtuales, llamadas m'_1 y m'_2 de la siguiente manera:

$$\begin{array}{ll} m'_1 = \{1\} & m'_2 = \{5\} \\ m'_1 = \{1, 2\} & m'_2 = \{4, 5\} \\ m'_1 = \{1, 2, 3\} & m'_2 = \{3, 4, 5\} \\ m'_1 = \{1, 2, 3, 4\} & m'_2 = \{2, 3, 4, 5\} \end{array}$$

Cada una de estas $m - 1$ agrupaciones constituye un “problema virtual” de dos máquinas que resolvemos con el algoritmo de Johnson. Los tiempos de proceso de las dos máquinas virtuales se calculan de la siguiente manera para todas las posibles agrupaciones k :

$$p_{1j}^k = \sum_{i=1}^k p_{ij}, \quad p_{2j}^k = \sum_{i=m-k+1}^m p_{ij}, \quad j = (1, \dots, n), \quad k = (1, \dots, m-1)$$

Para cada una de las $m - 1$ secuencias generadas con la regla de Johnson, se calcula el C_{max} con los tiempos de proceso del problema original de m máquinas y se devuelve aquella que tenga el menor C_{max} . La heurística CDS se comparó por los autores con la de Palmer, resultando esta última menos eficiente. Sin embargo, este resultado no sorprende, dado que la la heurística de Palmer solamente genera una única secuencia y en un tiempo muy corto, y como hemos visto, la heurística

CDS genera $m - 1$ secuencias en un tiempo previsiblemente mayor.

Gupta (1971) presentó una técnica heurística muy sencilla que funciona de manera muy parecida al método de Palmer. Gupta propuso un índice alternativo como sigue:

$$GI_j = \frac{A}{\min_{1 \leq i \leq (m-1)} (p_{ij} + p_{i+1,j})}, \quad j = (1, \dots, n)$$

donde $A = \begin{cases} 1, & \text{si } p_{mj} \leq p_{1j} \\ -1, & \text{en otro caso} \end{cases}$

Los trabajos se ordenan de forma ascendente según GI_j para obtener la secuencia final. Según el trabajo de Gupta, este método es más eficaz que el de Palmer, entendiendo como eficacia el obtener un mejor valor para el criterio de optimización considerado. El mismo autor (Gupta, 1972), propuso tres métodos heurísticos. Resulta interesante el hecho de que este trabajo es uno de los primeros en considerar un objetivo alternativo. Gupta tiene en cuenta dos objetivos, \bar{F} y F_{max} , aunque no simultáneamente, además, en el contexto del artículo, éste último objetivo es equivalente a C_{max} . Los tres métodos propuestos se llaman MINIT (MINimum Idle Time) que busca minimizar tiempo ocioso entre máquinas, MICOT (MIminimum COMpletion Time) que secuencia trabajos de acuerdo a los tiempos de finalización y el método MINIMAX, que se basa en la regla de Johnson. Los tres algoritmos resultan ser mejores que el método CDS para el criterio \bar{F} pero inferiores con el objetivo C_{max} .

Bonney y Gundry (1976), trabajaron con la idea de las propiedades geométricas de los trabajos cuando se disponen en un diagrama de Gantt y formularon un algoritmo que busca alinear las pendientes que forman la recta que une los inicios de las tareas y la recta que une los finales de las tareas. En sus pruebas, este algoritmo resultó ser superior al de Palmer y al de Gupta.

En 1977, Dannerbring propuso tres algoritmos, el primero de ellos es de tipo constructivo mientras que los dos últimos son de mejora y se tratarán más adelante. El algoritmo constructivo se denomina “*Rapid Access*” (RA) y aúna las ideas de los algoritmos de Johnson y de Palmer. En este caso se construye un problema con dos máquinas virtuales, como ya se hiciera en el método CDS, pero

en vez de aplicar la regla de Johnson directamente sobre los tiempos de proceso acumulados de cada trabajo en las máquinas virtuales, se calculan unos nuevos tiempos de proceso de acuerdo a la siguiente fórmula:

$$p_{1j} = \sum_{i=1}^m (m - i + 1)p_{ij}, \quad p_{2j} = \sum_{i=1}^m (i) \cdot p_{ij}, \quad j = (1, \dots, n)$$

Podemos ver que los nuevos tiempos de proceso imponen una ponderación que recuerda el método de Palmer.

Miyazaki, Nishiyama y Hashimoto (1978) propusieron un sencillo algoritmo que se basa en el intercambio de trabajos adyacentes para obtener una buena secuenciación. En este caso el criterio de optimización considerado es \bar{F} , aunque el algoritmo es general y se podría adaptar al C_{max} . Los resultados proporcionados por los autores son para problemas muy pequeños de 6 trabajos y 6 máquinas, por lo que no está claro si el algoritmo sería igual de eficiente y/o eficaz para ejemplos de mayor tamaño.

En un trabajo posterior, Gelders y Sambandam (1978), evaluaron un total de cuatro técnicas heurísticas utilizando un criterio de optimización poco usual; la minimización del retraso ponderado ($\sum_{j=1}^n w_j T_j$) y también los costes ponderados de tiempo de flujo. No se proporcionan resultados comparados con respecto a otros métodos puesto que era la primera vez que se consideraban estos criterios de optimización.

King y Spachis (1980) presentaron varias heurísticas, tanto para el $F//C_{max}$ como para el $F/nwt/C_{max}$. Estas técnicas heurísticas están basadas en elaboradas reglas de prioridad. En el caso del taller de flujo de permutación se presentan cinco métodos que resultan ser de una eficacia muy parecida a la heurística CDS, teniendo esta última una ligera ventaja en los ejemplos más grandes de entre los probados.

Una aproximación diferente se puede encontrar en los seis métodos heurísticos propuestos por Stinson y Smith (1982). En este caso, para cada par de trabajos que pueden ir juntos en la secuencia se define una matriz de costes C_{jk} , $j, k = (1, \dots, n)$. Cada uno de los seis métodos calcula de una manera distinta la matriz C_{jk} . Una vez se ha obtenido la citada matriz se resuelve el problema haciendo una

analogía con el problema del viajante de comercio con alguna ligera modificación y aplicando una versión del algoritmo de Vogel para problemas de transporte. La heurística más eficaz de entre las seis consideradas se comporta mejor que las heurísticas CDS, de Palmer y de Page.

Sin lugar a dudas, uno de los métodos más ampliamente citados y conocidos para el problema del taller de flujo de permutación es la heurística NEH de Nawaz, Enscore y Ham (1983). Este método ha sido reconocido en varias ocasiones como el más eficaz (ver Turner y Booth, 1987, o Taillard, 1990). La idea principal es que los trabajos con un alto tiempo total de proceso en las máquinas deben secuenciarse lo antes posible.

Este algoritmo tiene un papel clave en los métodos desarrollados en esta Tesis por lo que se expone con detalle a continuación:

1. Primero se calculan los tiempos totales de proceso de los trabajos en todas las máquinas:

$$P_j = \sum_{i=1}^m p_{ij}, \quad j = (1, \dots, n)$$

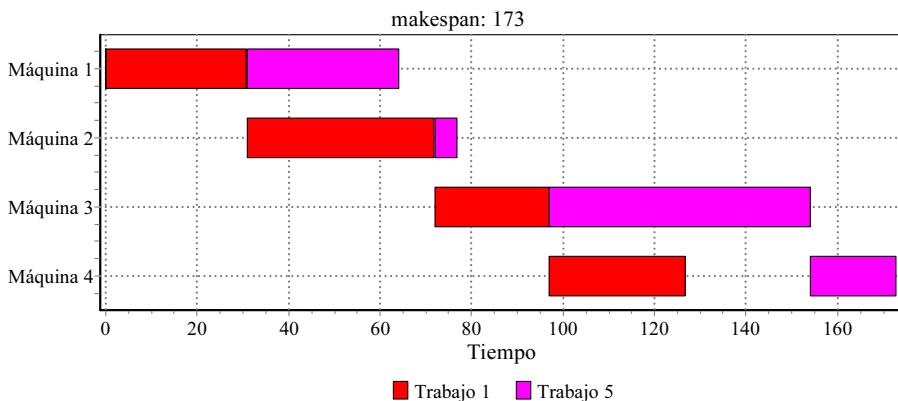
2. Los trabajos se ordenan descendenteamente de acuerdo con P_j y se almacenan en una lista ℓ . Se extraen los dos primeros trabajos de ℓ , $\ell_{(1)}$ y $\ell_{(2)}$, estos dos trabajos serán los de mayor P_j . Se evalúan las dos posibles secuencias parciales que contienen $\ell_{(1)}$ y $\ell_{(2)}$, o lo que es lo mismo, calculamos el “makespan” de las secuencias de permutación $\{\ell_{(1)}, \ell_{(2)}\}$ y $\{\ell_{(2)}, \ell_{(1)}\}$. De las dos secuencias, se escoge para el siguiente paso aquella con menor C_{max} .
3. Extraemos el trabajo k de ℓ , $\ell_{(k)}$, $k = (3, \dots, n)$ y buscamos la secuencia parcial con menor C_{max} resultante de insertar el trabajo $\ell_{(k)}$ en todas las posibles k posiciones de la mejor secuencia parcial encontrada hasta el momento. Por ejemplo, vamos a suponer que la mejor secuencia parcial encontrada hasta el momento es $\pi_p = \{\ell_{(1)}, \ell_{(3)}, \ell_{(2)}\}$ y que $k = 4$, entonces evaluaremos las secuencias $\pi_{p+\ell_{(4)}}^1 = \{\ell_{(4)}, \ell_{(1)}, \ell_{(3)}, \ell_{(2)}\}$, $\pi_{p+\ell_{(4)}}^2 = \{\ell_{(1)}, \ell_{(4)}, \ell_{(3)}, \ell_{(2)}\}$, $\pi_{p+\ell_{(4)}}^3 = \{\ell_{(1)}, \ell_{(3)}, \ell_{(4)}, \ell_{(2)}\}$ y $\pi_{p+\ell_{(4)}}^4 = \{\ell_{(1)}, \ell_{(3)}, \ell_{(2)}, \ell_{(4)}\}$ quedándonos para el siguiente paso ($k = 5$)

con aquella que tenga un menor C_{max} .

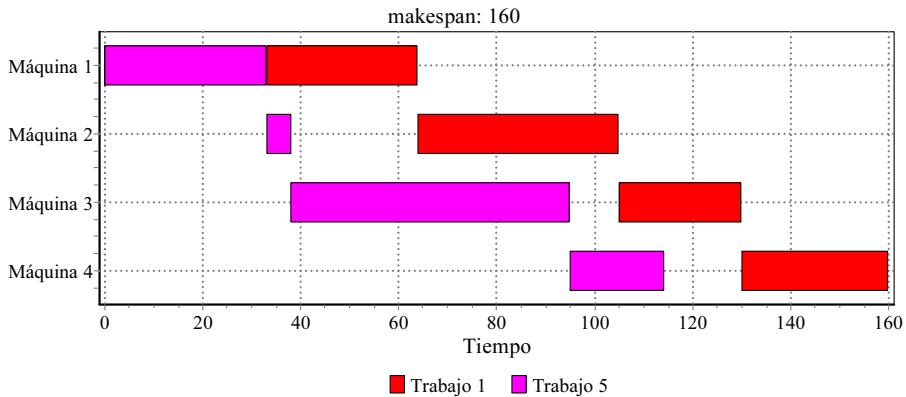
Como podemos observar, la heurística NEH se basa en inserciones de trabajos y ésta es su principal debilidad. En general hay que evaluar un total de $(n(n + 1))/2 - 1$ secuencias, de las cuales sólo n son secuencias completas. La complejidad computacional de este método es de $\mathcal{O}(n^3m)$ que puede ser demasiado para problemas de tamaño medio o grande. Afortunadamente, Taillard (1990) propuso una manera muy eficiente de calcular la mejor secuencia para un k dado en el paso 3 del método sin tener que evaluar las k secuencias parciales, reduciendo la complejidad computacional a $\mathcal{O}(n^2m)$.

Para comprender mejor la heurística NEH, vamos a resolver el sencillo ejemplo de la Tabla 3.4 aparecida en la Sección 3.1.3.

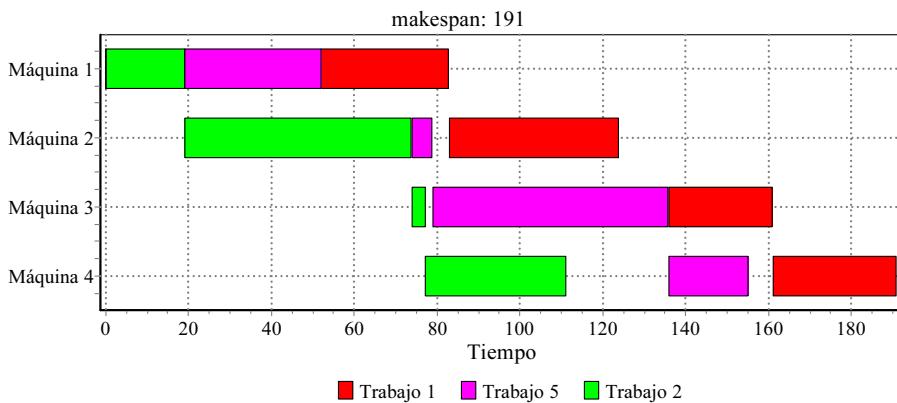
El primer paso requiere el cálculo de los P_j , que quedan: $P_1 = 31 + 41 + 25 + 30 = 127$, $P_2 = 111$, $P_3 = 98$, $P_4 = 62$ y $P_5 = 114$. Pasando al paso 2 ordenamos los trabajos descendientemente por el valor de P_j , con lo que obtenemos la lista ordenada $\ell = \{1, 5, 2, 3, 4\}$. Extraemos los dos primeros trabajos de ℓ , es decir, $\ell_{(1)} = 1$ y $\ell_{(2)} = 5$. Vamos a evaluar las dos secuencias parciales $\{1, 5\}$ y $\{5, 1\}$. Los diagramas de Gantt para las secuencias aparecen en la Figura 3.8.



(a) Secuencia $\{1, 5\}$ con un C_{max} de 173.

(b) Secuencia $\{5, 1\}$ con un C_{max} de 160.**Figura 3.8** – Paso 2 de la heurística NEH para el ejemplo.

Podemos observar que $C_{max}(\{5, 1\}) < C_{max}(\{1, 5\})$, por lo que la secuencia parcial $\{5, 1\}$ queda fijada para el resto del algoritmo. Empezando con el paso 3 y haciendo $k = 3$ tenemos que $\ell_{(3)} = 2$ y hay tres posibles posiciones donde colocar el trabajo 2 y por tanto tres secuencias parciales que se muestran en la Figura 3.9.

(a) Secuencia $\{2, 5, 1\}$ con un C_{max} de 191.

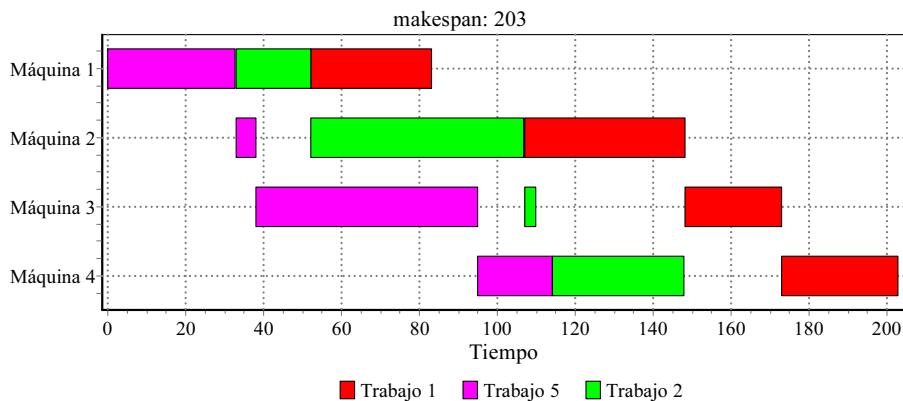
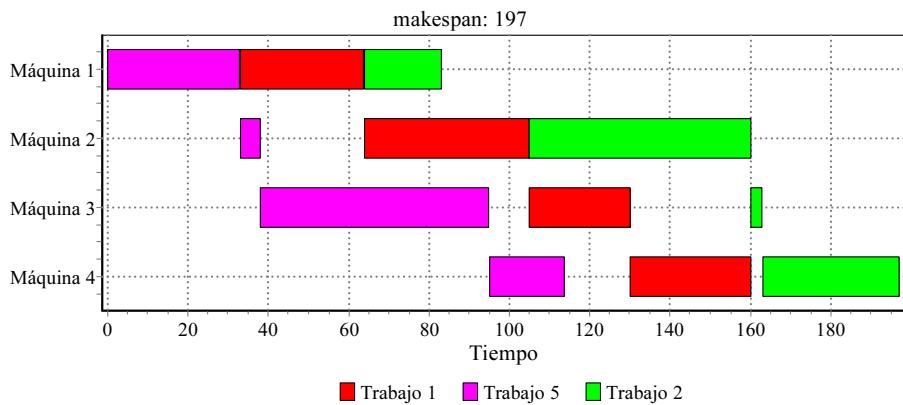
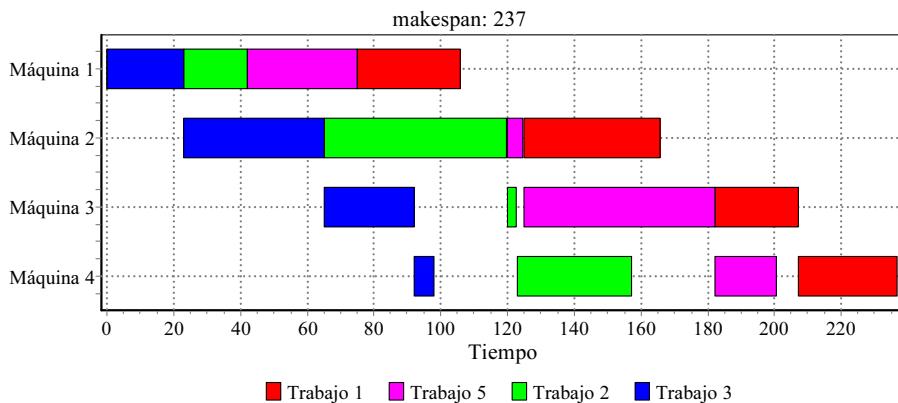
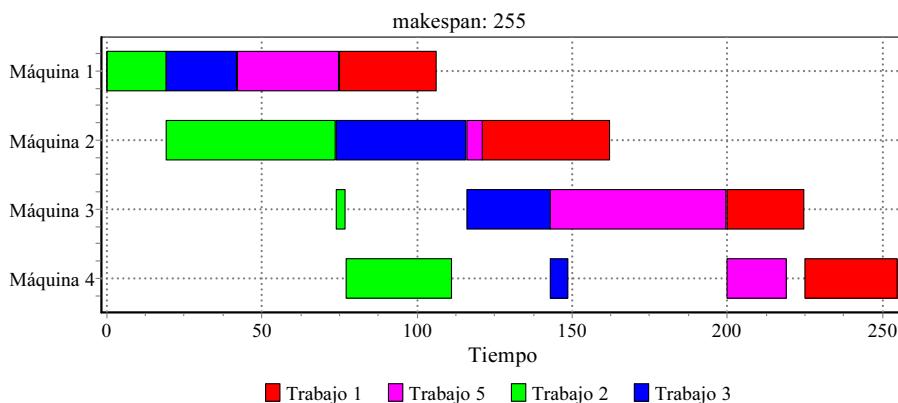
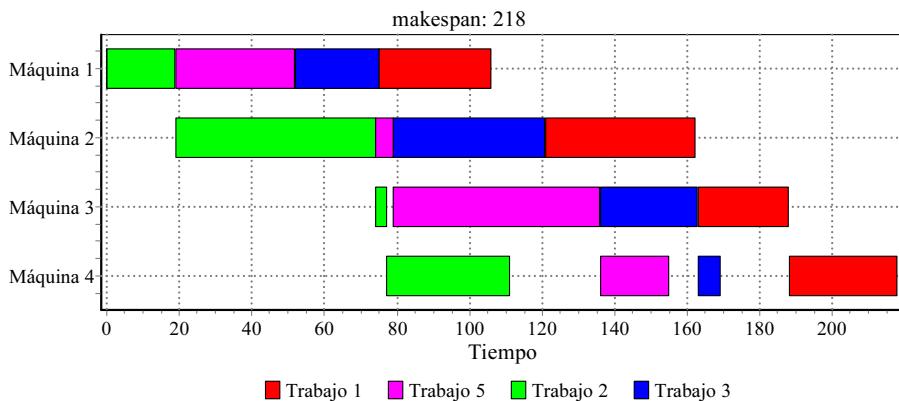
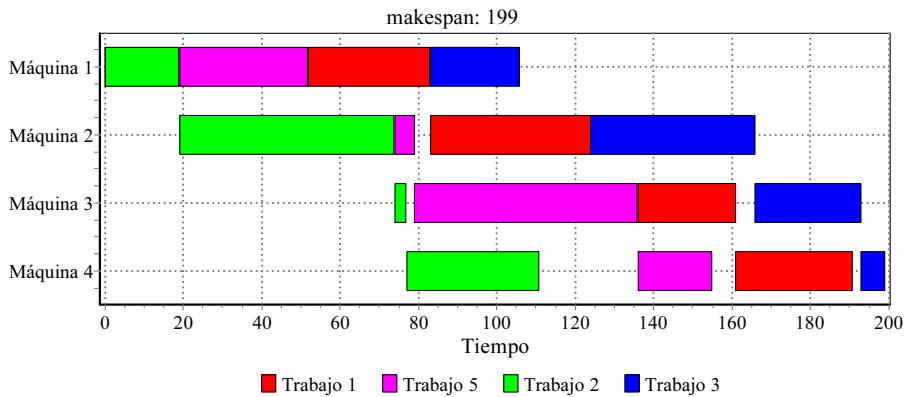
(b) Secuencia $\{5, 2, 1\}$ con un C_{max} de 203.(c) Secuencia $\{5, 1, 2\}$ con un C_{max} de 197.

Figura 3.9 – Paso 3 de la heurística NEH para el ejemplo,
 $k = 3$.

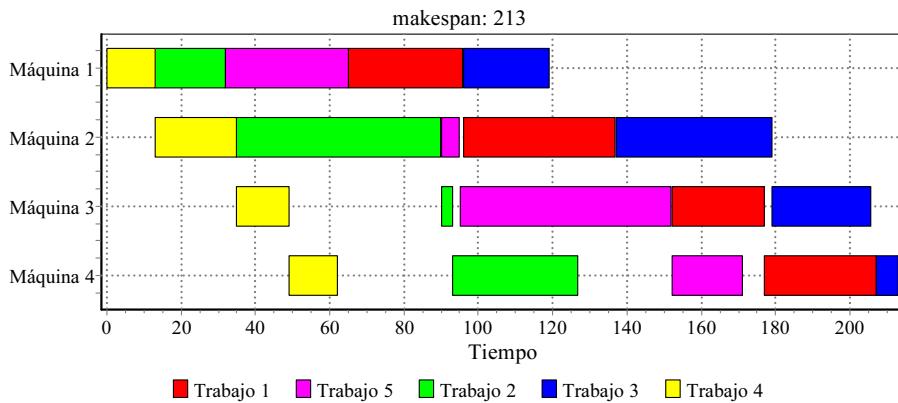
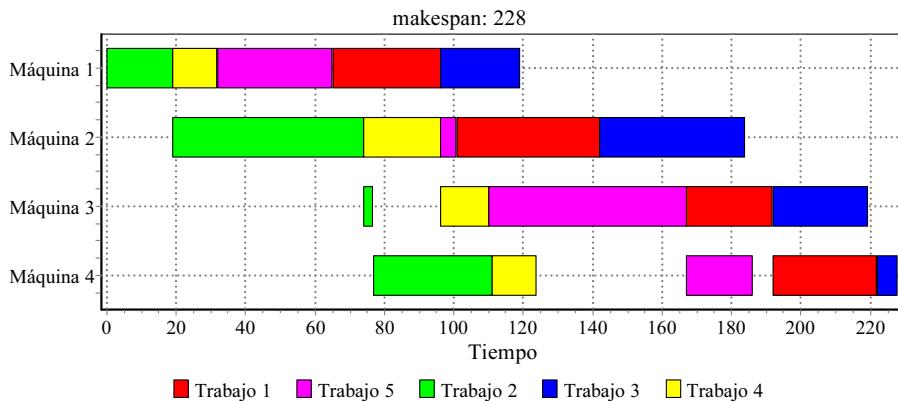
Como vemos, la secuencia parcial que menor C_{max} tiene es la secuencia $\{2, 5, 1\}$, que queda fijada para la siguiente iteración. Hacemos $k = 4$, por lo que $\ell_{(4)} = 3$. Existen 4 secuencias parciales resultado de colocar el trabajo 3 en las 4 posibles posiciones. Dichas secuencias, junto con sus “makespans” asociados se muestran en la Figura 3.10.

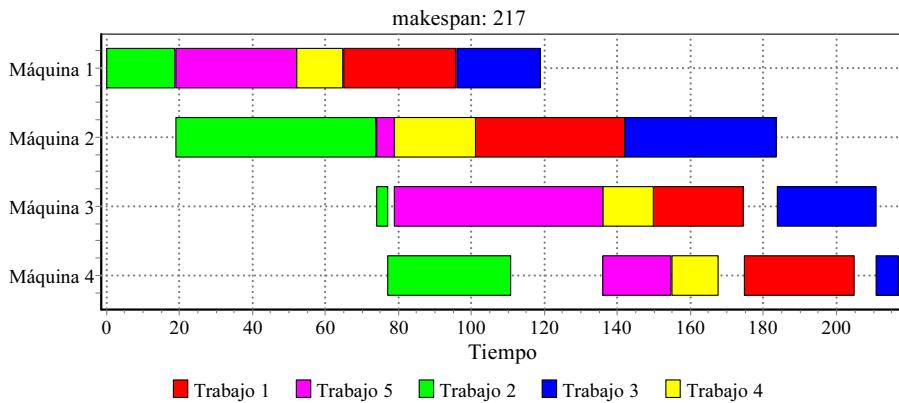
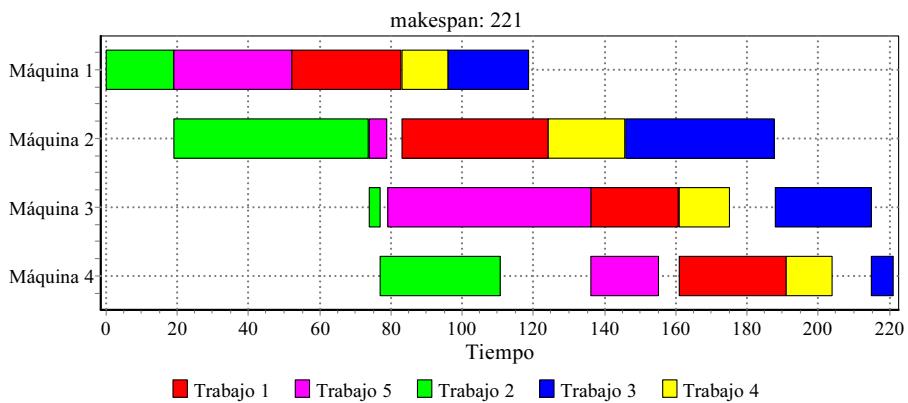
(a) Secuencia $\{3, 2, 5, 1\}$ con un C_{max} de 237.(b) Secuencia $\{2, 3, 5, 1\}$ con un C_{max} de 255.

(c) Secuencia $\{2, 5, 3, 1\}$ con un C_{max} de 218.(d) Secuencia $\{2, 5, 1, 3\}$ con un C_{max} de 199.

**Figura 3.10 – Paso 3 de la heurística NEH para el ejemplo,
 $k = 4$.**

La última de las anteriores secuencias parciales, $\{2, 5, 1, 3\}$ tiene C_{max} mínimo, luego la escogemos para la última iteración. Hacemos $k = 5$, $\ell_{(5)} = 4$, hay $n = k$ posibles posiciones donde colocar el trabajo 4, calculamos las secuencias, que en este caso ya no son parciales, puesto que cada una tiene n trabajos. Los diagramas de Gantt para estas secuencias aparecen en la Figura 3.11.

(a) Secuencia $\{4, 2, 5, 1, 3\}$ con un C_{max} de 213.(b) Secuencia $\{2, 4, 5, 1, 3\}$ con un C_{max} de 228.

(c) Secuencia $\{2, 5, 4, 1, 3\}$ con un C_{max} de 217.(d) Secuencia $\{2, 5, 1, 4, 3\}$ con un C_{max} de 221.

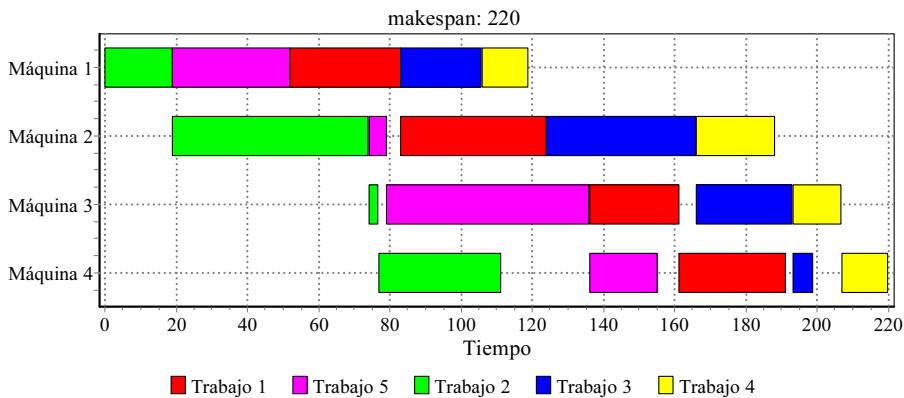
(e) Secuencia $\{2, 5, 1, 3, 4\}$ con un C_{max} de 220.

Figura 3.11 – Paso 3 de la heurística NEH para el ejemplo,
 $k = n$, última iteración.

En este último paso, la primera secuencia es la que menor C_{max} tiene, luego, tras aplicar la heurística NEH, la secuencia obtenida para el ejemplo es $\{4, 2, 5, 1, 3\}$ con un C_{max} de 213. Como hemos visto, la heurística NEH es muy intuitiva y fácil de aplicar. Muchos autores han estudiado este método y lo han aplicado al problema del taller de flujo con otros criterios de optimización o incluso a variantes del problema como veremos en el Capítulo 5.

Otro algoritmo es la extensión a la heurística de Palmer realizada por Hundal y Rajgopal (1988). La extensión se basa en un problema que tiene la heurística de Palmer cuando m es un número impar. En este caso, el índice de Palmer devuelve un valor de cero para la máquina $(m+1)/2$, ignorándola en la secuenciación. Para solucionar este problema, Hundal y Rajgopal proponen dos índices adicionales al de Palmer, que se calculan como sigue:

$$SI_j^1 = \sum_{i=1}^m (m - 2i)p_{ij}$$

$$SI_j^2 = \sum_{i=1}^m (m - 2i + 2)p_{ij}$$

$$j = (1, \dots, n)$$

Con estos dos índices y el original de Palmer se calculan tres secuencias y la mejor de las tres se devuelve como resultado.

Rajendran y Chaudhuri (1991) propusieron tres heurísticas competitivas considerando el criterio de minimización del tiempo total de flujo ($\sum_{j=1}^n F_j$). Este criterio se corresponde con el criterio \bar{F} dado que n es constante y se puede eliminar de la función objetivo. Las tres heurísticas se comparan favorablemente con el algoritmo de Miyazaki, Nishiyama y Hashimoto y con las heurísticas de Gupta. Otra heurística fue propuesta por Sarin y Lefoka en 1993. El método busca minimizar el tiempo ocioso en la última máquina de la secuencia, de manera que se da prioridad a aquellos trabajos que, una vez insertados en la secuencia resulten en un incremento mínimo de este tiempo ocioso. Los resultados indican que el método propuesto es más eficaz que la heurística NEH pero sólo en ejemplos que cumplen que $m \gg n$, lo cual resulta ser poco realista. Para el resto de ejemplos la heurística NEH resulta ser mejor. Adicionalmente, se comenta en este trabajo que el método propuesto es más eficiente que la heurística NEH. Sin embargo, la versión de la heurística NEH contra la que se compara no incorpora las mejoras de Taillard (1990).

Rajendran (1993) desarrolló otro algoritmo muy parecido a NEH pero modificando la ordenación inicial de los trabajos para permitir considerar el objetivo \bar{F} en vez del C_{max} . Según los resultados, se trata de una de las mejores propuestas para este objetivo.

Kim (1993) publicó una serie de heurísticas considerando el objetivo \bar{T} (minimización del retraso medio), que están basadas en modificaciones a reglas de prioridad que pueden permitir secuencias generales. El autor compara las heurísticas generales con las de permutación y contrasta varias adaptaciones de otras heurísticas, como por ejemplo la NEH. El resultado es interesante, las reglas de prioridad que permitían secuencias generales eran mejores que las reglas de prioridad considerando sólo secuencias de permutación, aunque estas últimas mejoraban a las primeras si se aplicaba una sencilla búsqueda local.

Chen et al. (1996) desarrollaron una heurística para el $F3//C_{max}$ basada en el método de Johnson, con la particularidad de que los autores acotaron el peor resultado posible que la heurística es capaz de obtener. En este caso se asegura que las secuencias obtenidas tienen un C_{max} que es como mucho un $5/3$ superior

a el de las secuencias óptimas.

Koulamas (1998) presentó una nueva técnica heurística, llamada HFC, que se compone de dos fases. La primera fase se basa en el algoritmo de Johnson, y es parecida a la CDS. La segunda fase es una forma especial de búsqueda local, cuya principal característica es permitir el “*job passing*”, es decir, se permite que los trabajos se adelanten unos a otros en las distintas máquinas. Este hecho provoca que el resultado pueda ser una secuencia general y no una de permutación. Esto es una idea muy interesante, ya que como hemos visto, cuando $m > 3$ las secuencias de permutación no dominan a las generales. La segunda fase podría considerarse como una técnica heurística de mejora en sí misma.

Woo y Yim (1998) desarrollaron también una heurística muy parecida a la de NEH y Rajendran (1993). Simplemente modifican la lista de trabajos previa a la fase de inserción y hacen que esta lista se actualice tras cada paso de inserción, es decir, la lista es dinámica y no va prefijada. El criterio de optimización considerado es \bar{F} .

Más recientemente, Davoud Pour (2001) desarrolló una sencilla heurística basada en intercambios de trabajos. El autor compara el método desarrollado con NEH, CDS y Palmer, presentando el método desarrollado un mejor comportamiento sólo cuando el número de máquinas m es grande.

Framinan, Leisten y Ruiz-Usano (2002) han propuesto una serie de heurísticas que tienen en consideración dos objetivos simultáneamente, esto es C_{max} y F_{max} , que están basadas en la NEH y proporcionan un buen compromiso para los dos objetivos considerados. Este trabajo no es el primero publicado donde se considera más de un objetivo de optimización, existen varios artículos donde se estudia el taller de flujo con dos objetivos e incluso donde se estudian más de dos. Una completa revisión de técnicas multiobjetivo se puede encontrar en Bagchi (1999) y en T'Kindt y Billaut (2002).

Actualmente se sigue investigando activamente en el campo de las heurísticas aplicadas al taller de flujo de permutación. Por ejemplo, Framinan, Leisten y Rajendran (2003) han realizado recientemente un estudio exhaustivo de la heurística NEH, donde han propuesto un total de 177 ordenaciones iniciales para el paso 2 del método. Con estas inicializaciones se busca encontrar soluciones para los criterios C_{max} , \bar{F} y la minimización del tiempo ocioso.

3.1.4.2. Heurísticas de mejora

Como se ha comentado, las heurísticas de mejora lo que hacen es tratar de mejorar una secuencia ya construida con respecto a algún criterio de optimización dado. Dentro de este tipo de métodos, uno de los trabajos más citados es el de Dannenbring (1977). En este estudio, el autor, además de desarrollar la heurística constructiva RA ya citada, propuso dos heurísticas de mejora, la heurística “*Rapid Access with Close Order Search*” (RACS) y la heurística “*Rapid Access with Extensive Search*” (RAES). La idea de estos dos métodos es el intercambio de trabajos. Dannenbring observó que tras obtener una secuencia con la heurística RA, un simple intercambio de trabajos adyacentes mejoraba el C_{max} . De esta manera la RACS parte de una solución obtenida con RA y examina todas las posibles $n - 1$ secuencias resultado de intercambiar todo par de trabajos j y k donde $j = (1, \dots, n - 1), k = j + 1$. La mejor secuencia es el resultado. La heurística RAES es una simple extensión de la heurística RACS donde se aplica RACS repetidamente hasta que no se obtiene mejora, es decir, se aplica RACS a una secuencia obtenida con la heurística RA y luego se vuelve a aplicar RACS a la secuencia resultante. El proceso termina cuando después de aplicar RACS a una secuencia no se obtiene mejora alguna.

Ho y Chang (1991) desarrollaron una heurística de mejora con la idea de minimizar los tiempos muertos que se producen entre todo par de trabajos en cada máquina, estos tiempos, a los que los autores llaman “*gaps*” se calculan de la siguiente forma:

$$d_{jk}^i = p_{i+1,j} - p_{ik}, \quad j, k = (1, \dots, n), \quad j \neq k, \quad i = (1, \dots, m - 1)$$

De esta manera, si un trabajo j precede a otro trabajo k en la secuencia, entonces un valor positivo de d_{jk}^i implica que el trabajo k debe esperar en la máquina $i + 1$ hasta que el trabajo j termine. Un valor negativo de d_{jk}^i significa que existe un tiempo ocioso de duración d_{jk}^i entre el trabajo j y el trabajo k en la máquina $i + 1$. Estos “*gaps*” se tratan en el desarrollo de la heurística de manera que el C_{max} se minimice. Los autores comparan favorablemente el método para este objetivo y para los objetivos \bar{F} y utilización media (U) partiendo de la secuencia generada con la heurística CDS con los métodos de Palmer, Gupta, CDS, RA de

Dannenbring y con la heurística de Hundal y Rajgopal.

Recientemente, Suliman (2000) desarrolló una heurística de mejora. El método parte de la solución proporcionada por la heurística CDS y luego busca reducir el C_{max} intercambiando trabajos adyacentes. Para reducir el coste de los intercambios se impone una restricción de direccionalidad, de esta manera si adelantamos un trabajo en la secuencia y conseguimos reducir el C_{max} , se asume que se podrá seguir mejorando la secuencia adelantando el trabajo, nunca retrasándolo. La heurística se compara favorablemente con el método NEH, aunque es bastante menos intuitiva y difícil de implementar.

3.1.5. Técnicas metaheurísticas

Existen muchos tipos de técnicas metaheurísticas, las hemos separado de las heurísticas de mejora principalmente porque en el contexto de programación de la producción en general y en el problema del taller de flujo en particular, los métodos metaheurísticos que se han publicado tienen un criterio de parada establecido, sea el tiempo de CPU, el número de C_{max} evaluados, número de generaciones o número de iteraciones de un algoritmo dado. Una primera clasificación se puede hacer a raíz de la naturaleza de las técnicas consideradas. Concretamente distinguiremos entre técnicas de recocido simulado o “*Simulated Annealing*” (SA), búsqueda tabú o “*Tabu Search*” (TS), algoritmos genéticos (GA) y otros. Muchas de estas técnicas se basan de alguna manera en alguna forma de búsqueda local, lo que conlleva la definición de algún tipo de vecindario para una solución.

Se puede definir un vecindario como el conjunto de secuencias que se pueden generar a partir de una secuencia base mediante “movimientos”. Por ejemplo, el vecindario de *intercambio* se puede definir de la siguiente manera: supongamos una secuencia $\pi = \{\pi_{(1)}, \pi_{(2)}, \dots, \pi_{(n)}\}$, una nueva secuencia o vecino se puede obtener seleccionando dos trabajos al azar en las posiciones j y k donde $j < k$, $j, k = (1, \dots, n)$ e intercambiando sus posiciones, la secuencia resultado será $\pi' = \{\pi_{(1)}, \dots, \pi_{(j-1)}, \pi_{(k)}, \pi_{(j+1)}, \dots, \pi_{(k-1)}, \pi_{(j)}, \pi_{(k+1)}, \dots, \pi_{(n)}\}$. Más formalmente podemos definir un movimiento de intercambio a la dupla (j, k) donde $j < k$, $j, k = (1, \dots, n)$, luego el conjunto de movimientos de intercambio se puede definir como $E = \{(j, k) : j < k, j, k = (1, \dots, n)\}$. Es

inmediato ver que $|E| = n \cdot (n - 1)/2$, es decir, una secuencia cualquiera π tiene $n \cdot (n - 1)/2$ vecinos de intercambio. De esta manera podemos definir el vecindario de intercambio de una secuencia π como: $V(E, \pi) = \{\pi_v : v \in E\}$

3.1.5.1. Recocido simulado (“*Simulated Annealing*”)

El primer algoritmo conocido de este tipo, que fue al mismo tiempo uno de los primeros métodos metaheurísticos propuestos se debe a Osman y Potts (1989). En el desarrollo de este algoritmo se consideraron dos tipos de vecindarios, el vecindario de intercambio definido anteriormente y el vecindario de inserción. Un vecindario de *inserción* viene definido por los movimientos de inserción donde se escogen dos posiciones j y k al azar donde $j \neq k$, se extrae el trabajo en la posición j y se inserta en la posición k . Más formalmente el conjunto de movimientos de inserción I se definen como todas las posibles duplas (j, k) de posiciones donde $j \neq k$, $j, k = (1, \dots, n)$ que en una secuencia cualquiera π dan como resultado una secuencia

$$\pi' = \{\pi_{(1)}, \dots, \pi_{(j-1)}, \pi_{(j+1)}, \dots, \pi_{(k)}, \pi_{(j)}, \pi_{(k+1)}, \dots, \pi_{(n)}\}$$

si se da el caso que $j < k$ y una secuencia

$$\pi' = \{\pi_{(1)}, \dots, \pi_{(k-1)}, \pi_{(j)}, \pi_{(k+1)}, \dots, \pi_{(j-1)}, \pi_{(j+1)}, \dots, \pi_{(n)}\}$$

en el caso en que $j > k$. El conjunto de movimientos de inserción I se define entonces como $I = \{(j, k) : j \neq k, j, k = (1, \dots, n)\}$ y el vecindario de inserción como $V(I, \pi) = \{\pi_v : v \in I\}$. Podemos observar que $|I| = (n - 1)^2$.

Adicionalmente a los dos vecindarios considerados, Osman y Potts consideraron dos estrategias a la hora de examinar todos los vecinos de una secuencia dada: una evaluación “ordenada” donde se examinan todos los posibles vecinos, y una evaluación “aleatoria”, donde se examinan un número determinado x de vecinos ($x \leq |E|$ ó $x \leq |I|$). Por ejemplo, para el vecindario generado por los movimientos de inserción I una evaluación ordenada supondría considerar las posiciones $(1, 2), (1, 3), \dots, (1, n), (2, 1), (2, 3), \dots, (n - 1, n)$. Adicionalmente, el SA planteado es muy sencillo, siendo el enfriamiento muy fácil de implementar,

además, el algoritmo realiza una única iteración para cada valor de temperatura. Los autores evaluaron un total de cuatro algoritmos distintos, resultado de los dos vecindarios considerados y de las dos posibilidades de evaluación de cada vecindario. Es necesario recalcar que el algoritmo parte de una permutación aleatoria de los trabajos. El mejor algoritmo resultó ser el de vecindario de inserción y evaluación aleatoria. Los autores compararon el algoritmo con la heurística NEH y con algunos métodos de búsqueda local resultando superior el método propuesto. Ogbu y Smith (1990b) propusieron otro método SA para el mismo problema donde también se trabaja con los vecindarios E e I , pero en este caso el número de vecinos a evaluar se hace aleatoriamente dependiendo del número de trabajos n . Los autores realizaron pruebas con distintas inicializaciones, concretamente una inicialización aleatoria, con la heurística de Palmer y con las de Dannenbring. Una característica interesante de este SA es el hecho de que la función que determina la probabilidad de aceptar una nueva solución (en el caso de que sea peor que la anterior) es independiente de la diferencia de C_{max} que ésta y la anterior solución tengan y sólo depende de la iteración en la que se encuentra el algoritmo. El mejor algoritmo SA resultante trabaja con un vecindario I y con una inicialización mediante las heurísticas de Dannenbring. En otro trabajo (Ogbu y Smith, 1990a), los autores compararon este algoritmo con el de Osman y Potts. Los resultados indican una ligera superioridad de este último, principalmente en los ejemplos de mayor tamaño, aunque los autores defienden la mayor simplicidad de su algoritmo.

Zegordi, Itoh y Enkawa (1995) presentaron un complejo algoritmo tipo SA llamado SA-MDJ que incorpora conocimiento sobre el problema en forma de una tabla MDJ que contiene un índice de la idoneidad de movimiento para los trabajos (“*Move Desirability for Jobs*”). Esta tabla, junto con otras reglas adicionales facilitan el desarrollo del algoritmo. Se compara con el SA de Osman y Potts, siendo este último más eficaz pero notablemente más lento.

De una forma parecida Ishibuchi, Misaki y Tanaka (1995) proponen un algoritmo SA con una modificación importante; en cada paso, en vez de evaluar un vecino y aceptarlo de acuerdo al C_{max} y a la función de aceptación, se generan K vecinos y se escoge el mejor, de esta manera se consigue un algoritmo con menos parámetros y con un rendimiento muy similar al de Osman y Potts. En este caso

la solución inicial es aleatoria.

Recientemente, Wodecki y Bozejko (2002) han propuesto un algoritmo SA que se ejecuta en un entorno de computación paralela, los resultados son mejores que los de la heurística NEH para un subconjunto de los problemas de Taillard (1993) (que veremos con más detalle en la Sección 3.2).

3.1.5.2. Búsqueda Tabú (“*Tabu Search*”)

Widmer y Hertz propusieron en 1989 un método de tipo TS al que llamaron Spirit¹. El algoritmo se basa en dos fases, en la primera se obtiene una secuencia inicial transformando el problema del taller de flujo en un problema del viajante de comercio abierto o “*Open Travelling Salesman Problem*” (OTSP). Para poder hacer la transformación se define una distancia entre todo par de trabajos j y k de la siguiente manera:

$$d_{jk} = p_{1j} + \sum_{i=2}^m (m-i) \cdot |p_{ij} - p_{i-1,k}| + p_{km}, \quad j, k = (1, \dots, n)$$

Una vez obtenidas las distancias, se aplica un método de inserción para obtener la ruta del OTSP. El método de inserción es muy parecido al funcionamiento de la heurística NEH. La segunda fase aplica un algoritmo de búsqueda tabú. El algoritmo se basa en el vecindario E que examina por completo en cada paso del algoritmo. La lista tabú contiene un máximo de siete movimientos y se actualiza en cada iteración con el mejor movimiento tras haber analizado todo el vecindario. Cuando la lista se llena se elimina el último movimiento (lista FIFO). Los autores realizaron una completa comparación del algoritmo con las heurísticas de Johnson, Palmer, Gupta, CDS, NEH y RA y contra las mismas heurísticas aplicando también los métodos de mejora RACS y RAES de Dannerbring. El algoritmo Spirit resultó ser mucho más eficaz que los demás.

Taillard (1990) propuso un método similar. En este caso el vecindario utilizado es el I que se examina completamente en cada paso. En la lista tabú propuesta no se almacenan los movimientos sino los C_{max} de las últimas soluciones aceptadas, de

¹Este nombre viene de “*Sequencing Problem Involving a Resolution by Integrated Tabu search techniques*”.

esta manera se evita que de una secuencia se vaya a otra con un C_{max} ya visitado. En este trabajo no se evaluó el algoritmo presentado frente a otras alternativas, aunque el autor propuso algunas ideas de cómo paralelizarlo de forma eficiente, repartiendo la evaluación del vecindario en cada paso entre varios ordenadores. Reeves (1993) amplió el algoritmo Spirit incorporando interesantes mejoras. La primera y más evidente es una inicialización mediante la NEH con las mejoras que Taillard incorporó. El vecindario se cambia a I por ser el que mejores resultados proporciona, además, este vecindario se examina de manera aleatoria. El autor realizó una evaluación donde el algoritmo resultante se compara con el SA de Osman y Potts. Para las evaluaciones utilizó, por primera vez en la literatura, el conjunto de ejemplos de Taillard (1993). El resultado del algoritmo propuesto es ligeramente mejor que el algoritmo de Osman y Potts.

Moccellin (1995) presentó, en un breve artículo, un algoritmo muy parecido al Spirit de Widmer y Hertz. Realmente, la única diferencia estriba en cómo se obtiene la solución inicial. En este caso las distancias entre trabajos se calculan de una manera más elaborada y la heurística utilizada para resolver el OTSP resultante es la conocida “*Farthest Insertion Travelling Salesman Procedure*” o (FITSP). El resto del algoritmo es exactamente la segunda fase de Spirit. El autor comparó esta “mejora” con el Spirit original y, como era de esperar, obtuvo mejores resultados. Uno de los algoritmos TS más citados y conocidos se debe a Nowicki y Smutnicki (1996). La principal aportación de este trabajo es la definición de una serie de teoremas y de resultados sobre el vecindario de tipo I basándose en la idea del camino crítico y en los bloques de trabajos, de manera que se genera un vecindario I' más reducido y por tanto mucho más rápido de evaluar. Concretamente, una secuencia se divide en una serie de bloques de trabajos (bloques constitutivos del camino crítico) y los movimientos de inserción sólo se hacen entre bloques, nunca dentro de un bloque, puesto que se asegura que el C_{max} resultante siempre será igual o mayor. El algoritmo parte de una solución inicial generada a partir de la heurística NEH con las mejoras de Taillard y la aceleración propuesta por bloques de trabajos. El algoritmo resultante es bastante complejo y difícil de implementar, puesto que tiene numerosas variantes y condicionantes. Los autores compararon el mejor de sus algoritmos TS propuestos (referido como TSAB) con el TS de Taillard (1990) y el algoritmo de Werner (1993) (que se verá más

adelante) mostrando el algoritmo TSAB un mejor rendimiento. Este algoritmo ha sido reconocido en algunos trabajos como el mejor algoritmo propuesto hasta la fecha para el taller de flujo (véase Anderson, Glass y Potts, 1997). No obstante, como veremos en el resto de la revisión, han aparecido resultados posteriores que ponen en tela de juicio esta afirmación.

Ben-Daya y Al-Fawzan (1998) propusieron un TS con algunas características adicionales como es la intensificación y la diversificación. También es interesante comentar que este algoritmo no utiliza un único tipo de vecindario, sino que en cada paso aleatoriamente se escoge un vecindario (I , E y otros) que se examina de una manera aleatoria. La solución inicial considerada es la heurística NEH. Los autores han comparado el TS resultante con el TS de Taillard resultando ser ligeramente más ineficaz pero más eficiente y con el SA de Ogbu y Smith al que han superado tanto en términos de calidad de las soluciones como en velocidad. Moccellin y dos Santos (2000) presentaron un híbrido entre TS y SA. El vecindario considerado es el I y examinan un número predeterminado de vecinos de manera aleatoria. La idea general es partir de un algoritmo tipo TS pero una vez se han examinado todos los vecinos, la probabilidad de aceptar el mejor sigue un esquema SA, es decir, si el mejor vecino de los examinados es peor que la solución actual, existe una probabilidad no nula de que sea aceptado. Si se acepta, el movimiento que nos ha llevado hasta él se introducirá en la lista tabú. El resto del algoritmo es muy parecido al algoritmo ya citado de uno de los autores (Moccellin, 1995). Los autores han comparado este método con implementaciones de simples SA y con el algoritmo TS de Moccellin. Como es de esperar, el algoritmo propuesto resulta superar a los otros dos, aunque también con un mayor coste computacional.

3.1.5.3. Algoritmos Genéticos

Los algoritmos genéticos se desarrollaron antes que los métodos SA y TS (Holland, 1975) y comenzaron a aplicarse a problemas de programación de la producción algún tiempo después. En Davis (1985b) aparece una primera aplicación a un problema real de programación de la producción, aunque no para el taller de flujo.

Uno de los primeros trabajos relacionados con algoritmos genéticos y con el problema del taller de flujo se debe a Cleveland y Smith (1989). Realmente, en este trabajo se resuelve el problema de secuenciación de sectores o “*sector scheduling problem*” que es un complejo problema de programación de la producción. De todas formas, los autores, en una primera aproximación, hacen una simplificación y resuelven un problema muy parecido al taller de flujo mediante un algoritmo genético aplicando operadores que se habían utilizado con éxito para resolver el problema del viajante de comercio (ver Grefenstette et al., 1985). Aplican varios operadores para el cruce, entre ellos el operador “*Partially Matched Crossover*” o PMX de Goldberg y Lingle (1985), cuyo funcionamiento veremos en el Capítulo 4. Realmente no se dan más detalles del algoritmo genético ni comparaciones dado que el problema que se resuelve resulta ser muy concreto.

Por tanto, uno de los primeros algoritmos genéticos que de forma general se aplicaron al problema del taller de flujo se debe a Chen, Vempati y Aljaber (1995). Este algoritmo incorpora varias modificaciones con respecto a un GA estándar. Por ejemplo, la población inicial no se genera de manera aleatoria, si no que se crea a partir de tres sencillos pasos:

1. Los $m - 1$ primeros individuos se generan a partir de las $m - 1$ secuencias que proporciona como respuesta la heurística CDS.
2. El individuo m se genera con la heurística RA de Dannenbring.
3. El resto de individuos hasta llegar al tamaño de la población (P_{size}) se generan a partir de simples intercambios de trabajos en los m individuos anteriores.

El algoritmo no aplica el operador de mutación, sólo el operador de cruce. En este caso el cruce utilizado es el anteriormente comentado PMX. Los autores compararon el GA propuesto con el algoritmo SPIRIT de Widmer y Hertz y con la heurística de mejora de Ho y Chang, resultando el GA más eficaz en todas las pruebas consideradas.

Reeves (1995) publicó otro GA para el taller de permutación también con características novedosas. La selección de los padres no es uniforme, dado que el padre se selecciona mediante una función de tipo ranking mientras que la madre

se selecciona con una probabilidad uniforme. El operador de mutación que se considera es de tipo inserción, basado en el vecindario I y el operador de cruce está basado en el cruce de un punto (detalles sobre estos operadores se verán en el Capítulo 4). Las características fundamentales del algoritmo son una mutación adaptativa y el esquema generacional utilizado. Mediante la mutación adaptativa, Reeves controla que la población no converja prematuramente; inicialmente se permite una probabilidad de mutación dada que se va aumentando conforme la diversidad de la población baja (tomando como medida el ratio entre el valor de adecuación del mejor individuo y el valor de adecuación medio de la población), si la diversidad de la población sobrepasa un cierto umbral máximo, entonces se restituye la probabilidad de mutación inicial. El esquema generacional difiere fundamentalmente de los algoritmos genéticos estándar; una vez se han aplicado los operadores de selección, cruce y mutación, los hijos generados no reemplazan directamente a los padres, sino que reemplazan a aquellos individuos de la población que tienen un valor de adecuación inferior a la media de la población. A los algoritmos genéticos que utilizan este tipo de esquemas generacionales (donde los hijos no reemplazan a los padres si no a otros individuos) se les conoce como algoritmos genéticos tipo “*steady state*”. Al igual que en el GA de Chen, Vempati y Aljaber, la inicialización de la población no es completamente aleatoria. En este caso se genera un único “super” individuo mediante la aplicación de la heurística NEH y el resto de individuos sí se generan al azar. El autor comparó este algoritmo genético con el SA de Osman y Potts y con una sencilla búsqueda local. El GA propuesto resultó ser más eficaz. Para la evaluación el autor utilizó dos bancos de pruebas, uno con datos aleatorios propuesto por él mismo y el conjunto de ejemplos de Taillard.

Murata, Ishibuchi y Tanaka (1996a) realizaron un completo estudio comparativo entre los paradigmas SA, TS y un sencillo algoritmo genético propuesto por ellos mismos. El algoritmo genético presentado utiliza un cruce de dos puntos, una mutación de inserción y la técnica elitista, junto con una población inicial aleatoria. En la comparativa el GA resultó ser peor que sencillas implementaciones de simulated annealing, tabu search y una búsqueda local. Debido a esto, los autores propusieron dos nuevos algoritmos resultantes de la hibridación del GA con búsqueda local y con simulated annealing, algoritmos que llamaron “*Genetic*

Local Search" y "*Genetic Simulated Annealing*". La hibridación realizada consiste en llevar a cabo unas pocas iteraciones de búsqueda local o simulated annealing en cada generación y a toda la población antes de comenzar el proceso genético con la selección. De las dos versiones híbridas, el algoritmo "*Genetic Local Search*" resultó ser mejor que el GA simple y las implementaciones de TS y SA. La contribución de este artículo estriba en la idea de hibridizar técnicas metaheurísticas. De hecho la hibridación de algoritmos genéticos con otras técnicas se ha estudiado posteriormente en otros trabajos (ver Reeves y Höhn, 1996). Los autores, en otro trabajo, experimentaron con técnicas multiobjetivo donde aplicaron este mismo algoritmo (ver Murata, Ishibuchi y Tanaka, 1996b).

Reeves y Yamada (1998) (ver también Yamada y Reeves, 1997) propusieron también un algoritmo genético híbrido donde la búsqueda local va incluida en el operador de cruce. Este operador, llamado "*MultiStep Crossover Fusion*" o MSFX realiza una búsqueda local en el vecindario I partiendo desde el primer parente y moviéndose de una manera dirigida hacia el otro parente. El hijo resultante de este cruce es la mejor secuencia encontrada en esta búsqueda local. La mutación también utiliza este tipo de búsqueda local. El esquema generacional utilizado se basa en el GA propuesto anteriormente (Reeves, 1995), pero en este caso el hijo generado no sustituye a un individuo con valor de adecuación por debajo de la media sino al peor individuo de la población. Los autores compararon, utilizando los ejemplos de Taillard, este algoritmo con el TS de Nowicki y Smutnicki, resultando ser este último menos eficaz, sobre todo en los ejemplos de mayor tamaño. Cabe destacar, no obstante, que los tiempos de proceso necesarios para obtener estas buenas soluciones superan en algunos casos las tres horas de cómputo en un superordenador, luego parece que el uso extensivo de la búsqueda local tiene un claro impacto en la eficiencia del algoritmo.

Onwubolu y Mutingi (1999) presentaron un GA, pero esta vez considerando los criterios \bar{T} (o tardanza media), N_T (o número de trabajos retrasados), y una combinación de ambos. El GA propuesto es bastante sencillo y la aportación de este algoritmo se centra en la consideración de los objetivos comentados que no son muy frecuentes en la literatura.

Jain, Bagchi y Wagneur (2000) realizaron un extenso estudio sobre el resultado de la hibridación de un GA (ver también Jain y Bagchi, 2000). A partir del

estudio queda claro que la inicialización no aleatoria de la población tiene un efecto positivo sobre la eficacia del GA, también se observa como la hibridación mediante la aplicación de mejoras a los individuos con diversas heurísticas de mejora resulta en algoritmos genéticos más eficaces.

Ponnambalam, Aravindan y Chandrasekaran (2001) desarrollaron un sencillo algoritmo genético con un operador de cruce llamado GPX (“*Generalized Position Crossover*”), mutación por intercambio y una inicialización de la población aleatoria. Los autores compararon el rendimiento de este algoritmo con algunas de las heurísticas más conocidas (Palmer, Gupta, CDS, RA y NEH), resultando ser, según sus pruebas, superior en todos los casos.

Recientemente, Tang y Liu (2002) han publicado un GA con el criterio de \bar{F} . El algoritmo genético aplica dos operadores no estándar, que son el operador de filtrado o “*filtering*” y el de cultivo o “*cultivation*”. El primer operador se aplica tras cada generación y lo que hace es reemplazar los dos peores individuos de la población con los dos mejores. El segundo operador realmente es una forma de búsqueda local. Simplemente, cuando el mejor individuo de la población ha permanecido sin cambios durante un número determinado de generaciones, se aplica una búsqueda local en el vecindario E restringido (sólo se intercambian trabajos adyacentes). Los autores comparan el algoritmo genético con la heurística de Ho y Chang y con la de Rajendran, resultando el GA mejor.

Consideramos interesante hacer referencia a otros trabajos sobre algoritmos genéticos aplicados a problemas de programación de la producción, aunque no precisamente al taller de flujo. Por ejemplo, Biegel y Davern (1990) desarrollaron un GA para el problema del Job Shop y aplicaron el resultado a un entorno productivo real. Chen, Neppalli y Aljaber (1996) aplicaron un sencillo GA al problema $F/nwt/C_{max}$. Jawahar, Aravindan y Ponnambalam (1998) desarrollaron y aplicaron un GA a un complejo problema de producción en celdas de tipo Job Shop. Estos son sólo algunos ejemplos, ya que la literatura de algoritmos genéticos aplicados a problemas de programación de la producción es muy extensa.

3.1.5.4. Otras técnicas metaheurísticas

Algunos autores han propuesto metaheurísticas que no se basan en ninguno de los paradigmas anteriores. Muchos de estos métodos explotan conocimiento específico del problema y no son fácilmente adaptables a otros entornos o a problemas distintos.

Werner (1993) propuso una serie de algoritmos basados en trayectorias (“*path algorithms*”) para el problema del taller de flujo. Concretamente los algoritmos generan trayectorias dentro de un vecindario representado mediante un tipo especial de grafo. Para generar las trayectorias, el autor examina la estructura de algunas buenas soluciones factibles y examina aquellos vecinos de éstas que cumplen una serie de condiciones. Al igual que el algoritmo TS de Nowicki y Smutnicki, no se examinan vecinos que no vayan a proporcionar soluciones mejores. El algoritmo resultante se compara favorablemente con otras implementaciones de tabu search y simulated annealing, sobre todo en términos de eficiencia.

Stützle (1998) presentó la aplicación de una metaheurística general llamada “*Iterated Local Search*” o ILS al taller de flujo de permutación. Este tipo de metaheurística es relativamente nueva y ha aparecido en pocos ámbitos y dado que no está tan extendida como SA o TS vamos a dar una descripción más extensa. En el Listado 3.2 tenemos el funcionamiento general de esta metaheurística.

```

procedure ILS
begin
  S0=Generar_Solución_Inicial;
  S=Búsqueda_Local(S0);
  while NOT (Condición_Terminación) do
    begin
      S1=Modificar(S);
      S2=Búsqueda_Local(S1);
      S3=Criterio_Aceptación(S,S2)
    end;
  end;

```

Listado 3.2 – Pseudocódigo de la metaheurística general ILS.

Como podemos observar, el método genera una secuencia de partida con la llamada Generar_Solución_Inicial y antes de comenzar el bucle principal, se busca el mínimo local de esta secuencia con respecto a alguna defi-

nición de vecindario con la llamada `Búsqueda_Local`. Entramos en el bucle y no saldremos de él mientras no se cumpla la condición de terminación (`Condición_Terminación`), que puede ser tiempo, número de iteraciones, número de secuencias evaluadas etc... Una vez dentro del bucle la primera llamada `Modificar` nos llevará, mediante una perturbación, a otra solución cercana alternativa (recordemos que la solución previa es un óptimo local). Volviendo a llamar al procedimiento `Búsqueda_Local` conseguiremos que de nuevo esta solución “modificada” sea un óptimo local. La llamada `Criterio_Aceptación` decidirá si este nuevo óptimo local es mejor que el óptimo local anterior. Cada llamada tiene sus propias características, por ejemplo, el procedimiento `Modificar` debe alterar suficientemente la solución actual para permitir que después se encuentre un nuevo óptimo local pero no debe ser muy fuerte para no perturbar demasiado la solución actual, perdiendo características importantes de la misma. Para el caso particular del taller de flujo, Stützle escogió partir de una solución inicial generada con la heurística NEH (con las mejoras de Taillard) y utilizó el vecindario *I* para el procedimiento `Búsqueda_Local`. De igual manera, la llamada `Modificar` simplemente realiza dos inserciones de trabajos (vecindario *I*) y un intercambio (vecindario *E*). Para la función `Criterio_Aceptación` se aplica un criterio muy parecido a los algoritmos SA, pero con una temperatura constante, es decir, existe una probabilidad pequeña, pero no nula de aceptar un nuevo óptimo local con un valor de C_{max} inferior al anterior. Como podemos observar, la metaheurística ILS aplicada al taller de flujo de permutación es muy sencilla y muy fácil de implementar. El autor logró mejorar las soluciones del algoritmo mucho más complejo de Nowicki y Smutnicki para el conjunto de problemas Taillard (1993) e incluso logró encontrar nuevas cotas superiores para el citado conjunto de pruebas. El autor proporcionó las secuencias completas de estas nuevas mejores soluciones en su trabajo.

3.2. Evaluación de heurísticas para el problema del taller de flujo

Como hemos podido ver en secciones anteriores, existe un gran número de métodos, tanto heurísticos de tipo constructivo y de mejora como metaheurísticos, para resolver el problema del taller de flujo. A lo largo de las últimas décadas han ido apareciendo distintos trabajos en la literatura donde se hacen evaluaciones comparativas entre algunos de los algoritmos y métodos anteriormente citados.

En 1970, Ashour realizó una de las primeras evaluaciones comparativas que se conocen para este problema. El autor evaluó principalmente métodos exactos, como los algoritmos de tipo Branch & Bound y la programación entera. Concretamente, la evaluación comprendía un total de cuatro métodos que se probaron con 550 problemas distintos de hasta 12 trabajos y 5 máquinas. Dannenbring (1977) evaluó 11 métodos, incluyendo los tres que él mismo propuso y que se han citado anteriormente utilizando un total de 1580 problemas de hasta 50 trabajos y 50 máquinas. Por otra parte, Byung Park, Pegden y Enscore (1984) evaluaron 16 métodos, todos ellos heurísticos, entre los que incluyeron algunas sencillas modificaciones de algoritmos conocidos. El conjunto de pruebas utilizado estaba formado por 1500 problemas de hasta 30 trabajos y 20 máquinas. Turner y Booth (1987) compararon los métodos NEH y RAES, que en aquel momento se consideraban los más eficaces. En la evaluación utilizaron 350 problemas, de los cuales los de mayor tamaño tenían 25 trabajos y 20 máquinas. Más recientemente, Ponnambalam, Aravindan y Chandrasekaran (2001) evalúan cinco heurísticas con 21 problemas de reducido tamaño. Otros autores hacen algunas evaluaciones, aunque limitadas, cuando proponen sus nuevos métodos, para así validar de alguna manera las prestaciones de los algoritmos presentados. Por ejemplo Widmer y Hertz (1989) evalúan seis métodos y los comparan con el metaheurístico SPIRIT presentado por ellos mismos. Ésta es quizá una excepción, dado que muchos de los trabajos presentados en la Sección 3.1.5 sólo evalúan dos, a lo sumo tres algoritmos para validar los métodos propuestos.

En las anteriores evaluaciones podemos encontrar una serie de deficiencias que pasamos a comentar:

- No se ha utilizado un banco de pruebas estándar en las distintas comparaciones realizadas. Sólo recientemente algunos autores comparan sus métodos utilizando los problemas de Taillard. Hasta el momento no se ha publicado ningún trabajo que realice una evaluación exhaustiva con este banco de pruebas.
- Al no ser los bancos de pruebas estándar, no es fácil extrapolar los resultados para otros conjuntos de problemas con distinto número de trabajos y de máquinas.
- Los bancos de pruebas utilizados son, en muchas ocasiones, de un tamaño muy reducido. No se han publicado evaluaciones comparativas con ejemplos de más de 50 trabajos.
- Si de todas formas pudiésemos comparar los resultados, nos encontraríamos con el problema de que los tiempos de proceso, es decir, la eficiencia de los distintos métodos evaluados, no es comparable. Distintas implementaciones pueden dar lugar a resultados muy dispares. Además, la utilización de plataformas hardware distintas en cada caso imposibilita, en buena medida, la extrapolación de los resultados de eficiencia.
- Los métodos que se evalúan en los trabajos publicados son siempre los mismos. Las heurísticas NEH, CDS y la de Palmer aparecen casi siempre en todos los trabajos. No existen evaluaciones donde se comparen heurísticas menos conocidas y no existe ningún trabajo donde se evalúen las heurísticas constructivas y de mejora más recientes.
- Las evaluaciones no suelen incorporar técnicas metaheurísticas. No hemos encontrado ningún trabajo donde se comparen métodos metaheurísticos de distinto tipo (TS, SA, GA), así como los metaheurísticos más recientes.

Para cubrir las lagunas anteriores presentamos una exhaustiva comparación de métodos heurísticos, tanto los clásicos más conocidos como los menos conocidos que nunca se han evaluado. Adicionalmente, nunca antes se ha hecho un esfuerzo para comparar la eficiencia y eficacia de las distintas propuestas de algoritmos metaheurísticos. Otro aspecto importante es la utilización tanto de un banco de

pruebas estándar como de una plataforma informática única para realizar las pruebas.

En este apartado vamos a realizar una completa evaluación de algoritmos para el problema del taller de flujo, teniendo en cuenta todos los aspectos antes mencionados. La Tabla 3.6 muestra, a modo de resumen y ordenadas por orden cronológico, las distintas heurísticas de tipo constructivo y de mejora para el problema del taller de flujo anteriormente citadas. Asimismo, la Tabla 3.7 muestra un resumen y características principales de los distintos métodos metaheurísticos. Debemos insistir, no obstante, en que existen más métodos de los aquí citados, sin embargo, creemos no haber omitido ningún método con aportaciones relevantes.

Año	Autor/es	Acrónimo	Tipo ¹	Comentarios
1954	Johnson	Johns	C	exacto para $m = 2$
1961	Page	Page	C	algoritmos de ordenación
1964	Dudek y Teuton		C	tipo Johnson
1965	Palmer	Palme	C	índices de pendiente
1970	Campbell, Dudek y Smith	CDS	C	tipo Johnson
1971	Gupta	Gupta	C	índices de pendiente
1972	Gupta		C	tres heurísticas para \bar{F} y C_{max}
1976	Bonney y Gundry		C	alineamiento de pendientes
1977	Dannenbring	RA,RACS, RAES	C/M	tres heurísticas, una constructiva y dos de mejora
1978	Miyazaki, Nishiyama y Hashimoto		C	intercambio de trabajos, criterio \bar{F}
	Gelders y Sambandam		C	basada en reglas de prioridad, cuatro métodos distintos
1980	King y Spachis		C	cinco heurísticas basadas en reglas de despacho
1982	Stinson y Smith		C	basado en viajante de comercio
1983	Nawaz, Encore y Ham	NEH	C	inserción de trabajos
1988	Hundal y Rajgopal	HunRa	C	basada en Palmer
1991	Rajendran y Chaudhuri Ho y Chang Sarin y Lefoka		C	criterio \bar{F}
	Rajendran	HoCha	M	minimización de tiempos entre trabajos
	Kim		C	minimización de tiempo ocioso en la última máquina
1996	Chen et al.		C	criterio \bar{F} . Basado en NEH
1998	Koulamas	Koula	C/M	criterio \bar{T}
	Woo y Yim		C	basada en Johnson, sólo para $F3//C_{max}$
2000	Suliman	Sulim	M	dos fases, 1 ^a basada en Johnson, 2 ^a mejora mediante intercambio de trabajos
2001	Davoud Pour	Pour	C	intercambio de parejas de trabajos
2002	Framinan, Leisten y Ruiz-Usano		C	intercambio de trabajos
2003	Framinan, Leisten y Rajendran		C	bi-objetivo: C_{max} y F_{max}
			C	basado en NEH, objetivos C_{max} , \bar{F} y minimización del tiempo ocioso

Tabla 3.6 – Heurísticas constructivas y de mejora para el problema del taller de flujo.

¹C: heurística de tipo constructivo, M: heurística de mejora.

Año	Autor/es	Acrónimo	Tipo	Comentarios
1989	Osman y Potts Widmer y Hertz	SAOP Spirit	SA TS	vecindario I , solución inicial aleatoria vecindario E , solución inicial basada en el OTSP
1990	Taillard Ogbu y Smith		TS SA	vecindario I , propuestas de paralelización vecindario I , solución inicial de Dannenbring
1993	Werner Reeves		otro TS	algoritmos basados en trayectorias vecindario I , Basado en Osman y Potts
1995	Chen, Vempati y Aljaber Reeves Ishibuchi, Misaki y Tanaka	GACHen GAReev	GA GA SA	cruce PMX mutación adaptativa, “ <i>steady state</i> ” se consideran K soluciones en cada iteración
	Zegordi, Itoh y Enkawa Moccellin		SA TS	incorpora conocimiento del problema basado en SPIRIT, inicialización por OTSP
1996	Nowicki y Smutnicki		TS	vecindario I reducido basado en bloques de trabajos
1998	Murata, Ishibuchi y Tanaka Reeves y Yamada Stützle Ben-Daya y Al-Fawzan	GAMIT ILS	híbrido GA otro TS	GA+Búsqueda local cruce con búsqueda local búsqueda local iterativa intensificación + diversificación, varios vecindarios
1999	Onwubolu y Mutingi		GA	objetivos \bar{T} y N_T .
2000	Moccellin y dos Santos Jain, Bagchi y Wagneur		híbrido híbrido	TS + SA GA+búsqueda local
2001	Ponnambalam, Aravindan y Chandrasekaran	GAPAC	GA	cruce GPX
2002	Tang y Liu Wodecki y Bozejko		híbrido SA	criterio \bar{F} , hibridación con búsqueda local computación paralela

Tabla 3.7 – Metaheurísticas para el problema del taller de flujo.

En las citadas tablas, la columna “Acrónimo” denota el código por el cual nos referiremos a las distintas heurísticas y/o metaheurísticas que se han implementado. Es necesario, no obstante, proporcionar información adicional sobre los detalles de programación y características concretas de las versiones codificadas de los métodos evaluados, dado que en algunos casos la implementación no es sencilla, o con los datos proporcionados por los autores no es posible hacer una

programación completa.

Las heurísticas y metaheurísticas se han programado en el entorno de desarrollo Delphi[©] de la empresa Borland Software Corporation (<http://www.borland.com>). La versión empleada es la 6.0 (ver por ejemplo, Charte Ojeda, 2001b, Cantú, 2001, Charte Ojeda, 2001a y Texeira y Pacheco, 2002). Se ha utilizado una programación procedural para los algoritmos y métodos evaluados y una programación orientada a objetos para el diseño de la interfaz de usuario. Esta elección viene motivada por la mayor velocidad del enfoque procedural. Es importante remarcar que en todo momento se ha hecho uso de una programación “eficiente”. Por ejemplo, muchos de los métodos evaluados requieren en algún momento la ordenación de los trabajos de acuerdo a algún índice (por ejemplo la heurística NEH en el paso 2), con esto hemos implementado una versión muy eficiente del mejor algoritmo de ordenación conocido, el QuickSort (ver Aho, Hopcroft y Ullman, 1983 y Knuth, 1998), eliminando la recursión de cola y haciéndolo con ello todavía más rápido. En otros casos se han utilizado pequeños segmentos de código en ensamblador, sobre todo para las funciones más utilizadas, por ser un lenguaje mucho más rápido (ver Charte Ojeda, 2003). Más información sobre programación eficiente en Delphi puede verse en Bucknall (2001) y en <http://www.optimalcode.com/opguide.htm>. Detalles sobre el software de programación de la producción desarrollado pueden consultarse en el Anexo C.

Para la regla de Johnson hemos utilizado el algoritmo presentado en la Sección 3.1.1 en el caso en que $m = 2$ y la extensión para m máquinas también de la misma sección. Como ya hemos comentado, Page presentó tres métodos heurísticos “merging”, “pairing” y “exchanging”, siendo el primero el menos eficaz y el último el más eficaz. Hemos decidido implementar el método “pairing” tras comprobar que el método “exchanging” resultaba tremadamente exhaustivo y necesitaba cientos de miles de iteraciones cuando $n \geq 75$.

Como ya se ha comentado anteriormente, hemos utilizado la implementación eficiente de Taillard (1990) para la heurística NEH ya que resulta en un algoritmo mucho más rápido. Por poner un ejemplo, la heurística NEH tarda 32.390 milisegundos en obtener una solución para un ejemplo de 500 trabajos y 20 máquinas

en un ordenador con un procesador AMD Athlon 1600+XP (1400 MHz) y 512 Mbytes de memoria RAM, mientras que la heurística NEH mejorada de Taillard necesita tan solo 280 milisegundos para el mismo ejemplo y en el mismo ordenador. Luego la mejora proporciona un algoritmo que es 100 veces más rápido.

Para la heurística de dos fases de Koulamas fue necesario contactar con el autor en varias ocasiones, sobre todo por la existencia de incongruencias en la fase 2 del algoritmo. Luego la fase 2 de este método se ha implementado conforme a las indicaciones del autor y no siguiendo el pseudoalgoritmo que apareció publicado. La heurística de mejora de Suliman también resultó muy complicada de codificar. Se intentó contactar con el autor en varias ocasiones para aclarar algunos aspectos del método pero no se obtuvo respuesta. La heurística se ha implementado tal y como aparece en el artículo aun habiendo algunos aspectos que no quedan claros. En cuanto a las técnicas metaheurísticas no hubo pegas para ninguna de ellas excepto para el método ILS de Stützle. El autor especifica en su trabajo que para la función Criterio_Aceptación se aplica un criterio muy parecido a los algoritmos SA, pero con una temperatura constante. El problema es que el autor no especifica como se calcula esta temperatura. Tras comunicarnos con él, obtuvimos el valor para esta temperatura T , que se calcula como sigue:

$$T = \frac{2}{3} \cdot \left(\frac{\sum_{i=1}^m \sum_{j=1}^n p_{ij}}{n \cdot m} \cdot \frac{1}{50} \right)$$

Para evaluar todos los métodos vamos a utilizar el conjunto de problemas que Taillard propuso en 1993 y que ya hemos citado anteriormente. Este conjunto de pruebas o “*benchmark*” para el taller de flujo de permutación se compone de un total de 120 problemas de varios tamaños. Concretamente, el benchmark se organiza en 12 grupos de 10 problemas cada uno. Dentro de cada grupo, el número de máquinas y de trabajos permanece constante y para todos los problemas tenemos que los tiempos de proceso (p_{ij}) son cantidades extraídas a partir de una distribución aleatoria uniforme $U[1; 99]$. El conjunto de problemas está accesible en <http://mscmga.ms.ic.ac.uk/jeb/>

orlib/floshopinfo.html. o en la propia página web personal de Taillard (<http://ina.eivd.ch/collaborateurs/etd/default.htm>). Habiendo transcurrido una década desde la publicación de este conjunto de problemas y que muchos autores han invertido un esfuerzo considerable en resolver de manera óptima los ejemplos, muchos de ellos todavía continúan abiertos. Concretamente, diversos autores han utilizado técnicas de bifurcación y acotación, así como otras técnicas enumerativas exactas, partiendo de soluciones iniciales de muy buena calidad, así como de cotas inferiores muy ajustadas para intentar resolver los ejemplos más duros. No se detallan el tiempo ni el número de iteraciones o plataforma hardware/software utilizada para la obtención de dichas soluciones. La manera que tienen los autores de determinar cuando un ejemplo de Taillard ha sido “cerrado” (es decir, se conoce su solución óptima) es cuando la máxima cota inferior conocida coincide con la mínima cota superior obtenida.

Es importante remarcar que hemos obtenido las soluciones óptimas y cotas inferiores y superiores no solo de las fuentes citadas, sino de otros múltiples trabajos posteriores donde se han publicado nuevos resultados para el mismo grupo de problemas, por ejemplo de Stützle (1998). No obstante, algunas de estas “nuevas soluciones” no han podido ser contrastadas, al no haber proporcionado los autores las permutaciones de los trabajos que han producido tales nuevas soluciones. A continuación, en la Tabla 3.8 mostramos el conjunto de pruebas de Taillard, junto con las soluciones óptimas, máximas cotas inferiores (CI) y mínimas cotas superiores conocidas (CS).

Problema	<i>n</i>	<i>m</i>	CI	CS	Problema	<i>n</i>	<i>m</i>	CI	CS
ta001	20	5	1278		ta002	20	5	1359	
ta003	20	5	1081		ta004	20	5	1293	
ta005	20	5	1235		ta006	20	5	1195	
ta007	20	5	1234		ta008	20	5	1206	
ta009	20	5	1230		ta010	20	5	1108	
ta011	20	10	1582		ta012	20	10	1659	
ta013	20	10	1496		ta014	20	10	1377	
ta015	20	10	1419		ta016	20	10	1397	
ta017	20	10	1484		ta018	20	10	1538	

Problema	<i>n</i>	<i>m</i>	CI	CS	Problema	<i>n</i>	<i>m</i>	CI	CS
ta019	20	10	1593		ta020	20	10	1591	
ta021	20	20	2297		ta022	20	20	2099	
ta023	20	20	2326		ta024	20	20	2223	
ta025	20	20	2291		ta026	20	20	2226	
ta027	20	20	2273		ta028	20	20	2200	
ta029	20	20	2237		ta030	20	20	2178	
ta031	50	5	2724		ta032	50	5	2834	
ta033	50	5	2621		ta034	50	5	2751	
ta035	50	5	2863		ta036	50	5	2829	
ta037	50	5	2725		ta038	50	5	2683	
ta039	50	5	2552		ta040	50	5	2782	
ta041	50	10	2991		ta042	50	10	2867	
ta043	50	10	2839		ta044	50	10	3063	
ta045	50	10	2976		ta046	50	10	3006	
ta047	50	10	3093		ta048	50	10	3037	
ta049	50	10	2897		ta050	50	10	3065	
ta051	50	20	3771	3855	ta052	50	10	3688	3707
ta053	50	20	3591	3642	ta054	50	10	3635	3726
ta055	50	20	3553	3614	ta056	50	10	3667	3686
ta057	50	20	3672	3705	ta058	50	10	3627	3696
ta059	50	20	3645	3743	ta060	50	10	3696	3767
ta061	100	5	5493		ta062	100	5	5268	
ta063	100	5	5175		ta064	100	5	5014	
ta065	100	5	5250		ta066	100	5	5135	
ta067	100	5	5246		ta068	100	5	5094	
ta069	100	5	5448		ta070	100	5	5322	
ta071	100	10	5770		ta072	100	10	5349	
ta073	100	10	5676		ta074	100	10	5781	
ta075	100	10	5467		ta076	100	10	5303	
ta077	100	10	5595		ta078	100	10	5617	
ta079	100	10	5871		ta080	100	10	5845	
ta081	100	20	6106	6219	ta082	100	20	6183	6204

Problema	<i>n</i>	<i>m</i>	CI	CS	Problema	<i>n</i>	<i>m</i>	CI	CS
ta083	100	20	6252	6271	ta084	100	20	6254	6269
ta085	100	20	6262	6319	ta086	100	20	6302	6368
ta087	100	20	6184	6274	ta088	100	20	6315	6421
ta089	100	20	6204	6275	ta090	100	20	6404	6434
ta091	200	10	10862		ta092	200	10	10480	
ta093	200	10	10922		ta094	200	10	10889	
ta095	200	10	10524		ta096	200	10	10329	
ta097	200	10	10854		ta098	200	10	10730	
ta099	200	10	10438		ta100	200	10	10675	
ta101	200	20	11152	11195	ta102	200	20	11143	11219
ta103	200	20	11281	11337	ta104	200	20	11275	11295
ta105	200	20	11259	11260	ta106	200	20	11176	11189
ta107	200	20	11337	11386	ta108	200	20	11301	11334
ta109	200	20	11145	11192	ta110	200	20	11284	11313
ta111	500	20	26040	26059	ta112	500	20	26500	26520
ta113	500	20	26371		ta114	500	20	26456	
ta115	500	20	26334		ta116	500	20	26469	26477
ta117	500	20	26389		ta118	500	20	26560	
ta119	500	20	26005		ta120	500	20	26457	

Tabla 3.8 – Número de trabajos (*n*), máquinas (*m*), máximas cotas inferiores (CI) y mínimas cotas superiores (CS) conocidas para el “benchmark” de Taillard.

Como podemos observar, un total de 33 problemas siguen abiertos, aunque en muchos casos la diferencia entre las máximas cotas inferiores y mínimas cotas superiores conocidas es muy pequeña.

El formato de los ficheros de datos para los problemas de ejemplo es muy sencillo, se trata de ficheros de texto donde en la primera línea tenemos el número de trabajos *n* y el número de máquinas *m*. Despues hay un total de *n* líneas que

contienen los tiempos de proceso (p_{ij}) y que tienen el siguiente formato:

$$\begin{array}{ccccccccc} 0 & p_{11} & 1 & p_{21} & \cdots & m-1 & p_{m1} \\ 0 & p_{12} & 1 & p_{22} & \cdots & m-1 & p_{m2} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & p_{1n} & 1 & p_{2n} & \cdots & m-1 & p_{mn} \end{array}$$

Un ejemplo puede ser el problema ta001, de 20 trabajos y 5 máquinas, que se muestra en el Listado 3.3.

20	5
0	54
0	83
0	15
0	71
0	77
0	36
0	53
0	38
0	27
0	87
0	76
0	91
0	14
0	29
0	12
0	77
0	32
0	87
0	68
0	94
1	79
1	3
1	11
1	99
2	15
2	49
2	89
2	23
2	57
2	64
2	1
2	63
2	41
2	63
2	47
2	21
2	75
2	77
3	66
3	58
3	31
3	68
3	78
3	91
3	13
3	59
3	49
3	85
3	85
3	9
3	39
3	41
3	56
3	40
3	54
3	77
3	51
4	58
4	56
4	20
4	85
4	53
4	35
4	53
4	41
4	69
4	13
4	86
4	72
4	8
4	49
4	47
4	87
4	58
4	18
4	68
4	28

Listado 3.3 – Problema ta001 de Taillard.

Para evaluar la bondad de cada uno de los métodos implementados, vamos a definir una medida de eficacia, que es el incremento porcentual por encima del óptimo o mínima cota superior conocida y al que denominaremos *IPSO* y que se expresa como:

$$IPSO = \frac{Heu_{sol} - Mejor_{sol}}{Mejor_{sol}} \cdot 100$$

Donde $Mejor_{sol}$ es la solución óptima para un ejemplo dado (si es que ésta se conoce) o la mínima cota superior conocida de acuerdo con la Tabla 3.8 y Heu_{sol} es la solución obtenida por una heurística o método concreto. De esta manera si

obtenemos un $IPSO = 0\%$ tendremos que $Heu_{sol} = Mejor_{sol}$, es decir, el método o heurística Heu ha sido capaz de obtener la solución óptima para el ejemplo en cuestión. En cambio, si $IPSO = 10\%$ tendremos que el método Heu ha proporcionado una solución que es un 10% peor que la mejor solución conocida. Normalmente, haremos referencia a un $IPSO$ medio sobre un grupo de problemas donde el número de trabajos y máquinas es constante. En este caso hablaremos del $IPSOM$. Por ejemplo, podríamos decir que un método Heu tiene un $IPSOM = 5,21\%$ para el grupo de problemas ta001-ta010, es decir, para el grupo de problemas donde $n = 20$ y $m = 5$ (10 problemas) el método Heu ha obtenido, de media, un incremento sobre las mejores soluciones conocidas del 5,21 %. También nos referiremos con frecuencia al $IPSOM$ total, o lo que es lo mismo, a la media del $IPSO$ para los 120 problemas de Taillard. En general nos interesará un $IPSOM$ total bajo para los métodos, dado que buscamos algoritmos que se comporten bien para todos los grupos de problemas, es decir, que tengan un comportamiento robusto independientemente del tamaño de la entrada, del número de trabajos o de máquinas.

Para las evaluaciones de las distintas heurísticas hemos utilizado un ordenador tipo PC/AT con un procesador AMD Athlon 1600+XP (1400 MHz) y 512 Mbytes de memoria RAM. Es importante remarcar que para los métodos metaheurísticos es necesario establecer un criterio de parada. Para poder hacer una evaluación en igualdad de condiciones, es importante la elección de este criterio. Lo más general es establecer el número de evaluaciones de la función objetivo. Es decir, el número de secuencias y C_{max} evaluados. De esta manera nos aseguramos que todos los métodos han podido evaluar el mismo número de soluciones antes de parar. Adicionalmente, es común en la literatura comparar los métodos metaheurísticos con una búsqueda aleatoria. Hemos implementado una sencilla regla, a la que llamamos RAND que genera y evalúa un número dado de secuencias obtenidas de una manera totalmente aleatoria, devolviendo la mejor de todas. Este método se puede considerar como el peor caso posible desde el punto de vista de los métodos con criterio de parada, es de esperar que cualquier otro método funcione mejor. Hemos fijado la evaluación de 50.000 C_{max} como criterio de parada. Este número viene motivado simplemente por el hecho de ser una cantidad que permite ejecutar

los algoritmos en un tiempo razonable y a la vez es suficientemente elevada como para asegurar una convergencia de los métodos en cada uno de los problemas de ejemplo considerados.

También hemos decidido incorporar tres sencillas reglas de prioridad en la comparación, simplemente para tener un valor de referencia de gran interés en la resolución de problemas reales. Dado que los problemas de Taillard carecen de tiempos de finalización o “*deadlines*”, nos hemos limitado a incluir en la comparativa las reglas FCFS, SPT y LPT, por ser tres sencillas reglas que sólo necesitan información sobre los tiempos de proceso (p_{ij}). Además, algunos autores han encontrado que estas reglas, en especial la regla SPT superan a otras mucho más elaboradas en diversos problemas de programación de la producción (ver Hunsucker y Shah, 1994).

Vamos ahora a mostrar los resultados de la evaluación para las técnicas heurísticas constructivas y de mejora. La Tabla 3.9 muestra los *IPSUM*, agrupados por tamaño de problema para las heurísticas consideradas, con 10 ejemplos por grupo. Dado que la regla RAND proporciona un resultado distinto cada vez que se ejecuta, hemos realizado un total de cinco ejecuciones y obtenido la media, por lo que los resultados realmente muestran, para cada grupo de problemas, el *IPSUM* para cinco ejecuciones de 10 problemas cada una (una media con 50 datos).

Problema	RAND	FCFS	SPT	LPT	Johns	Page
20x5	5,79	24,98	20,44	47,66	12,78	15,15
20x10	9,47	28,77	24,74	46,38	19,97	20,43
20x20	7,76	21,43	23,17	34,27	16,47	16,18
50x5	4,74	15,32	16,21	48,33	10,43	10,14
50x10	14,00	25,05	26,51	51,23	21,90	20,47
50x20	16,31	29,59	29,38	48,08	22,81	23,12
100x5	3,70	13,63	17,98	45,19	6,84	7,98
100x10	10,31	20,92	24,63	47,27	15,01	15,79
100x20	16,28	25,36	28,93	47,01	21,15	21,68
200x10	8,39	15,67	20,25	45,54	11,47	12,74
200x20	15,32	22,10	25,94	51,10	18,93	19,43
500x20	11,45	15,99	23,04	48,01	14,07	14,05
Media	10,29	21,57	23,44	46,67	15,99	16,43

- (a) Regla aleatoria RAND, reglas de prioridad FCFS, SPT y LPT y algoritmos de Johnson y Page.

Problema	Palme	CDS	Gupta	RA	RACS	RAES
20x5	10,58	9,54	12,45	8,86	7,71	4,95
20x10	15,28	12,13	24,48	15,40	10,66	8,62
20x20	16,34	9,64	22,53	16,35	8,16	6,41
50x5	5,34	6,10	12,43	6,30	5,40	3,28
50x10	14,01	12,98	22,05	15,05	12,19	10,41
50x20	15,99	13,85	23,66	18,88	12,86	10,00
100x5	2,38	5,01	6,02	3,54	4,58	3,25
100x10	9,20	9,15	15,12	10,48	8,72	7,31
100x20	14,41	13,12	22,68	16,96	12,48	10,56
200x10	5,13	7,38	11,80	6,17	7,11	6,16
200x20	13,17	12,08	19,67	12,67	11,83	10,39
500x20	7,09	8,55	13,66	8,34	8,38	7,77
Media	10,74	9,96	17,21	11,58	9,17	7,43

(b) Algoritmos de Palmer, Campbell, Dudek y Smith (CDS) , Gupta y las tres heurísticas de Dannenbring.

Problema	NEH	HunRa	HoCha	Koula	Sulim	Pour
20x5	3,35	9,35	6,94	7,68	4,46	12,05
20x10	5,02	13,34	10,51	11,82	7,84	12,34
20x20	3,73	13,47	8,30	11,89	6,69	10,71
50x5	0,84	4,21	3,33	4,03	2,20	6,58
50x10	5,12	13,35	11,29	12,13	8,46	14,44
50x20	6,20	14,99	12,40	14,93	9,62	14,87
100x5	0,46	1,99	2,70	3,12	1,28	4,06
100x10	2,13	8,12	7,96	7,50	5,75	7,82
100x20	5,11	13,65	11,10	14,04	9,28	13,18
200x10	1,43	4,50	5,11	5,09	4,09	5,52
200x20	4,37	12,59	9,99	11,60	8,85	11,50
500x20	2,24	6,75	7,14	6,82	6,06	7,69
Media	3,33	9,69	8,06	9,22	6,21	10,07

(c) Algoritmos de Nawaz, Enscore y Ham (NEH), Hundal y Rajgopal, Ho y Chang, Koulamas, Suliman y Davoud Pour.

Tabla 3.9 – Incremento porcentual medio por encima del óptimo o mínima cota superior conocida (*IPSONM*) de los métodos heurísticos constructivos y de mejora para los problemas de Taillard.

Pasamos a comentar los resultados de las 18 heurísticas consideradas anteriormente. El primer resultado era esperado, las reglas de prioridad FCFS, SPT y LPT han demostrado tener un rendimiento muy pobre, son con diferencia los tres peores métodos de la comparación con unos valores totales de *IPSONM* de 21,57 %, 23,47 % y 46,67 % respectivamente. El peor caso es la regla LPT, que en algunos grupos de problemas, como por ejemplo en las 10 instancias de 50 trabajos y 10 máquinas, o en las de 200 trabajos y 20 máquinas, el *IPSONM* pasa del 51 %. También sorprende que la regla FCFS resulta mejor (aunque marginalmente) que la regla SPT. Este resultado de las reglas de prioridad es muy revelador, porque como es bien sabido, las reglas de prioridad son muy utilizadas en la práctica como método estándar de programación de la producción y como vemos, su eficacia deja

mucho que desear. Además, como se puede observar en las tablas de resultados, el uso de la simple regla RAND ofrece unos resultados mucho mejores. Es por esto que podríamos recomendar el uso de una regla aleatoria (repetida múltiples veces) por encima de una regla de despacho, al menos para el problema del taller de flujo.

Entrando en métodos específicamente propuestos para el problema que nos ocupa, tenemos que los dos métodos más antiguos de entre los implementados, los algoritmos Johns y Page tienen un comportamiento regular. Aunque cabe destacar que el algoritmo Johns sea más eficaz de media que el algoritmo Page, aunque por poco, dado que este último es bastante más complejo. Es necesario recordar que la versión del algoritmo de Johnson que se está utilizando es una modificación para tener en cuenta más de dos máquinas. No obstante, como vemos en las tablas, ambos métodos resultan ser considerablemente más eficaces que las reglas de despacho.

La heurística Palme es más eficaz y consigue un *IPSUM* total de 10,74 %, lo cual se puede considerar un buen resultado, teniendo en cuenta su antigüedad. La heurística CDS logra bajar del 10 % siendo mejor que el método Palme, este resultado coincide con otras evaluaciones anteriormente publicadas. El resultado de la heurística Gupta es de un 17,20 %, lo cual contrasta con los resultados del autor en los que se mostró que este método resultaba ser mejor que la heurística Palme. Nuestras pruebas indican claramente que la heurística Gupta es bastante peor que el método Palme, y este resultado se mantiene para todos los grupos de problemas considerados.

Las tres heurísticas de Dannenbring ofrecen un buen rendimiento, sobre todo el método RA, teniendo en cuenta su sencillez. Como era de esperar, la heurística de mejora RACS es mejor que RA pero peor que RAES.

Han transcurrido ya 20 años desde que se publicara el método NEH. Nunca hasta ahora se había contrastado este algoritmo con un conjunto de datos del tamaño de los problemas de Taillard. Como vemos en las tablas, el método NEH es, con diferencia, el método más eficaz de los que se han implementado, y como veremos, con las mejoras que Taillard introdujo en el 1990, es también uno de los más eficientes. Luego de nuevo se confirman los resultados obtenidos en anteriores publicaciones donde se concluye que el método NEH es el mejor. Concretamente,

para el banco de pruebas utilizado, el *IPSOM* total es de 3,33 %.

Otro método que se puede calificar de bueno es la heurística HunRa, pese a tratarse de una sencilla modificación de la regla Palme, consigue superar a ésta y a la más complicada CDS. Esto es muy interesante, dado que la heurística CDS evalúa $m - 1$ secuencias y la HunRa tan sólo tres. La heurística HoCha supera escasamente el 8 % de *IPSOM* total, lo que la sitúa como tercera heurística de la comparativa en términos de eficacia. No obstante, hemos de tener en cuenta que los requerimientos de memoria de este método hacen que no sea adecuado para problemas muy grandes.

En cuanto a las heurísticas más recientes, los resultados, contrariamente a lo que se podría esperar, no son especialmente buenos. A excepción del método Sulim, que consigue poco más de un 6 % de *IPSOM* total, las heurísticas Koula y Pour tienen un rendimiento similar al método CDS, que es mucho más simple y antiguo. Concretamente, la heurística Pour evalúa $[n(n+1)/2] - 1$ secuencias, un número muy superior a las tres secuencias que evalúa el método HunRa, que además resulta ser ligeramente mejor. De alguna manera esperábamos un rendimiento superior del método Koula, sobre todo por su complejidad y por el hecho de que permite generar secuencias generales y no de permutación. Adicionalmente, el author del método Pour concluyó en su artículo que este método es mejor a la heurística NEH, algo que no se cumple, y por una diferencia considerable, en las pruebas realizadas.

Hasta ahora nos hemos limitado a comparar las heurísticas desde el punto de vista de la eficacia, concretamente con la medida de eficacia del *IPSOM* total. Realmente esto sólo nos da una visión parcial de cada método al no estar estudiando la *eficiencia* de cada algoritmo. Por ejemplo, las heurísticas HunRa y Koula están muy igualadas en términos de eficacia, siendo la primera mejor que la segunda para algunos grupos de problemas pero peor para otros. Solo evaluando también la eficiencia podremos discernir entre qué métodos son realmente los mejores. Hemos medido el tiempo (en segundos) que cada algoritmo ha necesitado para resolver el grupo de problemas de Taillard. Los resultados se recogen en la Tabla 3.10. La tabla se ha simplificado puesto que los métodos RACS, FCFS, SPT, LPT, Johns, Page, Palme, CDS, Gupta, RA, NEH y HunRa han

necesitado menos de 1000 milisegundos para resolver cada ejemplo, y por tanto menos de 1 segundo en general, independientemente del tamaño de la instancia. Luego podemos decir, sin temor a equivocarnos, que todos estos métodos son muy eficientes. Cabe destacar el método NEH, que además de ser el más eficaz de la comparativa necesita menos de medio segundo para obtener una solución, incluso para los grupos de problemas más grandes con 500 trabajos. Luego hemos dejado en la tabla aquellos métodos y grupos de problemas para los cuales el tiempo de cómputo superaba un segundo de media.

Problema	RAND	RAES	HoCha	Koula	Sulim	Pour
100x10	1,08	<1	<1	<1	<1	1,54
100x20	2,02	<1	<1	<1	<1	2,92
200x10	2,26	<1	<1	1,04	<1	13,81
200x20	4,50	<1	<1	4,52	<1	26,00
500x20	22,38	4,67	1,18	64,55	13,61	532,61
Media	2,95	<1	<1	5,93	1,20	48,20

RAND=Regla aleatoria, RAES=Algoritmo "Rapid Access with Extensive Search" de Dannenbring (1977), HoCha=Algoritmo de Ho y Chang (1991), Koula=Algoritmo de Koulamas (1998), Sulim=Algoritmo de Suliman (2000), Pour=Algoritmo de Davoud Pour (2001)

Tabla 3.10 – Tiempos de proceso (en segundos) utilizados por los métodos heurísticos constructivos y de mejora para los problemas de Taillard.

Como podemos observar, la heurística RAES, que se basa en el método RACS es muy eficiente, dado que necesita algo menos de 5 segundos para los problemas más grandes de 500 trabajos. El método HoCha también resulta ser muy eficiente, pero es necesario concretar que la plataforma hardware utilizada dispone de una gran cantidad de memoria RAM (512 Mbytes). Pudimos comprobar como el método necesitaba más de 10 segundos para el grupo de problemas más grande con una cantidad de memoria menor (384 Mbytes). Este método aborta para los problemas por encima de 100 trabajos y 20 máquinas en ordenadores con 64 Mbytes de memoria RAM. De nuevo, los tres métodos más recientes no resultan ser especialmente eficientes. El algoritmo Koula necesita casi un minuto para

los problemas de 500 trabajos mientras que el método Sulim algo menos de 15 segundos. El resultado más destacado de esta tabla es el método Pour dado que no sólo tiene una eficacia inferior a métodos más simples y antiguos, sino que la eficiencia es muy baja, la más baja de toda la evaluación con una diferencia considerable. Podemos observar como los tiempos de proceso se disparan conforme aumenta el número de trabajos y de máquinas, hasta llegar a casi nueve minutos de media para los problemas grandes.

Como resumen para las técnicas heurísticas constructivas y de mejora, podemos decir que las reglas de prioridad, aunque muy rápidas y generales, proporcionan unos resultados muy pobres en comparación con algunos métodos específicos para el taller de flujo que no tienen por qué ser complejos en cuanto a su comprensión y codificación. De estos métodos el que es claramente superior, tanto en términos de eficiencia como de eficacia, es la heurística NEH, que resulta ser un 86 % más eficaz que el segundo mejor método al tiempo que es muy eficiente, necesitando menos de medio segundo de tiempo de cómputo, incluso para los problemas más grandes. La evaluación comparativa también nos ha reportado resultados interesantes, como por ejemplo el método Gupta, que es reconocido como superior al método Palme en varios trabajos y por el propio autor y que en nuestro estudio presenta un comportamiento peor para todos los problemas aquí considerados. También hemos podido ver como las tres heurísticas más recientes, Koula, Sulim y Pour no son especialmente eficaces (a excepción del método Sulim) pero sí muy complejas de codificar y con ello también bastante inefficientes, especialmente el método Pour que necesita del orden de minutos para obtener una solución en los problemas grandes.

Una vez hemos comparado las técnicas heurísticas, tanto constructivas como de mejora, vamos a proceder de igual manera con los algoritmos metaheurísticos de la Tabla 3.7. Tenemos que recordar que el criterio de parada para todos estos métodos es la evaluación de 50.000 secuencias (“*makespans*”). También es importante comentar que, dado que es posible que se den situaciones en las que alguno de estos métodos puede haberse quedado estancado en un óptimo local para cuando se alcance este número de evaluaciones, hemos realizado un total

de 5 ejecuciones de cada algoritmo, tal y como hicimos anteriormente con el método RAND. De todos los métodos metaheurísticos, el único determinista es el tabu search **Spirit**, aunque la inicialización de este método (resolución de un OTSP mediante una heurística de inserción) proporciona soluciones iniciales distintas en las ejecuciones, por lo que el método completo tampoco dará la misma solución en todas las ocasiones. Los resultados se muestran en la Tabla 3.11.

Problema	SAOP	Spirit	GAChen	GAReev	GAMIT	ILS	GAPAC
20x5	1,39	5,22	3,82	0,70	4,21	0,30	8,98
20x10	2,66	5,86	4,89	1,92	5,40	0,94	13,61
20x20	2,31	4,58	4,17	1,53	4,53	0,87	11,03
50x5	0,69	2,03	2,09	0,26	3,11	0,13	6,50
50x10	4,25	5,88	6,60	2,58	8,38	2,16	16,41
50x20	5,13	7,21	8,03	3,76	10,65	3,45	18,56
100x5	0,40	1,06	1,32	0,18	5,41	0,11	5,32
100x10	1,88	5,07	3,75	1,08	12,05	0,71	12,34
100x20	5,21	10,15	7,94	3,94	18,24	3,30	18,25
200x10	1,56	9,03	2,70	0,82	7,52	0,45	9,75
200x20	4,83	16,17	7,07	3,33	15,35	2,72	17,06
500x20	3,40	13,57	4,61	1,83	12,17	1,30	12,61
Media	2,81	7,15	4,75	1,83	8,92	1,37	12,53

SAOP=Simulated annealing de Osman y Potts (1989), Spirit=Tabu search de Widmer y Hertz (1989), GAChen=GA de Chen, Vempati y Aljaber (1995), GAReev=GA de Reeves (1995), GAMIT=GA de Murata, Ishibuchi y Tanaka (1996a), ILS=Búsqueda local iterativa de Stützle (1998), GAPAC=GA de Ponnambalam, Aravindan y Chandrasekaran (2001)

Tabla 3.11 – Incremento porcentual medio por encima del óptimo o mínima cota superior conocida (*IPSON*) de los métodos metaheurísticos para los problemas de Taillard.

Podemos comenzar con el simulated annealing SAOP, que consigue obtener un *IPSON* total de 2,81 %, lo que resulta ser mejor que la heurística NEH en todos los casos menos en los problemas de mayor tamaño. En este caso es muy probable que el método SAOP no haya convergido totalmente y necesite más de 50.000 evaluaciones. Consideramos que este es un buen resultado, dado que el simulated annealing SAOP, como hemos podido ver en la Sección 3.1.5.1, es muy sencillo de implementar y además parte de una solución completamente aleatoria. El tabu search **Spirit** resulta ser considerablemente menos eficaz que el método

SAOP, esta diferencia se hace incluso más evidente para los problemas de tamaño superior. El algoritmo genético GACHen resulta ser menos eficaz que la heurística NEH, lo cual no se puede considerar como un buen resultado, sobre todo teniendo en cuenta que el algoritmo genético es más lento y mucho más complejo. La situación es mucho mejor con respecto al algoritmo genético GAReev, donde el *IPSUM* total baja del 2 % hasta un 1,83 %. En cambio, el algoritmo genético híbridizado con búsqueda local GAMIT resulta ser bastante ineficiente, dado que proporciona un *IPSUM* total de casi un 9 %, un resultado claramente superado por otros métodos heurísticos constructivos y de mejora mucho más sencillos que un algoritmo de este tipo. La búsqueda local iterativa ILS es el método más eficaz de toda la comparación realizada, con un *IPSUM* total de 1,37 %. El algoritmo genético GAPAC es el más ineficaz de los métodos metaheurísticos. Es interesante observar que la regla aleatoria RAND anteriormente evaluada es incluso más eficaz. Este es el peor resultado del análisis realizado, el hecho de que un algoritmo genético ofrezca una eficacia inferior a la regla aleatoria es indicativo de algún grave error de diseño, dado que el algoritmo es incapaz de converger más allá de la solución inicial. Adicionalmente, estas cifras contrastan, de manera evidente con los resultados proporcionados por los autores de este algoritmo genético, dado que en su artículo indican que este algoritmo genético es más eficaz que la heurística NEH. Hemos comprobado que las tablas que se proporcionan en el citado artículo son incorrectas, los resultados que se dan para la heurística NEH para los problemas de Carlier (<http://mscmga.ms.ic.ac.uk/jeb/orlib/flowshopinfo.html>) son muy superiores a los que la correcta implementación del método NEH proporciona.

A continuación analizaremos más detenidamente la eficacia de los dos mejores métodos que se han codificado, el algoritmo genético GAReev y la búsqueda local iterativa ILS. Para ello vamos a comparar, problema por problema, el número de veces que un método ha proporcionado una solución mejor, igual, o peor que el otro método. Los resultados se muestran en la Tabla 3.12.

Problema	Número de veces solución mejor por ILS	Número de veces solución igual entre ILS y GAReev	Número de veces solución mejor por GAReev
20x5	7	2	1
20x10	10	0	0
20x20	9	0	1
50x5	9	0	1
50x10	7	1	2
50x20	9	0	1
100x5	7	0	3
100x10	9	1	0
100x20	10	0	0
200x10	9	0	1
200x20	10	0	0
500x20	10	0	0
Total	106	4	10

ILS=Búsqueda local iterativa de Stützle (1998), GAReev=GA de Reeves (1995)

Tabla 3.12 – Comparación entre los métodos metaheurísticos ILS y GAReev para los problemas de Taillard.

Podemos ver como el método ILS es mejor que el algoritmo genético GAReev en 106 de los 120 casos, mientras que el método GAReev tan solo consigue superar al algoritmo ILS en 10 ocasiones. El número de veces en que ambos métodos empatan es muy pequeño. Luego se pone de manifiesto que el método ILS resulta ser el mejor método metaheurístico de la evaluación.

Como ya hicimos con los métodos heurísticos constructivos y de mejora, vamos ahora a comparar los tiempos de proceso para los algoritmos metaheurísticos anteriormente evaluados. Los resultados se muestran en la Tabla 3.13.

Problema	SAOP	Spirit	GAChen	GAReev	GAMIT	ILS	GAPAC
20x5	<1	<1	<1	1,08	<1	4,01	<1
20x10	<1	<1	<1	1,17	<1	4,09	<1
20x20	<1	<1	<1	1,35	<1	4,63	<1
50x5	<1	<1	<1	1,82	<1	6,38	<1
50x10	<1	<1	<1	2,08	<1	9,94	<1
50x20	1,04	<1	1,45	2,53	<1	11,82	1,28
100x5	<1	<1	1,79	3,97	<1	15,31	1,07
100x10	1,10	1,03	2,26	4,49	1,02	18,79	1,57
100x20	2,09	2,00	3,24	5,54	1,99	24,04	2,55
200x10	2,29	2,25	5,97	12,91	2,20	33,73	3,24
200x20	4,59	4,51	8,18	15,16	4,50	41,80	5,46
500x20	39,48	39,70	55,30	101,71	37,82	192,03	30,62
Media	4,42	4,38	6,77	12,82	4,21	30,55	4,04

SAOP=Simulated annealing de Osman y Potts (1989), Spirit=Tabu search de Widmer y Hertz (1989), GAChen=GA de Chen, Vempati y Aljaber (1995), GAReev=GA de Reeves (1995), GAMIT=GA de Murata, Ishibuchi y Tanaka (1996a), ILS=Búsqueda local iterativa de Stützle (1998), GAPAC=GA de Ponnambalam, Aravindan y Chandrasekaran (2001)

Tabla 3.13 – Tiempos de proceso (en segundos) utilizados por los métodos metaheurísticos para los problemas de Tai-llard.

A primera vista podemos ver que las metaheurísticas tienen, como es de esperar, unos requerimientos computacionales más elevados que los métodos heurísticos. No obstante, todas las metaheurísticas tienen un comportamiento bueno, necesitando menos de un minuto de media para obtener las soluciones de todos los problemas excepto para los más grandes. Ya hemos visto que el algoritmo SAOP es más eficaz que el algoritmo Spirit, ahora vemos que ambos algoritmos son igualmente eficientes, necesitando de media casi los mismos tiempos para resolver los problemas. De los algoritmos genéticos, el método GAPAC es el más eficiente, seguido de los algoritmos GAMIT, GAChen y GAReev. El algoritmo más costoso es el ILS que en el caso de los problemas de 500x20 necesita algo más de tres minutos para terminar. Como podemos observar, una mayor eficacia va unida, en este caso con una menor eficiencia.

Esta última afirmación se puede observar de una manera muy intuitiva representando los *IPSOM* totales de los métodos heurísticos frente a los tiempos

de proceso necesarios. La Figura 3.12 muestra este estudio en un diagrama de dispersión.

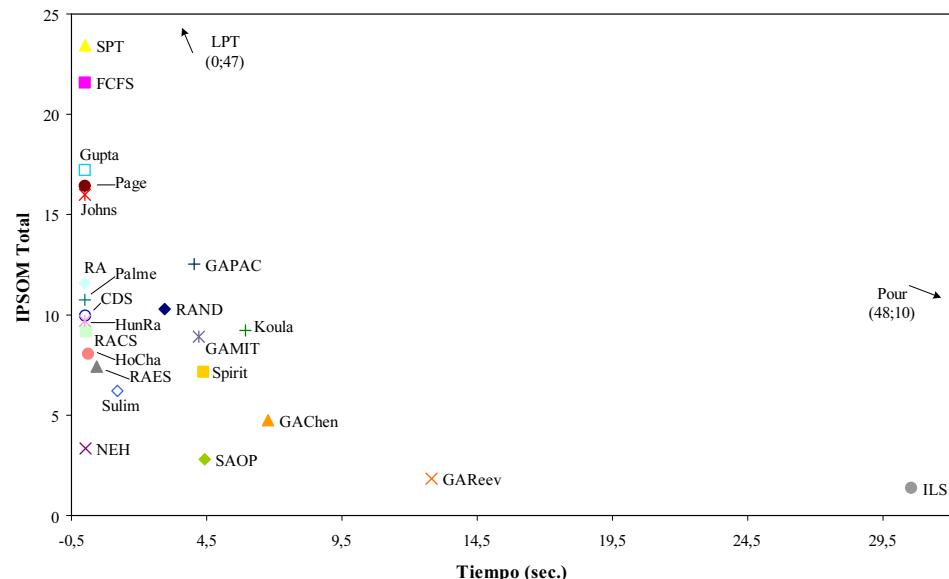
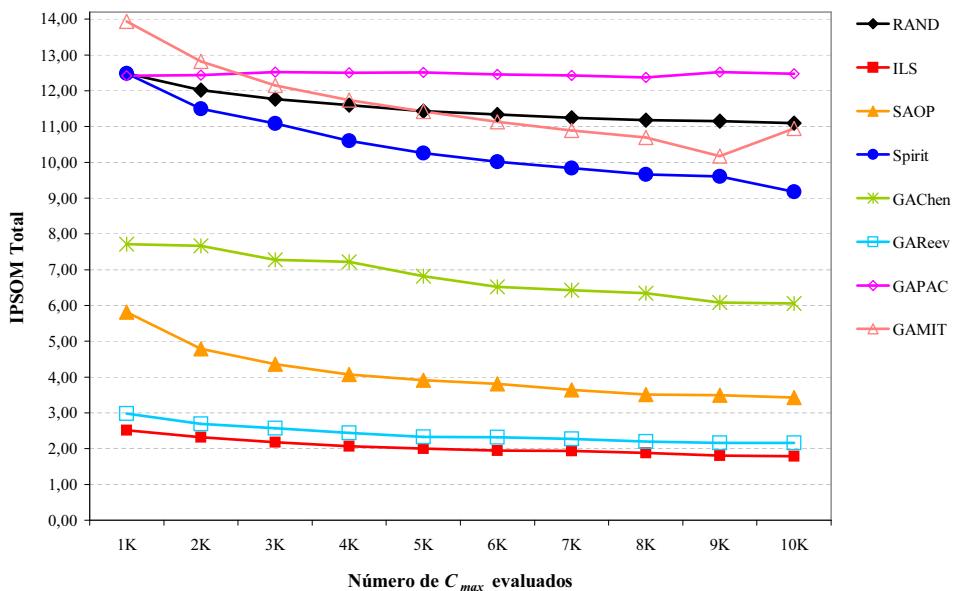


Figura 3.12 – Diagrama de dispersión del incremento porcentual total por encima del óptimo o mínima cota superior conocida (*IPSMOM*) frente al tiempo de proceso para todos los métodos evaluados (criterio de parada 50.000 “*makespans*”).

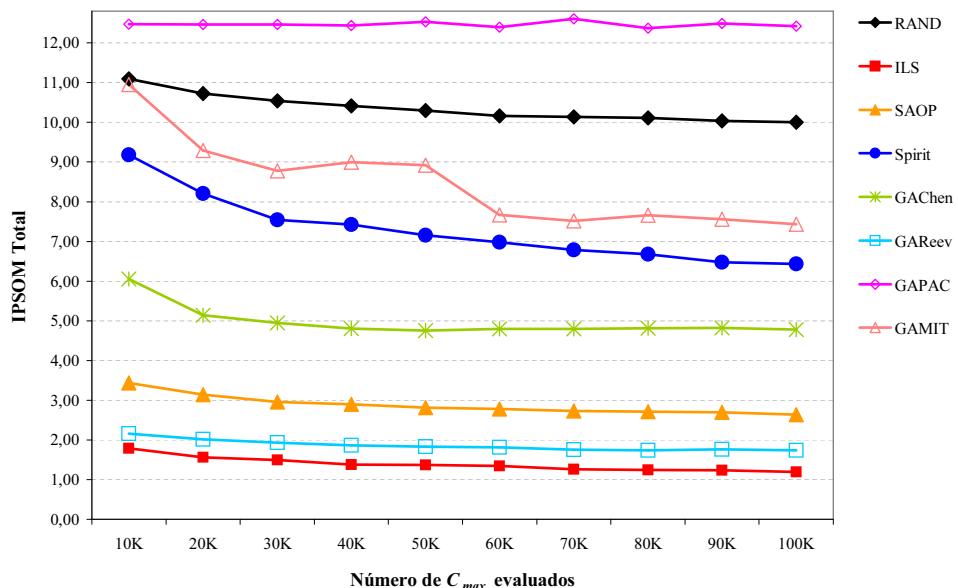
Como vemos, existe una correlación inversa entre las variables *IPSMOM* total y tiempo, aunque existen claras excepciones como son la regla LPT y el método Pour.

Podría argumentarse que los resultados obtenidos con 50.000 evaluaciones del C_{max} no son extrapolables en el caso en el que se permitan un mayor o menor número de éstas. Para poder comprobar este extremo hemos realizado dos experimentos, en el primero se hacen 5 ejecuciones de los 7 métodos metaheurísticos anteriores variando el criterio de parada desde 1.000 hasta 10.000 “*makespans*” a incrementos de 1.000. En el segundo experimento se evalúan los mismos métodos

pero variando el número de C_{max} evaluados desde 10.000 hasta 100.000 en incrementos de 10.000. Para mantener una referencia hemos incluido en estos experimentos la regla aleatoria RAND. Los resultados se muestran en la Figura 3.13.



(a) Desde 1.000 hasta 10.000 “makespans”.



(b) Desde 10.000 hasta 100.000 “makespans”.

Figura 3.13 – Evolución del incremento porcentual total por encima del óptimo o mínima cota superior conocida (*IPSON*) frente al número de C_{max} evaluados para los métodos metaheurísticos.

A partir de este estudio resulta interesante ver cómo para el método GAPAC da igual el número de evaluaciones del C_{max} que permitamos, el método funciona igual para 1.000 C_{max} que para 100.000. Esto corrobora nuestra hipótesis de que existe un grave error de diseño en el algoritmo. De igual manera, el algoritmo genético GAMIT tiene un comportamiento mediocre (incluso peor que RAND) hasta 4.000 evaluaciones del C_{max} , aunque mejora conforme se permiten más evaluaciones. En este mismo caso está el método Spirit, vemos como la solución con 1.000 C_{max} es igual que el método RAND pero su eficacia no para de mejorar conforme se permiten más y más evaluaciones. De hecho, si permitimos un número de evaluaciones muy alto este método acaba siendo comparable al método SAOP en términos de eficacia. Por ejemplo, para un total de 1.000.000 evaluaciones de C_{max} , el método Spirit obtiene un *IPSON* total de 4,8 %,

que es lo que obtiene el método SAOP para 5.000 evaluaciones. Este resultado coincide con el experimento realizado por Reeves (1993), quien llegó a las mismas conclusiones. Podemos decir que el conjunto de métodos más eficaces (GAChen, SAOP, GAReev e ILS) prácticamente se estancan al pasar de las 40.000 o 50.000 evaluaciones de C_{max} aunque para los dos últimos se siguen obteniendo mejoras marginales en el *IPSOM* total, que no justifican el enorme incremento en los tiempos de proceso.

Hasta ahora hemos agrupado los resultados de todos los métodos para todos los grupos de problemas. Por último, vamos a representar de manera gráfica los *IPSOM* de los mejores métodos evaluados según el tamaño de los problemas de Taillard, con el criterio de parada fijado a la evaluación de 50.000 “*makespans*” (Figura 3.14).

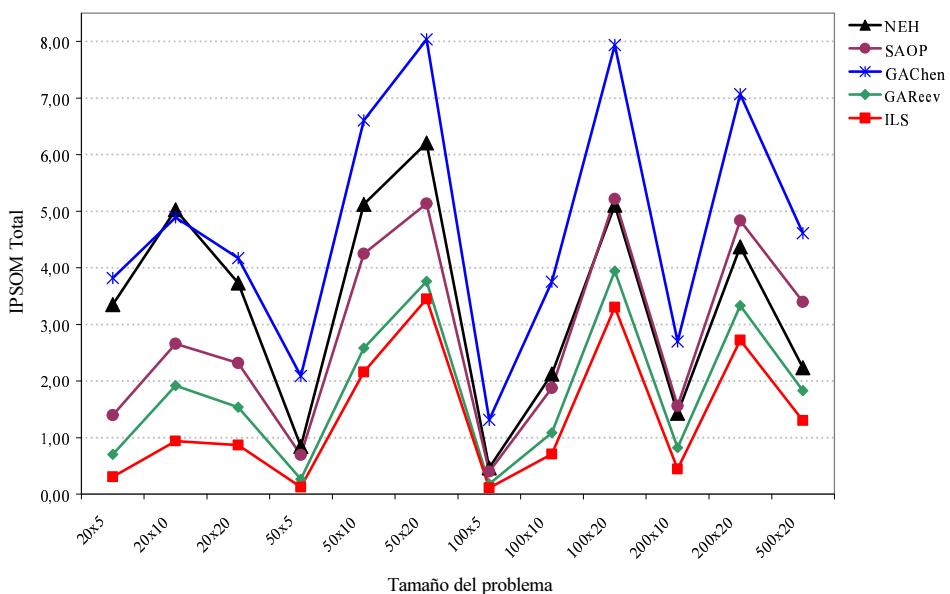


Figura 3.14 – Incremento porcentual total por encima del óptimo o mínima cota superior conocida (*IPSOM*) de los mejores métodos de la comparativa para todos los grupos de problemas de Taillard (criterio de parada 50.000 C_{max}).

Destaca el hecho de que algunos grupos de problemas resultan relativamente fáciles para todos los métodos mientras que otros grupos, como por ejemplo los problemas de 50 trabajos y 20 máquinas son difíciles para todos los métodos. Existe una alta similitud entre todos los algoritmos y como se puede observar, las distintas gráficas tienen prácticamente el mismo perfil, hasta tal punto que parece que los perfiles de rendimiento de los distintos métodos para los 12 grupos de problemas están simplemente desplazados en el eje de ordenadas.

Recientemente, Watson et al. (2002) han ampliado un estudio que presentaron con anterioridad (ver Watson et al., 1999) donde se examina el rendimiento de algunos métodos punteros para el flowshop de permutación, concretamente la heurística NEH y el tabu search de Nowicki y Smutnicki junto con algunas variantes. La aportación de este trabajo es que los autores utilizan problemas donde existe uno de los tres siguientes tipos de correlación en los tiempos de proceso:

1. Correlación en los trabajos. Los p_{ij} están correlacionados de acuerdo con el índice j . Es decir, hay que trabajos que son “largos” en todas las máquinas.
2. Correlación en las máquinas. Los p_{ij} están correlacionados de acuerdo con el índice i , o lo que es lo mismo, existen máquinas que procesan todos los trabajos j a una mayor o menor velocidad.
3. Correlación mixta. Es una mezcla de las dos anteriores.

Este tipo de correlaciones son muy comunes en la práctica. Es normal tener un trabajo que es muy costoso y por tanto va a llevar más tiempo en todas las máquinas del taller (correlación en los trabajos) o que alguna máquina sea más antigua que las demás y por tanto procese todos los trabajos a una menor velocidad (correlación en las máquinas). Los autores estudiaron las características de este tipo de problemas y la conclusión resultó ser muy interesante; los problemas de ejemplo con correlación resultan mucho más fáciles de resolver en la práctica que los problemas donde todos los p_{ij} son aleatorios. Esto viene a confirmar la creencia que ha existido siempre de que los problemas generados aleatoriamente son muy difíciles de resolver de manera óptima y por tanto constituyen un buen “benchmark” para los distintos algoritmos.

3.3. Conclusiones del capítulo

En este capítulo se ha presentado el problema del taller de flujo, concretamente la versión de permutación y se ha realizado una completa revisión del estado del arte de métodos exactos, heurísticos y metaheurísticos utilizados para resolver este problema, principalmente con el objetivo de la minimización de la fecha máxima de finalización o “*makespan*” (C_{max}). Asimismo se ha llevado a cabo una extensa evaluación comparativa donde se han codificado un total de 18 técnicas heurísticas, tanto constructivas como de mejora y siete métodos metaheurísticos incluyendo algoritmos de búsqueda tabú, recocido simulado, algoritmos genéticos y búsqueda local iterativa. Se ha utilizado como criterio de parada (en los algoritmos que lo requieren) el máximo número de C_{max} calculados. Para la comparativa se ha utilizado en conjunto de problemas ejemplo de Taillard (1993) y se ha obtenido como medida de eficacia el incremento medio porcentual sobre la solución óptima o mínima cota superior conocida. También se ha evaluado la eficiencia de los métodos considerando el tiempo transcurrido. Finalmente se ha estudiado el comportamiento de las técnicas metaheurísticas conforme se aumenta el máximo número de C_{max} en el criterio de parada y la dificultad de los distintos grupos de problemas dentro del conjunto de Taillard.

Como principal resultado tenemos una completa y actualizada revisión y una evaluación de 25 métodos para el problema $F/prmu/C_{max}$ que se ha realizado con el mismo banco de pruebas y en la misma plataforma hardware. Este trabajo viene a cubrir una carencia existente en la literatura sobre el tema puesto que no existen comparativas de heurísticas actuales así como tampoco de algoritmos metaheurísticos. Esta comparativa nos permite identificar los mejores algoritmos que pueden servir para resolver problemas de este tipo o que pueden adaptarse para resolver problemas más realistas en capítulos posteriores.

CAPÍTULO

4

NUEVOS ALGORITMOS GENÉTICOS PARA EL PROBLEMA DEL TALLER DE FLUJO

Como vimos en el capítulo anterior, se han propuesto numerosos algoritmos metaheurísticos para el problema del taller de flujo, concretamente para la versión de permutación, lo que ha convertido a este problema en un campo de estudio muy competitivo. En este capítulo vamos a presentar dos nuevos algoritmos genéticos desarrollados en esta Tesis Doctoral. Estos algoritmos se generalizan en capítulos posteriores para resolver problemas más complejos como los que representan la introducción de tiempos de cambio de partida dependientes de la secuencia y el taller de flujo híbrido, que son problemas menos estudiados en la literatura, pero de gran importancia práctica.

4.1. Funcionamiento general de los algoritmos genéticos

Los algoritmos genéticos (GAs) pueden verse como algoritmos estocásticos de búsqueda de soluciones que imitan en su funcionamiento el principio de evolución y selección natural de las especies de Darwin. De una manera muy general, en un GA tenemos una *población* de *individuos* que representan distintas soluciones para el problema que se pretende resolver. La “calidad” de un individuo, es decir, lo buena que es la solución que el individuo representa, se conoce como valor de *adecuación*. De esta manera, e imitando el proceso de selección natural, los mejores individuos (aquellos con un mayor valor de adecuación) se cruzarán en la fase de *cruce* más veces que los peores individuos. En el proceso de cruce se mezclarán las características de los padres creando una nueva generación de individuos previsiblemente mejores. Además, algunos de estos nuevos individuos pueden sufrir una *mutación* que modificará algunas de las propiedades heredadas de los padres. Tras cada generación, los individuos irán mejorando, es decir, obtendremos mejores soluciones para nuestro problema conforme evolucione la población. Este tipo de algoritmos fueron propuestos por Holland en 1975 y han sido aplicados en múltiples campos de estudio distintos. Vamos a utilizar las siguientes definiciones y notación para trabajar con los GAs:

- Representación genética (“*genetic representation*”): también conocida como representación de las soluciones o codificación de las soluciones. Es la función que traduce de la especificación de una solución del problema a un individuo de la población.
- Individuo (“*Individual*”): también llamado genotipo (“*genotype*”), estructura o a veces cromosoma (“*chromosome*”) o cadena (“*string*”). Es la representación en sí de una solución del problema. El individuo se compone de una serie de genes dispuestos en serie.
- Gen: también se le conoce como “*feature*”, “*character*” o “*detector*”. Representa una unidad indivisible de información sobre el individuo.
- Alelos (“*Allele*”): son los posibles valores que puede tomar cada gen. Normalmente existe un valor máximo y mínimo.

- Lugar (“*locus*”): son las posiciones que cada gen ocupa dentro del individuo.
- Población (“*Population*”): es el conjunto de individuos, al que llamaremos P .
- Fenotipo (“*phenotype*”): es el significado de cada individuo en términos del problema a resolver, el individuo descodificado. Normalmente el fenotipo es el valor de la función objetivo que representa el individuo.
- Valor de adecuación (“*fitness value*”): es una medida de lo bueno que es cada individuo dentro del ámbito del algoritmo genético. En algunos casos puede coincidir con el fenotipo, en otros no.
- Generación (“*generation*”): es la población en un instante concreto del proceso evolutivo, en una iteración dada del GA. Denotaremos por P_t a la población en la generación t .

Una vez definida la terminología, podemos describir el funcionamiento de un algoritmo genético. En la Figura 4.1 tenemos un sencillo diagrama de flujo que representa las etapas de un GA simple.

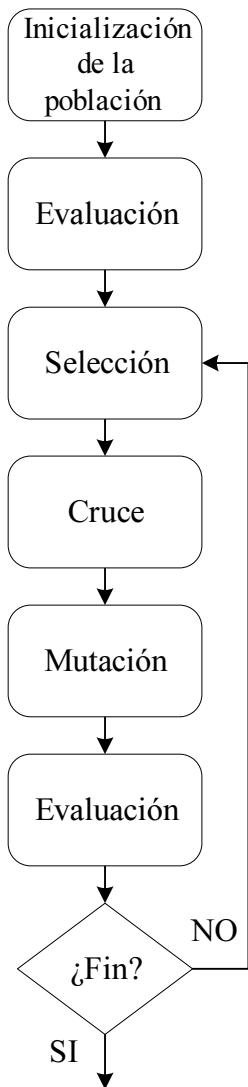


Figura 4.1 – Esquema general de funcionamiento de los algoritmos genéticos.

El primer paso consiste en crear una población inicial. El tamaño de la población que aquí llamaremos P_{size} es, como veremos, uno de los múltiples parámetros de control de un GA. En los GAs más simples la población inicial se

forma obteniendo P_{size} individuos generados de forma aleatoria. Esto es, todos los genes de cada uno de los individuos se llenan con alelos obtenidos al azar. Como se puede observar, el hecho de que los GAs trabajen con un conjunto de soluciones (individuos) es una característica claramente diferenciadora con respecto a otras técnicas metaheurísticas como tabu search o simulated annealing que trabajan con una única solución.

Tras la inicialización es necesario evaluar la población. En esta fase lo que se hace es calcular un valor de adecuación para cada individuo. Este valor de adecuación debe estar relacionado con la calidad real de cada solución, es decir, con el valor de la función objetivo asociada. Una vez se han calculado los valores de adecuación entramos en el bucle principal del algoritmo genético. La primera fase de este bucle es la selección, que simplemente elige los individuos que van a participar en la siguiente generación, de manera que aquellos con un mayor valor de adecuación participen de una manera más activa y tengan más probabilidades de generar descendencia. Después, en la fase de cruce, los individuos seleccionados se emparejan, y de acuerdo a una probabilidad de cruce P_c (otro parámetro del GA) se cruzan, generando dos nuevos individuos que reemplazan en la nueva generación a los padres. Si el proceso de cruce no se produce, los padres pasan inalterados a la siguiente generación. Este hecho permite que en una población convivan individuos provenientes de distintas generaciones.

Tras el cruce, los nuevos individuos generados (o las copias de los padres si el cruce no se llegó a producir) pueden sufrir una mutación de acuerdo al parámetro P_m o probabilidad de mutación. La mutación normalmente consistirá en la variación del alelo de algún gen escogido al azar en el individuo. La mutación tiene por finalidad introducir variabilidad en la población (nuevas características) o recuperar información que se había perdido en el proceso evolutivo. Una vez se ha terminado la mutación, se vuelven a evaluar todos los individuos y se comprueba la condición de terminación. Esta condición puede contemplar un número máximo de generaciones, de individuos generados (soluciones evaluadas), tiempo transcurrido, número de iteraciones, generaciones que han transcurrido sin mejora, etc... Más información sobre los algoritmos genéticos puede encontrarse en el trabajo original de Holland (1975) y en Goldberg (1989), Michalewicz (1996) o Davis (1996).

Ahora vamos a hacer referencia a la aplicación de los GAs al ámbito de la programación de la producción y concretamente al taller de flujo de permutación.

4.2. Algoritmos genéticos y el problema del taller de flujo de permutación

Antes de comentar las distintas fases de los GAs aplicados al $F/prmu/C_{max}$ es necesario considerar cómo se van a codificar las soluciones, es decir, cuál va a ser la función de representación genética.

4.2.1. Representación genética

Un GA trabaja sobre los individuos o cromosomas, que como se ha comentado, son una representación de las soluciones del problema, no las soluciones en sí. En el caso del taller de flujo la solución viene dada por la programación de todas las tareas, es decir, necesitaríamos saber los instantes de comienzo y finalización de cada tarea, esto es, los valores de $O_{ij}s$ y $O_{ij}f$, $i = (1, \dots, m)$, $j = (1, \dots, n)$. En realidad, para el caso concreto del taller de flujo de permutación no es necesaria tanta información, dado que el orden de entrada de los trabajos es el mismo para todas las máquinas y por tanto, a partir de la permutación de n elementos y las funciones para el cálculo del C_{max} que se vieron al principio de la Sección 3.1, es posible obtener la programación de las tareas sin necesidad de explicitar tanta información. Este hecho, unido a la gran similitud que existe entre los problemas del viajante de comercio (TSP) y el taller de flujo (como se vio en el capítulo anterior) ha motivado el uso de la *representación ordinal* para el taller de flujo. En esta representación el individuo tiene n genes, tantos como trabajos. Los genes pueden tomar valores entre 1 y n y los lugares que ocupa cada gen son muy importantes. De esta manera cada gen junto con su valor correspondiente (alelo) representa uno de los trabajos en el taller. La posición o locus de cada gen o trabajo en el individuo indicará el orden de procesamiento de éste. Un posible individuo siguiendo la representación ordinal para el primer ejemplo de los 120 de Taillard (ta001), de 20 trabajos y 5 máquinas, se muestra en la Figura 4.2.

3	17	9	15	8	14	11	13	19	6	7	5	1	4	2	18	16	10	20	12
---	----	---	----	---	----	----	----	----	---	---	---	---	---	---	----	----	----	----	----

Figura 4.2 – Ejemplo de representación genética ordinal para el problema ta001 de Taillard (20 trabajos).

Como vemos, esta representación depende tan solo del número n de trabajos y no del número de máquinas m . De esta manera, tenemos que el trabajo 3 se procesaría en primer lugar en las 5 máquinas, después entraría el trabajo 17 en todas las máquinas y así sucesivamente hasta llegar al trabajo 12 que se procesaría en último lugar. Como vemos, con esta representación es posible codificar todas las $n!$ posibles soluciones del taller de flujo de permutación, luego la codificación ordinal es completa, esto es, permite codificar todo el espacio de soluciones en el GA. Para que un individuo sea factible, hemos de asegurarnos que no hay dos genes con el mismo alelo, y que no hay ningún alelo faltante, o lo que es lo mismo, no pueden haber trabajos repetidos o trabajos perdidos.

El fenotipo, o la decodificación del individuo es muy sencilla, simplemente aplicaríamos las citadas fórmulas de cálculo del C_{max} para obtener la programación de todas las tareas (programa de producción). Para este ejemplo el fenotipo después de evaluar el individuo es el diagrama de Gantt que se muestra en la Figura 4.3.

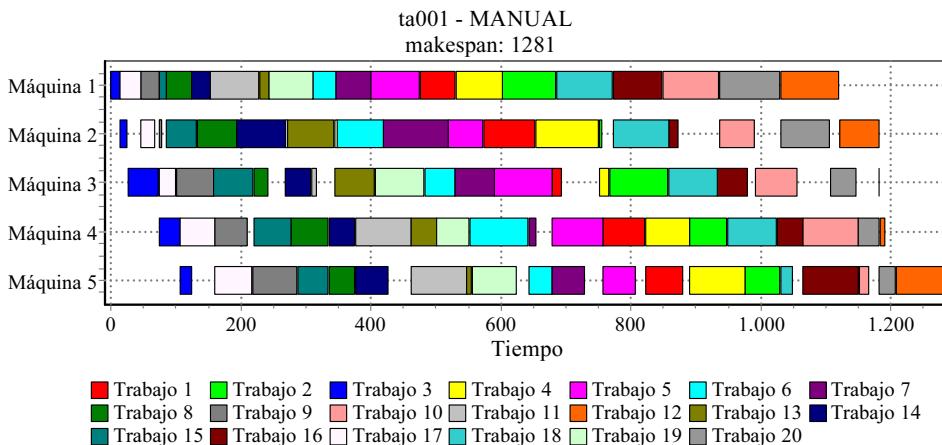


Figura 4.3 – Programa de producción a partir de la representación genética ordinal para el problema ta001 de Taillard.

Como vemos, al decodificar el individuo del ejemplo obtenemos una secuencia con un C_{max} de 1281, siendo el óptimo (Tabla 3.8) de 1278.

Que nosotros sepamos, no existen codificaciones alternativas a la ordinal. Realmente, para el $F/prmu/C_{max}$, esta representación proporciona muy buenos resultados y es la que se ha utilizado en todos los trabajos sobre algoritmos genéticos publicados hasta la fecha para este problema (ver la Sección 3.1 del Capítulo 3). Para otros problemas de planificación de la producción, como por ejemplo el $J//C_{max}$, se han propuesto codificaciones alternativas (ver Davis, 1985b y Bruns, 1993).

4.2.2. Evaluación

Con la evaluación asignamos a cada individuo un valor de adecuación o “*fitness value*” que indica la calidad del individuo con respecto al total de la población. De manera directa se podría asignar el valor del C_{max} al valor de adecuación. No obstante, en el funcionamiento del GA, y concretamente, en la fase de selección, un valor de adecuación alto indica un individuo bueno. En nuestro caso, un valor del C_{max} bajo indica un buen individuo. Para solucionar este problema lo que se suele hacer es *mapear* el C_{max} obtenido a un valor

de adecuación de manera que un C_{max} bajo se corresponda con un valor de adecuación alto. Denotamos por $P_{t(l)}$ al individuo l de la población P en la generación t , y por $C(P_{t(l)})$ y $f(P_{t(l)})$ al C_{max} y valor de adecuación de este individuo l . Un posible mapeado que se propone en Goldberg (1989) y que se utiliza en el GA de Chen, Vempati y Aljaber (1995) (entre otros) es el siguiente:

$$\begin{aligned} f(P_{t(l)}) &= \max\{C(P_{t(1)}), C(P_{t(2)}), \dots, C(P_{t(P_{size})})\} - C(P_{t(l)}) \\ l &= (1, \dots, P_{size}) \end{aligned}$$

De esta manera el peor individuo de la población, es decir, aquel con un mayor C_{max} , tendrá un valor de adecuación de cero y el mejor individuo, el de menor C_{max} , tendrá el valor de adecuación más alto.

Existe un problema adicional con este tipo de funciones de evaluación mediante mapeados y con los GAs en general. Puede ocurrir que en las primeras generaciones del GA haya uno o más individuos con un valor de adecuación muy superior a la media, con esto, el mecanismo de selección (que veremos más adelante) seleccionará siempre estos individuos y el resultado es que tras unas pocas generaciones la población será muy homogénea al haber participado tan solo unos pocos individuos en el proceso genético. Este problema se conoce como *convergencia prematura*. De igual manera también se puede dar el caso contrario, que haya muy poca variabilidad en los valores de adecuación y ningún individuo o individuos dominen al resto ocurriendo que la selección se parecerá poco a la selección natural y se convertirá en una selección aleatoria, por lo que la población no evolucionará. A este otro problema se le conoce como *convergencia lenta*. Para solucionar estos inconvenientes se utilizan técnicas de *escalado* en las funciones de evaluación que aumentan las diferencias entre los valores de adecuación cuando estas son pequeñas y las reducen cuando son grandes.

4.2.3. Selección

Es el proceso de selección el que se va a ocupar de que los individuos con un mayor valor de adecuación participen más en la construcción de la siguiente generación. Se trata de imitar la naturaleza en el principio de la supervivencia de

los más fuertes. En la construcción del operador de selección hay dos aspectos a tener en cuenta, la *presión* y la *diversidad* del proceso. Si sólo los individuos más fuertes participan en la evolución, la población degenerará en un óptimo local donde todos los individuos serán copias de los mejores; se perderá diversidad. En cambio, si todos los individuos, independientemente de su valor de adecuación, participan en el proceso genético, el algoritmo será incapaz de mejorar el valor medio de adecuación; perderemos presión, los peores individuos permanecerán en la población generación tras generación y el GA degenerará en una búsqueda local.

El mecanismo más sencillo de selección es el que selecciona individuos de acuerdo a una probabilidad que se asigna a cada individuo y que es proporcional al valor de adecuación. Otro tipo de selección, llamada de “*Boltzmann*” no requiere escalado y trabaja haciendo una transformación exponencial del valor de adecuación. Uno de los métodos de selección más conocidos es la selección por *ruleta* o selección simple, cuyo funcionamiento es el siguiente: se divide una ruleta en tantos sectores como individuos hay en la población, donde el tamaño de cada sector es proporcional al nivel de adecuación del individuo. De esta forma se extrae un número aleatorio en el rango $[0, \dots, 1]$, se busca en qué sector cae este número dentro de la ruleta y se devuelve el individuo seleccionado. El Listado 4.1 muestra el funcionamiento de este tipo de selección en forma de pseudoalgoritmo.

```

function Selección_Ruleta( $P$ : Population): individual;
begin
    //calculamos la suma  $F$  de los valores de adecuación de la población
    for  $l := 1$  to  $P_{size}$  do
         $F := F + f(P_{t(l)})$ ;
    //calculamos la probabilidad  $p_l$  de cada individuo
    for  $l := 1$  to  $P_{size}$  do
         $p_l := f(P_{t(l)})/F$ ;
    //calculamos la probabilidad acumulada  $q_l$  hasta cada individuo
    for  $l := 1$  to  $P_{size}$  do
        for  $m := 1$  to  $l$  do
             $q_l := q_l + p_m$ ;
    //obtenemos un número aleatorio uniforme entre 0 y 1
    num := rand();
    //vemos donde cae num en la ruleta
    for  $l := 1$  to  $P_{size}$  do
        if num <  $q_l$  then return  $l$ 
    end;

```

Listado 4.1 – Pseudoalgoritmo del operador de selección por ruleta o selección simple.

En los mecanismos de selección que hemos visto no es posible garantizar el número de veces que un individuo de la población va a ser seleccionado. En la selección por *muestreo determinístico* o en las selecciones por *muestreo estocástico sin reemplazamiento* y *muestreo estocástico con reemplazamiento* se calcula el número de veces que se espera que un individuo sea seleccionado E_l , donde $E_l = p_l \cdot P_{size}$, $l = (1, \dots, P_{size})$. De esta manera, cada individuo se selecciona tantas veces como indica E_l .

Otro tipo de selección muy utilizado en la práctica es la selección por *torneo* o por *competición*. El funcionamiento es muy sencillo, se hacen k llamadas a la selección por ruleta y de los k individuos resultantes se selecciona aquel que tiene un mayor valor de adecuación. Por norma general se utiliza el *torneo binario* donde $k = 2$.

Por último, en la selección por *rango* o “*ranking selection*”, se ordenan los individuos por orden creciente de sus valores de adecuación y el valor de p_l se calcula en proporción al lugar que los individuos ocupan en el “*ranking*” en vez de calcularse en proporción a su valor de adecuación. Concretamente, si denotamos por $index_{(l)}$ a la posición en este ranking que ocupa el individuo l , la probabilidad

de seleccionar este individuo se calcula como:

$$p_l = \frac{index(l)}{\sum_{m=1}^{P_{size}} index(m)}, \quad l = (1, \dots, P_{size})$$

Como podemos observar, la probabilidad es independiente del valor de adecuación, luego la selección por rango no tiene problemas de escala y por tanto no es necesario hacer la operación de escalado.

4.2.4. Cruce

El cruce puede verse como la reproducción de los seres vivos. Una vez hemos seleccionado los individuos “buenos” de la población en el proceso de selección, éstos se emparejan y de acuerdo a una probabilidad P_c , que es un parámetro de control del GA, se reproducen. El Listado 4.2 muestra el funcionamiento de la fase de cruce.

```

procedure cruce(padre, madre: individual; var hijo,hija: individual);
begin
    //obtenemos un número aleatorio uniforme entre 0 y 1
    num := rand();
    //¿cruzamos?
    if num < P_c then
        procedimiento_cruce(padre,madre,hijo,hija);
    else
        begin
            hijo := padre;
            hija := madre;
        end;
    end;
end;

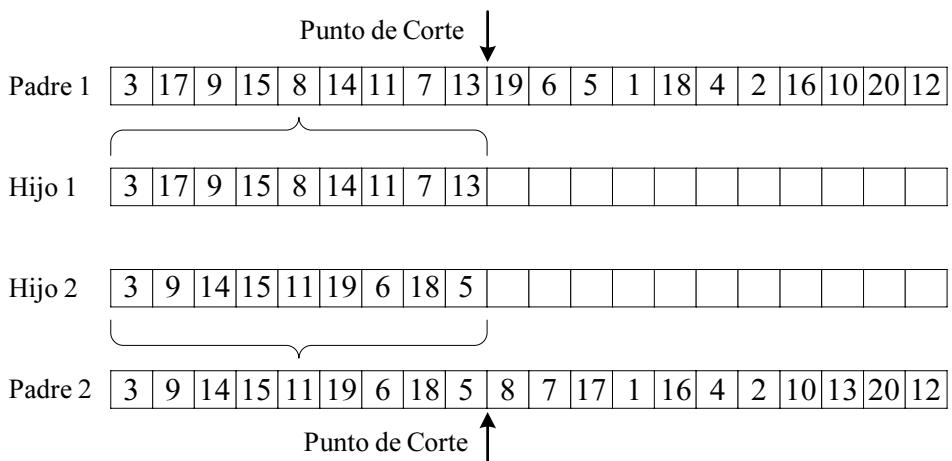
```

Listado 4.2 – Pseudoalgoritmo con el funcionamiento de la fase de cruce genético.

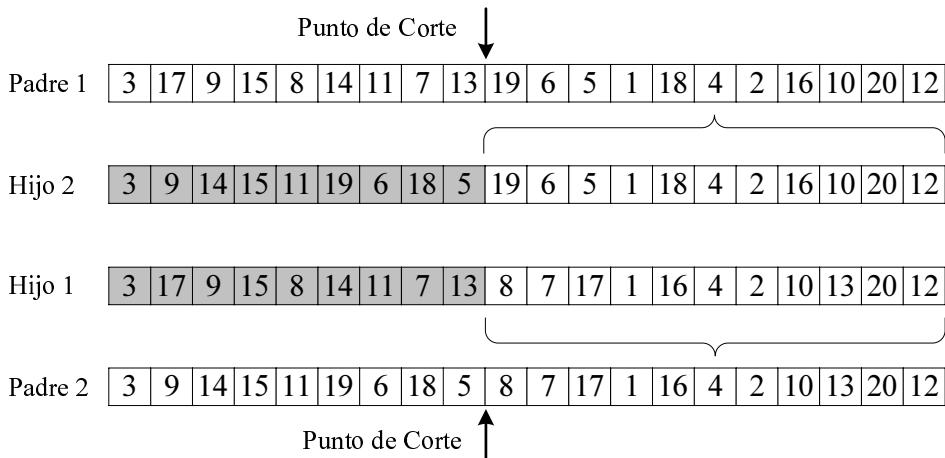
Como se puede observar, si la pareja de “padres” no se cruzan, pasarán inalterados a la siguiente generación, en cambio si se cruzan habrá que aplicar un procedimiento de cruce para generar nuevos individuos. De manera general, el cruce combina los individuos progenitores para generar nuevos individuos, la descendencia (“*offspring*”). El objetivo que se persigue es crear nuevas soluciones

para nuestro problema de manera que tengan las mejores características de los padres. Evidentemente, identificar por qué dos soluciones dadas son buenas y construir una solución mejor es difícil, de ahí que se hayan propuesto numerosos procedimientos de cruce.

Uno de los cruces más simples es el llamado *cruce de un punto*. A partir de los dos padres (que llamaremos padre 1 y padre 2), se establece al azar una posición común de corte. De esta manera los hijos (hijo 1 e hijo 2) heredan los genes antes del punto de corte de un parente y los genes después del punto de corte del otro parente. Podemos ver este cruce, ejecutado en dos fases, en la Figura 4.4.



(a) Los hijos heredan la parte anterior al punto de cruce de uno de los padres.



(b) Luego heredan la parte posterior al punto de cruce del otro parente.

Figura 4.4 – Operador de cruce de un punto estándar.

El problema con este cruce es que los hijos normalmente no serán soluciones posibles, si nos fijamos bien, los hijos generados con este cruce tienen posiciones con alelos repetidos, es decir, hay trabajos que aparecen repetidos en la secuencia, y al haber repeticiones, hay trabajos que faltan. Este problema afecta a todos los operadores de cruce cuando se trabaja con la representación genética ordinal. El número de posiciones erróneas es elevado, tal y como se muestra en la Figura 4.5.

Hijo 1	[3 9 15 8 14 11 13 8 7 17 1 16 4 2 10 13 20 12]
--------	--

Hijo 2	[3 9 14 15 11 19 6 18 5 19 6 5 1 18 4 2 16 10 20 12]
--------	--

Figura 4.5 – Errores en el cruce de un punto estándar.

Adicionalmente a la dificultad de diseñar un operador de cruce que genere buenos hijos, está la problemática de que los hijos sean factibles. Para ello, varios autores han propuesto modificaciones a los operadores de cruce estándar para que generen hijos factibles cuando se trabaja sobre la representación ordinal

anteriormente expuesta. Por ejemplo, en el *cruce de un punto por orden* se modifica el segundo paso del anterior cruce de un punto de manera que los genes se heredan del otro parente en el orden relativo en el que aparecen. Es decir, los alelos que faltan en el hijo se heredan del otro parente en el orden en que aparecen en éste. Siguiendo el ejemplo anterior, aplicamos la primera fase del *cruce de un punto* y se modifica la segunda fase de la forma que se muestra en la Figura 4.6.

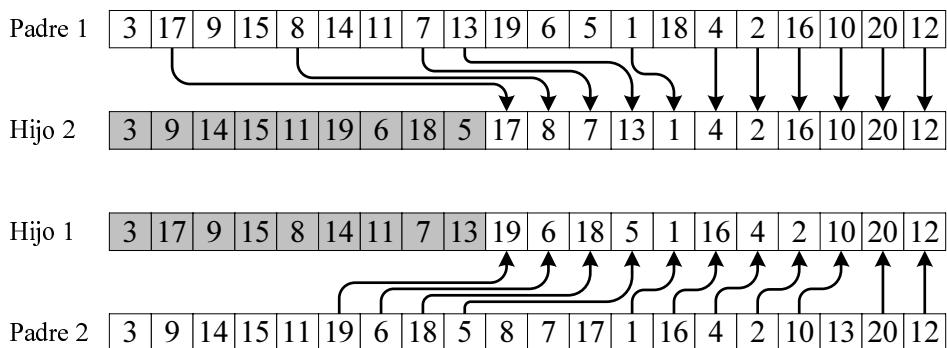
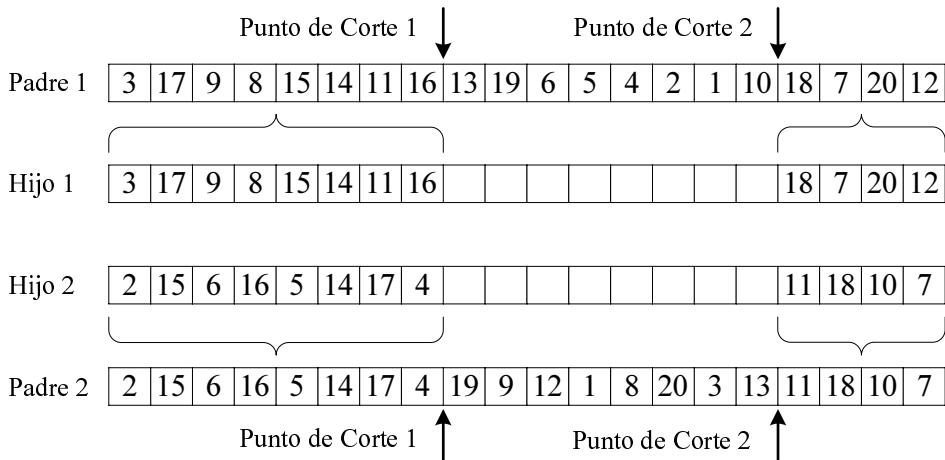
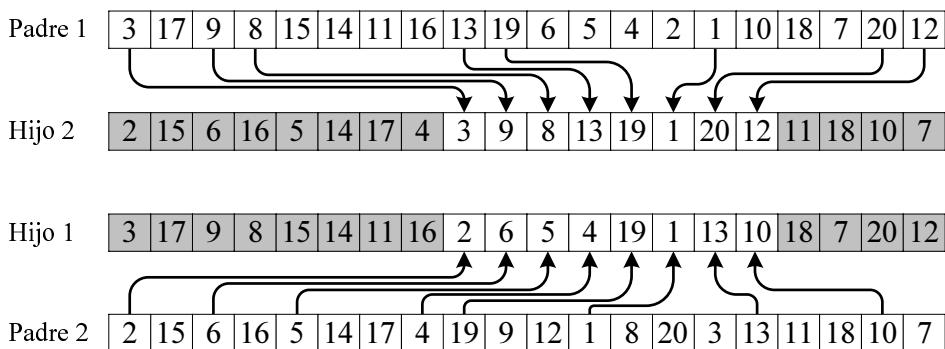


Figura 4.6 – Segunda fase del cruce de un punto por orden.

De esta manera siempre conseguimos que los hijos sean factibles. Este cruce se utiliza por ejemplo en el GA de Reeves (1995). Una variación a este cruce muy utilizada en la práctica es el *cruce de dos puntos por orden*, el funcionamiento es muy parecido a la versión con un punto, con la excepción de que existen dos puntos de corte en vez de uno, de manera que los padres quedan separados en tres partes. En Murata, Ishibuchi y Tanaka (1996a) se estudian varias versiones de este operador entendiendo a qué parte de cada parente se copia en los hijos. Aquí comentamos la versión I de este operador de acuerdo al citado trabajo. En esta versión, la parte antes del primer punto de cruce y la parte que viene después del segundo punto de cruce se copian directamente de uno de los padres. La parte que queda entre los dos puntos de cruce se copia en el orden que aparece en el otro parente. Un ejemplo de este tipo de cruce se muestra en la Figura 4.7.



(a) Los hijos heredan la parte anterior al primer punto de cruce y la parte posterior al segundo punto de cruce de uno de los padres.

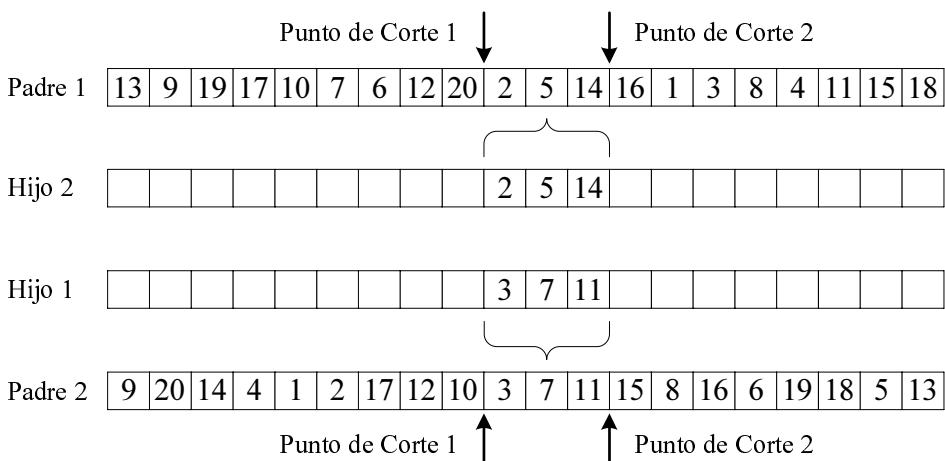


(b) Luego heredan la parte que queda entre los puntos de cruce en el orden relativo del otro parente.

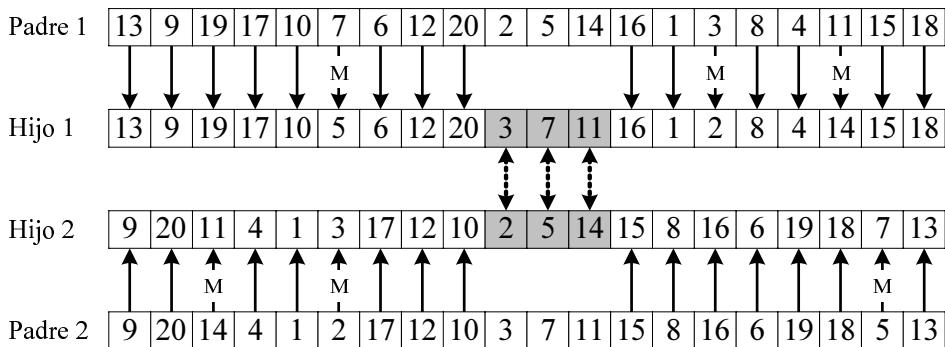
Figura 4.7 – Operador de cruce de dos puntos por orden.

La investigación de la aplicación de los GAs al problema del viajante de comercio (TSP) es directamente aplicable al taller de flujo de permutación dado que un tour del viajante se representa de manera natural mediante la codificación genética ordinal anteriormente expuesta. De esta manera se han propuesto varios operadores que pasamos a comentar a continuación. Uno de los operadores para el

TSP más citado es el “*Partially Matched Crossover*” (PMX) de Goldberg y Lingle (1985), (algunos autores se refieren a este operador como “*Partially Mapped Crossover*”). Este operador se ha utilizado también en un GA para el taller de flujo en Chen, Vempati y Aljaber (1995) y también en otros entornos como en GAs aplicados a la fabricación flexible en celdas (ver Jawahar et al., 1998). En este caso se utilizan dos puntos de corte que determinan una región que se copia inalterada del otro parente a cada hijo tal y como se muestra en la Figura 4.8(a). Esta región establece lo que se conoce como parejas de mapeado entre ambos padres. En el segundo paso del cruce, los hijos heredan las posiciones faltantes (antes y después de los puntos de cruce) del parente directo, con la salvedad de que hay que aplicar el mapeado para aquellas posiciones que ya se copiaron en la fase primera (Figura 4.8(b)).



(a) Los hijos heredan la parte entre los puntos de corte del otro parente.



(b) Luego heredan las posiciones faltantes del padre directo, pero teniendo en cuenta el mapeado.

Figura 4.8 – Operador de cruce “*Partially Matched Crossover*” (PMX).

En el ejemplo de la Figura 4.8, podemos ver como en el segundo paso, la posición 6 del padre 1 se copia directamente al hijo 1, esta posición contiene el alelo 7, que ya se ha copiado en el paso 1 del cruce, por tanto se miran las parejas de mapeado, que indican los intercambios entre los alelos del padre 1 y 2, $2 \leftrightarrow 3$, $5 \leftrightarrow 7$ y $14 \leftrightarrow 11$. Según esto, el alelo 7 se mapea al alelo 5, de ahí el intercambio. En general, serán necesarias tantas operaciones de mapeado como posiciones tenga la región entre ambos puntos de corte. El cruce PMX es considerado como uno de los mejores para el problema del viajante de comercio y también ha demostrado un buen rendimiento para el problema del taller de flujo (ver Chen, Vempati y Aljaber, 1995).

Existe un inconveniente con este operador de cruce al que no se da solución en el trabajo original de Goldberg y Lingle y es cuando hay una estructura del tipo $a \leftrightarrow b$, $b \leftrightarrow c$ en las parejas de mapeado. Este problema causa infactibilidad en los hijos como el siguiente ejemplo muestra.

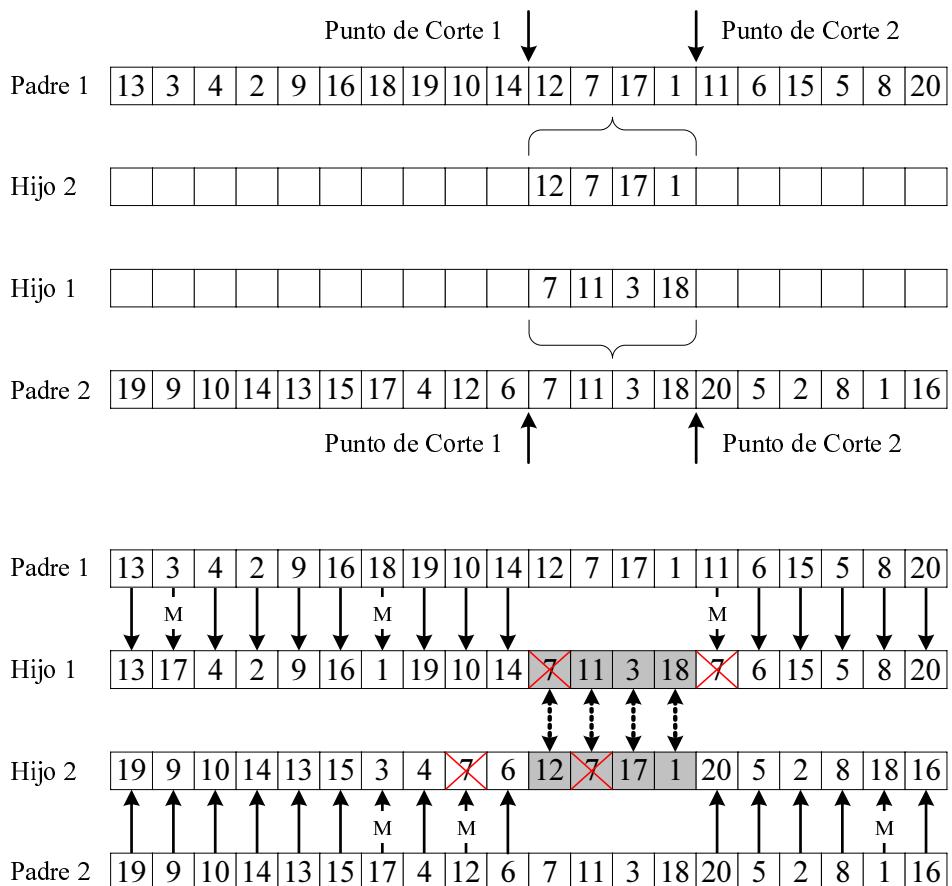


Figura 4.9 – Fallos en el operador de cruce “*Partially Matched Crossover*” (PMX) estándar.

Como se puede observar, la aplicación directa del mapeado causa que el alelo 7 esté repetido en ambos hijos y que los alelos 12 y 11 estén ausentes en los hijos 1 y 2 respectivamente, por ello ambos hijos son infactibles. Chen, Vempati y Aljaber aplican un algoritmo no especificado de “reconstrucción” que analiza los hijos creados y repara el cromosoma, eliminando los alelos duplicados y reintroduciendo los alelos faltantes. La solución propuesta por Cotta y Troya (1998) es mucho más acertada. Los autores proponen continuar aplicando el mapeado tantas veces como sea necesario hasta que los hijos sean factibles. De

esta manera, el paso 2 del cruce PMX se modifica de la siguiente manera: en la Figura 4.9, al generar el hijo 1 a partir del padre 1, la posición 15 contiene el alelo 11 en el padre, que se mapea al alelo 7. Como vemos, este alelo 7 ya está copiado, luego buscamos como seguir mapeando este 7. Según la pareja de mapeado $7 \leftrightarrow 12$ este alelo se mapea por el 12. Así que intercambiaremos el trabajo 11 en la posición 15 por el trabajo 12 aplicando dos mapeados. De la misma manera el trabajo 12 en la posición 9 termina mapeándose por el trabajo 11 al aplicar el mapeado dos veces. La solución correcta se muestra en la Figura 4.10.

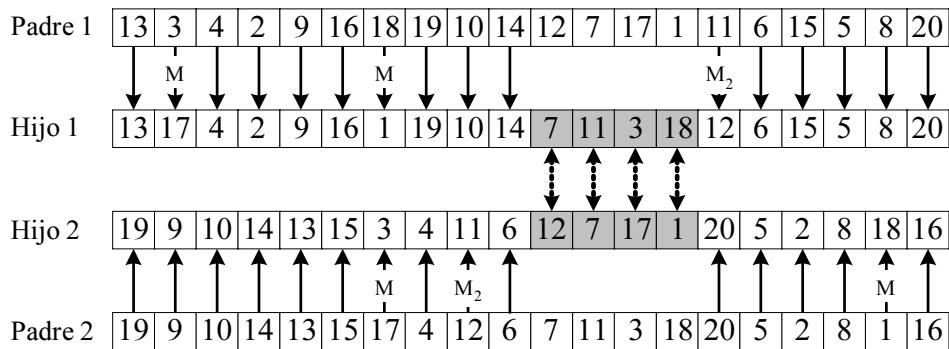
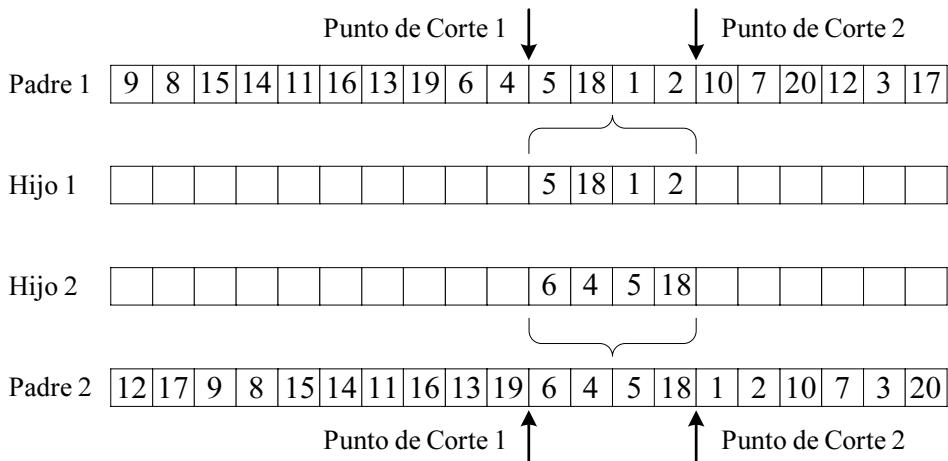
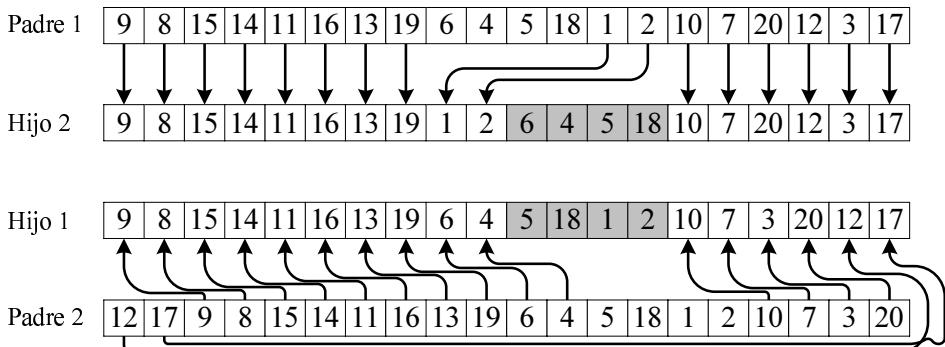


Figura 4.10 – Resultado de aplicar el operador PMX modificado.

Otro operador de común aplicación para TSP es el “*Order Crossover*” (OX) propuesto por Davis (1985a). En este caso se utilizan también dos puntos de cruce y la parte interior entre estos puntos se hereda del padre directo. Las posiciones libres se rellenan en los hijos a partir del segundo punto de cruce en el orden relativo del otro parente. Vamos a ver el funcionamiento de este operador con un ejemplo que se muestra en la Figura 4.11.



(a) En la primera fase se hereda el segmento entre los puntos de corte del padre directo.



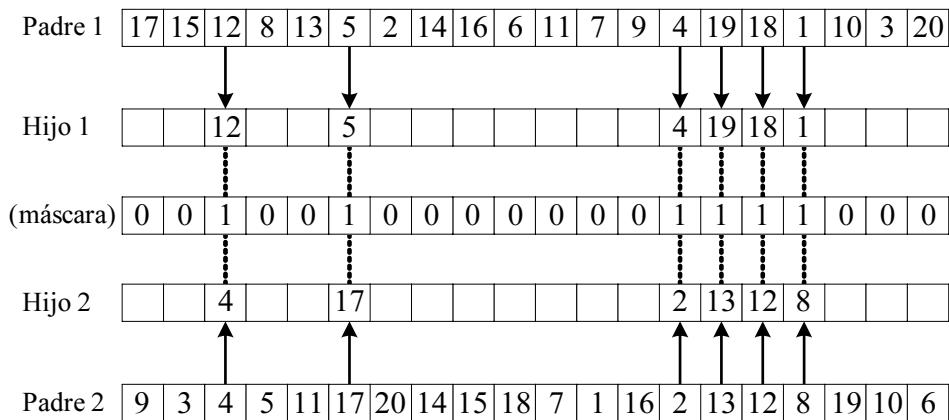
(b) Después, se heredan los alelos del otro parente que no se hayan heredado y a partir del segundo punto de corte.

Figura 4.11 – Operador de cruce “Order Crossover” (OX).

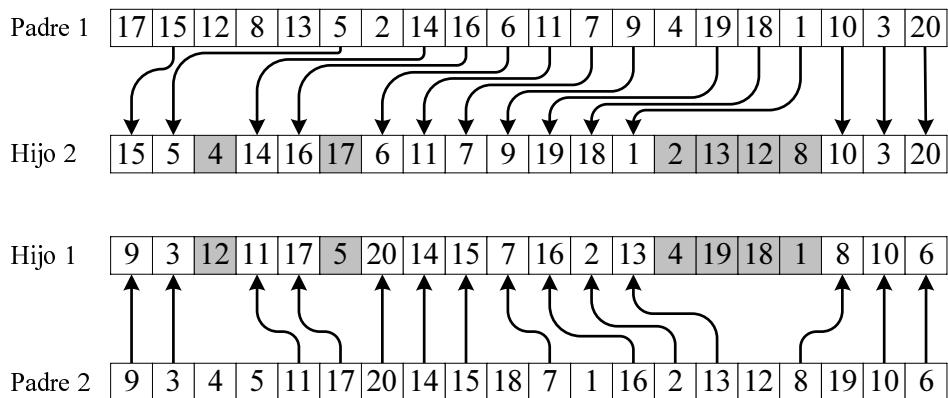
Si nos fijamos en la Figura 4.11(b), y en concreto en la generación del hijo 1. Vemos que empezamos a heredar los alelos del otro parente (padre 2) a partir de la posición siguiente al segundo punto de corte. En este caso la posición contiene el alelo 1 que ya se ha heredado, por lo que pasamos a la siguiente posición, donde se repite el mismo caso. La primera posición que se hereda al hijo es la que contiene

el alelo 10. Una vez llegamos a la posición n en el padre, iremos a la primera posición para continuar el proceso hasta que no queden posiciones libres en los hijos.

Un operador muy interesante surge de las ideas de los operadores de cruce uniformes de Spears y De Jong (1991) y del trabajo de Syswerda (1996). Se trata simplemente de una extensión del cruce uniforme para la representación ordinal. El cruce se llama “*Uniform Order Based Crossover*” (UOB) y trabaja de una manera ligeramente distinta a lo visto hasta el momento. En vez de establecer uno o dos puntos de corte, el operador UOB trabaja con una máscara de bits con la misma longitud que el cromosoma (n). En cada posición de esta máscara podremos encontrarnos con los valores 1 ó 0. El valor 1 indica que esa posición se hereda directamente del padre directo mientras que un valor de 0 indica que esa posición se heredará del otro parente, pero en su orden relativo. El funcionamiento es entonces muy similar a los cruces de uno o dos puntos basado en orden vistos anteriormente pero con múltiples puntos de corte que vienen dados por la máscara. Un ejemplo de este tipo de cruce se muestra en la Figura 4.12.



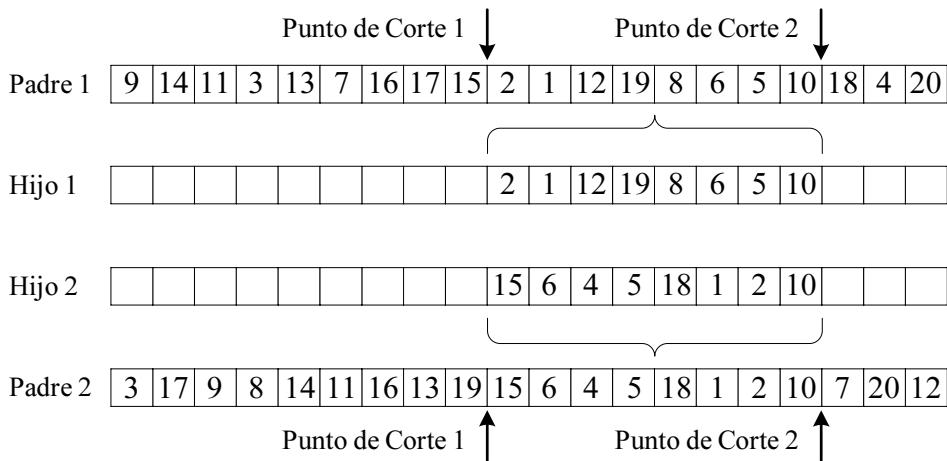
(a) Se heredan del parente directo los alelos en las posiciones donde la máscara tiene el valor 1.



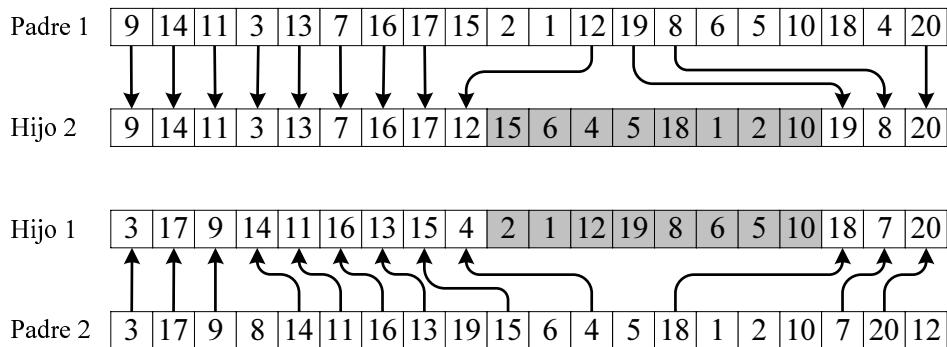
(b) Los alelos que faltan en cada hijo se heredan del otro parente en su orden relativo.

Figura 4.12 – Operador de cruce “Uniform Order Based Crossover” (UOB).

Un operador más reciente es el “*Generalized Position Crossover*” o GPX, propuesto por Mattfeld (1996) y que se ha aplicado al taller de flujo de permutación en Ponnambalam, Aravindan y Chandrasekaran (2001). En realidad puede verse simplemente como una variación del cruce de dos puntos por orden, de hecho Murata, Ishibuchi y Tanaka (1996a) evalúan una variante del cruce de dos puntos por orden al que llaman versión II que resulta ser idéntico al cruce GPX. En este cruce se hereda la sección que queda entre los dos puntos de cruce del parente directo en la primera fase. En la segunda fase se heredan las posiciones faltantes del otro parente en el orden en el que aparecen en éste. Un ejemplo puede verse en la Figura 4.13.



(a) Se heredan del padre directo los alelos entre los dos puntos de corte.



(b) Los alelos faltantes se heredan del otro parental en su orden relativo.

Figura 4.13 – Operador de cruce “*Generalized Position Crossover*” (GPX).

4.2.5. Mutación

El efecto del proceso de mutación, como su nombre indica y siguiendo la analogía de la genética, es la variación de uno o más de los genes de los individuos. El objetivo de este proceso es introducir información nueva en la población o

recuperar información que alguna vez estuvo en la población pero que se perdió en el proceso de cruce. Si con los procesos de selección y de cruce buscamos mejorar la población y una convergencia hacia mejores valores de la función objetivo, con el proceso de mutación nos aseguramos de que toda posible solución para el problema tiene una probabilidad no nula de ser explorada.

Algunos autores aplican el proceso de mutación a todo el individuo, es decir, en un único paso se determina si el individuo va a mutar, de acuerdo a una probabilidad de mutación P_m y si en efecto muta, se varían algunos de sus genes. No obstante, lo más común es realizar el proceso de mutación por cada posición del cromosoma. De esta manera se recorren todos los genes y se determina, para cada uno de ellos si hay mutación. El Listado 4.3 muestra el funcionamiento genérico del operador de mutación.

```
procedure mutacion(var hijo: individual);
begin
  //recorremos todas las posiciones del individuo
  for j:=1 to n do
    begin
      //obtenemos un número aleatorio uniforme entre 0 y 1
      num := rand();
      //¿mutamos?
      if num < Pm then
        procedimiento_mutacion(hijo, j);
    end;
  end;
```

Listado 4.3 – Pseudoalgoritmo con el funcionamiento de la fase de mutación genética.

La forma de mutar un gen que generalmente se utiliza consiste simplemente en cambiar su alelo por otro valor. Este procedimiento, como ya viéramos en el operador de cruce, puede originar individuos no factibles cuando se utiliza la representación ordinal. De esta manera, los procedimientos de mutación más utilizados para la representación ordinal se basan en la variación de las posiciones que ocupan los genes, más que en la modificación de sus alelos. De manera general se han propuesto tres tipos de mutación para el problema del taller de flujo y que pasamos a comentar. El primer tipo de mutación, llamada mutación por intercambio o “SWAP”, se basa en la trasposición de las posiciones de dos trabajos. Más concretamente, dadas dos posiciones j y k en el cromosoma,

$j < k$, $j, k = (1, \dots, n)$ la mutación SWAP asigna el alelo de la posición k a la posición j y el alelo de la posición j a la posición k . Un ejemplo de este tipo de mutación se muestra en la Figura 4.14. Esta mutación está muy relacionada con los movimientos de intercambio y el vecindario E vistos en la Sección 3.1.5. Este tipo de mutación se utiliza por ejemplo en Ponnambalam, Aravindan y Chandrasekaran (2001).

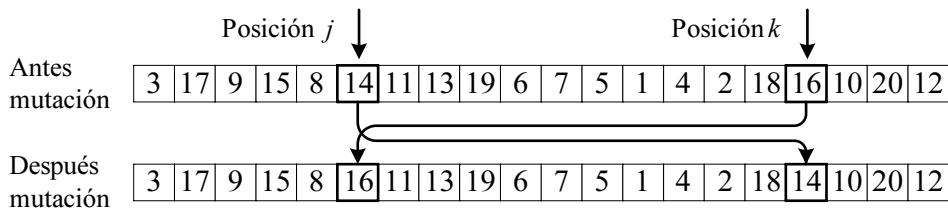


Figura 4.14 – Mutación por intercambio (“*SWAP mutation*”).

Otro esquema de mutación surge de limitar la mutación SWAP para que sólo se intercambien trabajos adyacentes, en este caso, dadas dos posiciones j y k en el cromosoma, tenemos que $k = j + 1$, $j = (1, \dots, n)$. A este tipo de mutación se la conoce como “*POSITION mutation*” y un ejemplo aparece en la Figura 4.15.

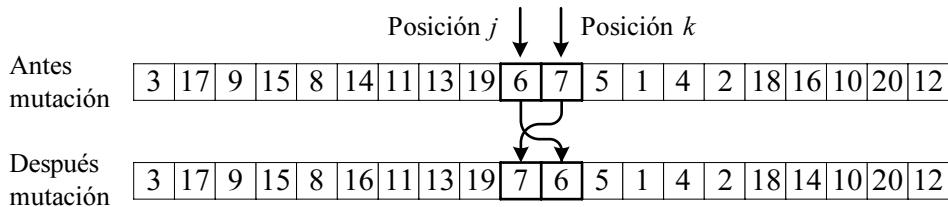


Figura 4.15 – Mutación por intercambio adyacente (“*POSITION mutation*”).

Por último, uno de los esquemas más utilizados es la mutación por desplazamiento. Por ejemplo Reeves (1995), Murata, Ishibuchi y Tanaka (1996a) o Tang y Liu (2002) hacen uso de este tipo de operador que funciona de forma muy parecida al vecindario de inserción; se escogen dos posiciones j y k al azar donde $j \neq k$, $j, k = (1, \dots, n)$, se extrae el alelo que ocupa la posición j y se inserta en

la posición k de manera que los alelos entre las posiciones k y $j - 1$ (en el caso en que $k < j$) o entre las posiciones j y $k - 1$ (en el caso en que $j < k$) pasan a ocupar las posiciones entre $k + 1$ y j , si $k < j$ o entre $j + 1$ y k si $j < k$. Un ejemplo de este tipo de mutación, con $k < j$ puede verse en la Figura 4.16.

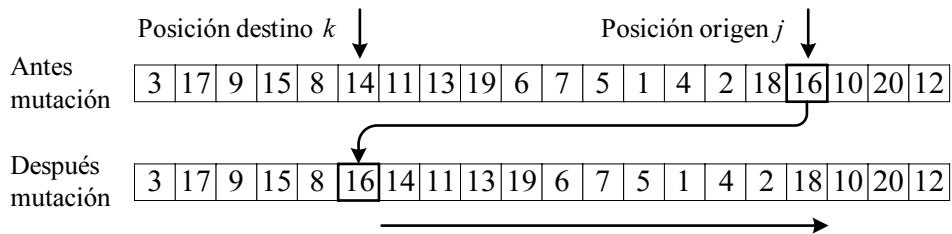


Figura 4.16 – Mutación por desplazamiento (“SHIFT mutation”).

4.3. Un nuevo algoritmo genético

Tras ver en detalle cómo funcionan los algoritmos genéticos en general y en particular para el problema del taller de flujo vamos ahora a exponer un nuevo algoritmo genético que hemos desarrollado en esta Tesis Doctoral. A lo largo de la presente sección explicaremos todas las alternativas de operadores y selección de parámetros que se han considerado en la construcción del GA propuesto. Se detallará un amplio experimento computacional encaminado a validar los operadores propuestos y a conseguir una precisa parametrización del GA. Posteriormente se comparará este nuevo algoritmo con las mejores heurísticas y metaheurísticas evaluadas en el capítulo anterior.

4.3.1. Representación, evaluación e inicialización de la población

Para la codificación de los individuos vamos a utilizar la representación ordinal vista en la Sección 4.2.1. La motivación de esta elección viene dada por la sencillez de esta representación y por ser la única que se ha utilizado en la literatura con unos resultados muy buenos, como por ejemplo en el GA de Reeves (1995).

Para la fase de evaluación de los individuos utilizaremos primero una función de mapeado muy parecida a la vista en la 4.2.2, pero con una ligera modificación tal y como se muestra a continuación:

$$\begin{aligned} f(P_{t(l)}) &= \max\{C(P_{t(1)}), C(P_{t(2)}), \dots, C(P_{t(P_{size})})\} - C(P_{t(l)}) + 1 \\ l &= (1, \dots, P_{size}) \end{aligned}$$

El motivo de sumar 1 en la función de mapeado es para asegurarnos que el peor individuo de la población (aquel con el mayor C_{max}) tendrá un valor de adecuación o “*fitness value*” de 1. Esto evita problemas a la hora de hacer escalados cuando el fitness es cero (divisiones por cero, etc...). Adicionalmente al mapeado, aplicamos una función de escalado en la etapa de evaluación cuando el método de selección lo requiera (los no basados en ranking). Utilizamos el escalado lineal (Goldberg, 1989) a partir del cual, y tras calcular dos parámetros a y b , la función de mapeado final o f_f queda como sigue:

$$\begin{aligned} f_f(P_{t(l)}) &= a \cdot f(P_{t(l)}) + b \\ l &= (1, \dots, P_{size}) \end{aligned}$$

Una conclusión que se obtiene tras examinar la literatura y los algoritmos genéticos existentes para el problema del taller de flujo y problemas afines, como el TSP (ver Sección 3.1.5.3 en el Capítulo 3) es que una inicialización aleatoria de la población requiere de muchas iteraciones del GA para converger y que aún así, los resultados no son buenos. Este hecho se confirma con el estudio ya citado de Jain, Bagchi y Wagneur (2000). De ahí que varios autores como Reeves (1995) o Chen, Vempati y Aljaber (1995) decidan inicializar la población mediante el uso de heurísticas competitivas. Como se ha comentado, mientras Chen, Vempati y Aljaber utilizan una inicialización basada en los $m - 1$ individuos generados con la heurística CDS, Reeves inicializa la población de manera aleatoria menos un individuo que se genera con la heurística NEH. El problema con esta inicialización es que tenemos un “super” individuo en la población cuando los demás, en comparación con éste, van a ser mediocres. Hemos podido comprobar, tras la implementación de este GA de Reeves en el Capítulo 3, como esta inicialización puede provo-

car en algunos casos una convergencia prematura, dado que la población pierde rápidamente diversidad al dominar el “super” individuo el proceso de selección. Una de las soluciones que Reeves propuso fue el esquema de mutación adaptativo que aumenta la probabilidad de mutación cuando la población pierde diversidad. Esta solución, aunque buena, no es capaz de recuperar toda la información que se ha perdido en el proceso genético, provocando que al aumentar la probabilidad de mutación el GA se convierta en una búsqueda local que básicamente examina mutaciones del “super” individuo generado inicialmente.

La solución que se propone aquí es generar varios individuos que sean a la vez buenos y también diferentes para que exista una porción de la población que contenga buenos individuos a la vez que diversos, evitando así posteriores problemas de convergencia prematura o la dominación de un único individuo en el proceso de selección. ¿Cómo generar buenos individuos?, ya vimos en el Capítulo 3 como la heurística de Nawaz, Enscore y Ham (NEH) con la mejoras de Taillard constituye el método heurístico más eficaz, a la par que eficiente, para el taller de flujo de permutación. El problema es que esta heurística es determinista y para el mismo problema genera una única permutación de los trabajos como respuesta, con lo que sólo se puede obtener un individuo para la población. Proponemos una modificación de esta heurística, concretamente en el paso 2 de la misma, para obtener una fuente de muy buenos individuos distintos entre sí. Como ya viéramos en la descripción de la heurística NEH, en el paso 2 se ordenan los trabajos de forma descendiente según el tiempo de proceso acumulado en todas las máquinas, lo que habíamos llamado P_j . Los trabajos ordenados se almacenan en una lista que se denotó por ℓ . En este mismo paso de la heurística se escogían los dos primeros trabajos de la lista ℓ , $\ell_{(1)}$ y $\ell_{(2)}$ y se evaluaban las dos posibles secuencias parciales. Tras esto, nos quedábamos con la mejor secuencia parcial y entrábamos en el paso 3 donde se continuaba eligiendo el tercer trabajo de ℓ , $\ell_{(3)}$. La modificación que proponemos es, una vez generada la lista ordenada ℓ , alterar la ordenación y luego continuar con el resto del método. Los primeros trabajos de ℓ tienen mucho peso sobre la secuencia final, luego si escogemos al azar dos posiciones j y k en ℓ , donde $j \neq k$ y $j, k = (3, \dots, n)$ e intercambiamos los trabajos que ocupan las posiciones 1 y 2 con los trabajos que ocupan las posiciones j y k , obtendremos una secuencia modificada. De esta manera tenemos un total de

$((n - 2) \cdot (n - 3))/2$ posibles modificaciones de la lista ℓ atendiendo a todos los posibles valores de j y de k , lo cual representa un número de secuencias iniciales virtualmente ilimitado, dado que para los problemas de ejemplo más pequeños de Taillard (20 trabajos) tendremos un total de 153 posibles secuencias generadas con la heurística NEH modificada, más la secuencia NEH original lo que da un total de 154 posibles individuos iniciales, un número bastante superior a los tamaños de población que normalmente se manejan.

Con esta heurística NEH modificada, que llamaremos NEH_m , con la heurística NEH original y con un parámetro que denotamos por B_i , aplicamos la inicialización que se puede ver en el Listado 4.4.

```

function Inicializacion(var P: Population);
begin
    //El primer individuo se genera por la heurística NEH estándar con las
    //mejoras de Taillard
    Population[1]:=NEH;
    //Hasta el  $B_i\%$  de los individuos se generan con la heurística NEH
    //modificada
    for l := 2 to round( $B_i \cdot P_{size}$ ) do
        Population[l]:=NEHm;
    //El resto de individuos hasta completar la población se generan al azar
    for m := round( $B_i \cdot P_{size}$ )+1 to Psize do
        Population[m]:=RANDOM;
end;

```

Listado 4.4 – Proceso de inicialización para el GA propuesto en forma de pseudoalgoritmo.

Como podemos observar, el primer individuo de la población se genera con la heurística NEH original, tal y como se hace en el GA de Reeves, después, hasta alcanzar un porcentaje B_i de la población, se utiliza la heurística NEH_m . El resto de individuos hasta completar la población ($100 - B_i\%$) son secuencias obtenidas totalmente al azar. De esta manera introducimos una alta diversidad en la población.

Pese a haber utilizado las mejoras de Taillard, la heurística NEH y la heurística NEH_m , aunque muy rápidas, son considerablemente más lentas que generar todos los individuos al azar, de ahí que se aconseje que el porcentaje B_i de individuos generados de acuerdo a estas heurísticas no sea muy elevado. Además, pudimos comprobar en experimentos iniciales como para porcentajes de B_i superiores al

25 % ($B_i = 0,25$) no se apreciaban mejoras en el algoritmo genético, así que fijaremos, para el GA propuesto, el parámetro B_i a este valor.

4.3.2. Selección

Para el operador de selección nos hemos centrado en los operadores más ampliamente utilizados entre los comentados en la Sección 4.2.3, concretamente en la selección por ruleta, selección por torneo binario y la selección por ranking. Experimentos iniciales mostraron unos malos resultados con el proceso de selección por ruleta, por lo que se descartó para estudios posteriores. Para la selección por torneo se aplica el mapeado lineal especificado en la sección anterior mientras que para la selección por ranking no es necesario. Es interesante aclarar que la selección por ranking requiere de ordenaciones de los individuos de acuerdo a su valor de adecuación. Todas estas ordenaciones se hacen con la versión eficiente del algoritmo QuickSort que se comentó en la Sección 3.2 del Capítulo 3.

4.3.3. Cruce

Inicialmente consideramos los siguientes operadores de cruce que ya se han visto con detalle en la Sección 4.2.4:

- “*Partially Matched Crossover*” (PMX) de Goldberg y Lingle (1985), con la modificación de Cotta y Troya (1998).
- “*Uniform Order Based Crossover*” (UOB) a partir del trabajo de Spears y De Jong (1991) y Syswerda (1996).
- *cruce de un punto por orden* (OP) (ver Michalewicz, 1996).
- “*Order Crossover*” (OX) de Davis (1985a).
- *cruce de dos puntos por orden* (TP) (ver Michalewicz, 1996).

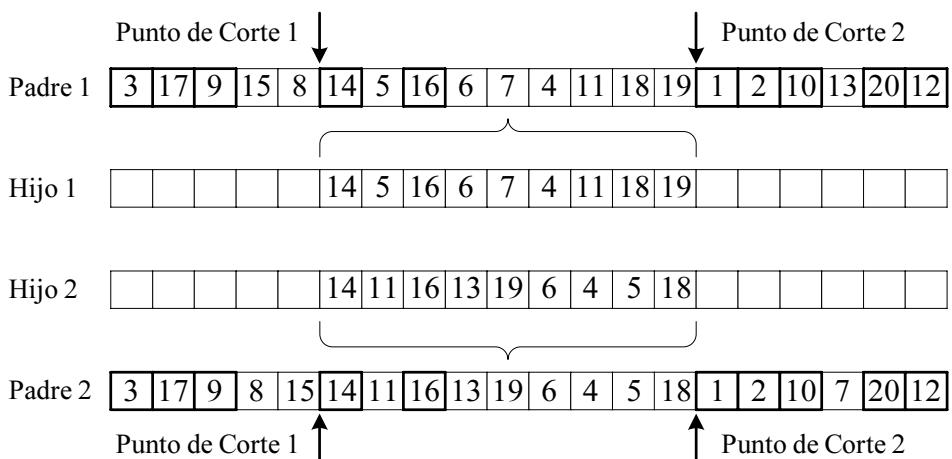
El operador “*Generalized Position Crossover*” (GPX), propuesto por Mattfeld (1996) resultó ser prácticamente equivalente al operador TP en experimentos previos y se decidió excluirlo del estudio.

Tras la implementación de todos estos operadores pudimos comprobar como muchas veces los operadores de cruce generaban hijos con valores de C_{max} inferiores a los de los padres. Más concretamente, se realizó un análisis donde para los cinco tipos de cruce considerados se contabilizaron los siguientes casos: número de veces que ambos hijos eran peores que los padres (A), número de veces que el hijo 1 o el hijo 2 eran peores que ambos padres (B), número de veces que el hijo 1 o el hijo 2 eran mejores que ambos padres (C) y número de veces que ambos hijos resultaban ser mejores que ambos padres (D). Se obtuvieron los porcentajes para cada operador y cada caso. Sorprendentemente, el caso A, para algunos de los tipos de cruce, alcanzaba el 40 %. Es decir, 4 de cada 10 operaciones de cruce resultaban en una descendencia con peores características que los padres, lo que representa que no es un operador adecuado. Para el caso B el porcentaje subía en algunos tipos de operador hasta el 78 %, mientras que el caso C se daba con mucha menos frecuencia, el mejor operador de la comparativa (OP) resultaba tener un 23 % de casos donde al menos alguno de los hijos resultaba ser mejor que los padres mientras que el caso D resultó ser muy raro, el operador OP consiguió apenas un 3 % de casos donde ambos hijos mejoraban a ambos padres.

Un análisis más en profundidad reveló que este comportamiento de los operadores varía conforme el número de generaciones evaluadas en el GA aumenta, más concretamente, en las primeras generaciones del GA, la mayoría de los operadores de cruce se comportan razonablemente bien, mientras que en las últimas generaciones los operadores generan hijos que son consistentemente peores que los padres en la mayoría de las ocasiones.

La explicación para este comportamiento es la siguiente: al iniciar el proceso genético hay mucha diversidad en la población, y la media del C_{max} es alta. En esta situación, los operadores de cruce combinan buenos individuos (los obtenidos tras el proceso de selección) y generalmente, es fácil que los hijos generados tengan un C_{max} menor. A medida que el GA avanza, la diversidad en la población disminuye y cada vez los individuos de la población se “parecen” más entre sí. El parecido entre los individuos puede explicarse con la idea del camino crítico de la secuencia, que está formado por aquellas tareas entre las que no hay tiempos de espera ni tiempos ociosos desde el inicio del proceso de la primera tarea en la primera máquina hasta la finalización de la última tarea en la última máquina.

La idea del camino crítico en problemas de programación de la producción y de cómo explotar la información contenida en este camino se debe a Grabowski (ver Grabowski, 1980, Grabowski, Skubalska y Smutnicki, 1983, Grabowski, Nowicki y Zdrzałka, 1986 y también a Nowicki y Smutnicki, 1996). Normalmente, cuando se han evaluado varias generaciones en el GA, las soluciones contendrán partes comunes del camino crítico, es decir, agrupaciones o “bloques” de trabajos que aparecen contiguos en los individuos y en la misma posición dentro de la secuencia. Estas partes del camino crítico son en buena medida las responsables de que los individuos tengan un C_{max} bajo y por tanto hayan sobrevivido al proceso genético. El problema con los operadores considerados es que cuando estos bloques de trabajos son numerosos, es muy fácil que un punto de corte en el cruce caiga justo dentro de un bloque, partiéndolo y generando hijos donde no se ha respetado este “bloque constructivo” responsable de la alta “calidad” de los padres, generando un hijo con un mayor C_{max} . Pudimos comprobar este extremo analizando los padres tras varias generaciones en el GA. Comúnmente, los alelos de los padres coincidían en varias posiciones contiguas, mientras que en otras no, los cruces normalmente no respetaban estas posiciones contiguas, generando hijos con una menor calidad. Si utilizamos el cruce OX, un ejemplo de esta situación puede verse en la Figura 4.17, donde se resuelve el problema ta001 de Taillard.



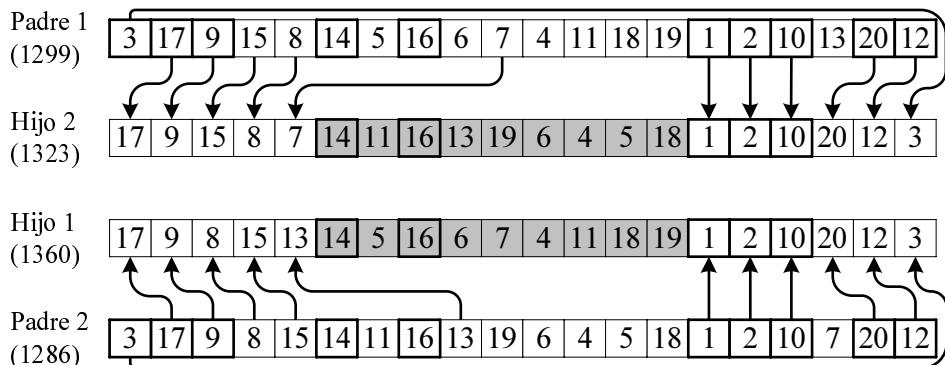
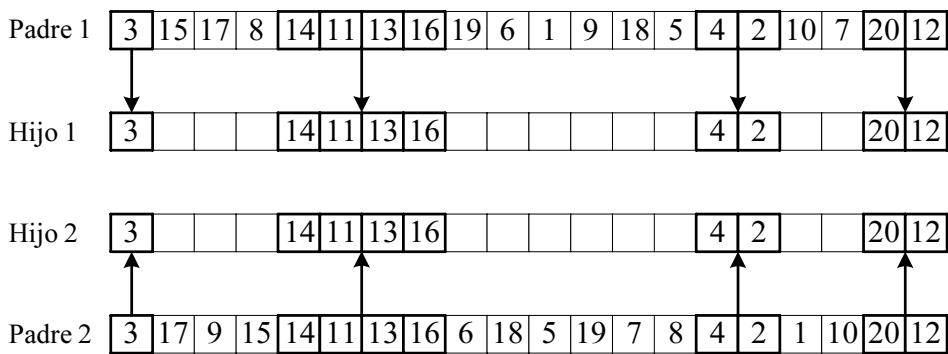


Figura 4.17 – Destrucción de los bloques en el operador de cruce “Order Crossover” o OX.

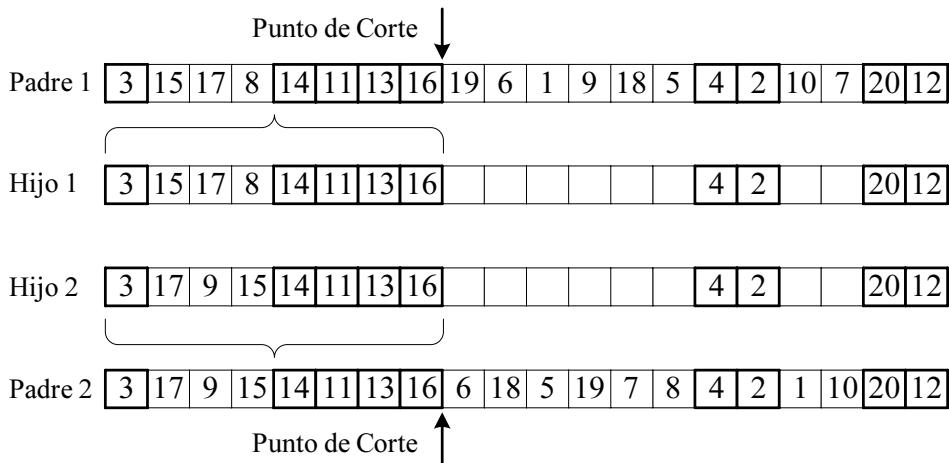
En el ejemplo, se han marcado en trazo grueso aquellas posiciones cuyos alelos coinciden tanto en el padre 1 como en el padre 2. Como vemos hay cinco grupos de posiciones, con 10 trabajos en total, que coinciden en ambos padres. Tras la operación de cruce, sólo tres de estos cinco bloques de trabajos se han respetado en los hijos. El efecto de esta destrucción de bloques puede verse en el valor del C_{max} que aparece en la citada figura. Mientras los padres tienen un C_{max} de 1299 y 1286 respectivamente (teniendo en cuenta que la solución óptima para este ejemplo es de 1278, estos dos individuos tienen un IPSO de 1,64 % y de 0,63 %), los hijos terminan con un C_{max} de 1360 y 1323, o lo que es lo mismo, un IPSO de 6,42 % y 3,52 % respectivamente. Como se puede observar, en este ejemplo el operador de cruce OX ha generado hijos muy alejados de los valores de C_{max} de los padres. Todo este razonamiento sigue las ideas del *Teorema de Esquemas* de Holland (1975) y la hipótesis de los *Bloques Constructivos* de Goldberg (1989). Concretamente, el cruce OX provoca la desaparición de esquemas, aunque éstos presenten una longitud definida y orden pequeño.

Proponemos cuatro nuevos operadores de cruce que tratan de resolver este problema. La idea es identificar los bloques constructivos o conjuntos de trabajos consecutivos en ambos padres y respetarlos tras el cruce, es decir, que los hijos contengan los mismos bloques que los padres.

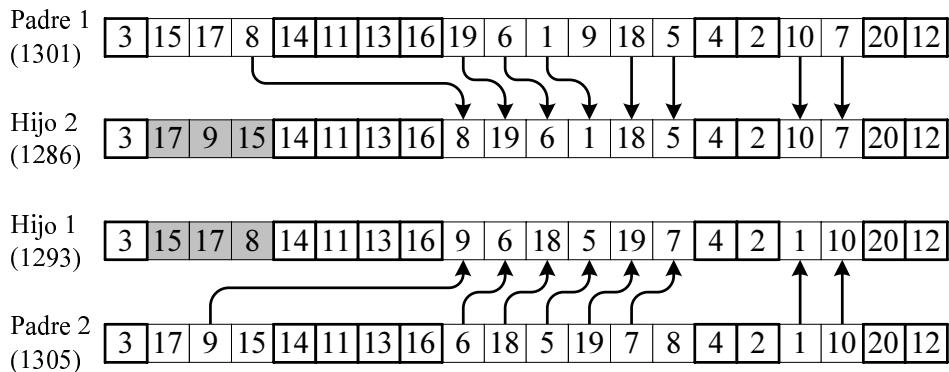
El primer tipo de cruce lo llamamos “*Similar Job Order Crossover*” o SJOX, y el funcionamiento es el siguiente: primero se recorren todas las posiciones de ambos padres y se examinan sus correspondientes alelos. Los hijos heredan aquellas posiciones que contengan alelos idénticos en ambos padres (Figura 4.18(a)). Después se obtiene un punto de corte al azar y los hijos heredan todas las posiciones hasta el punto de corte del padre directo (Figura 4.18(b)). Por último, los hijos heredan las posiciones faltantes del otro parente, en el orden relativo en que aparecen. (Figura 4.18(c)).



(a) Primero, las posiciones con idénticos alelos en los dos padres se pasan directamente a los hijos.



(b) Despu  s se hereda la parte anterior al punto de corte del padre directo.



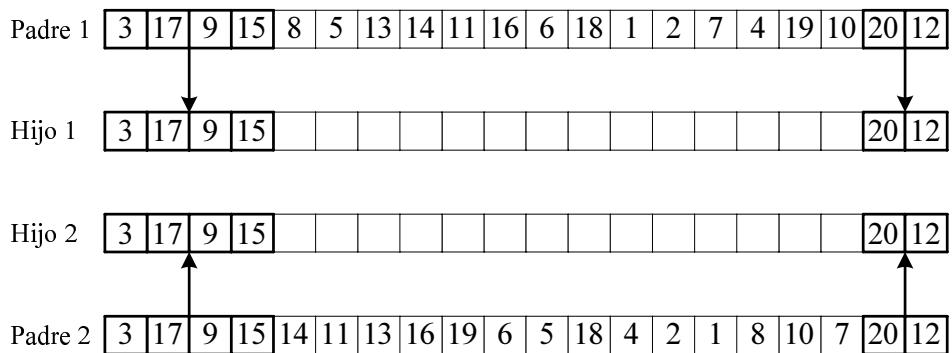
(c) Por   ltimo, se heredan las posiciones faltantes del otro padre en su orden relativo.

Figura 4.18 – Operador de cruce “Similar Job Order Cross-over” (SJOX).

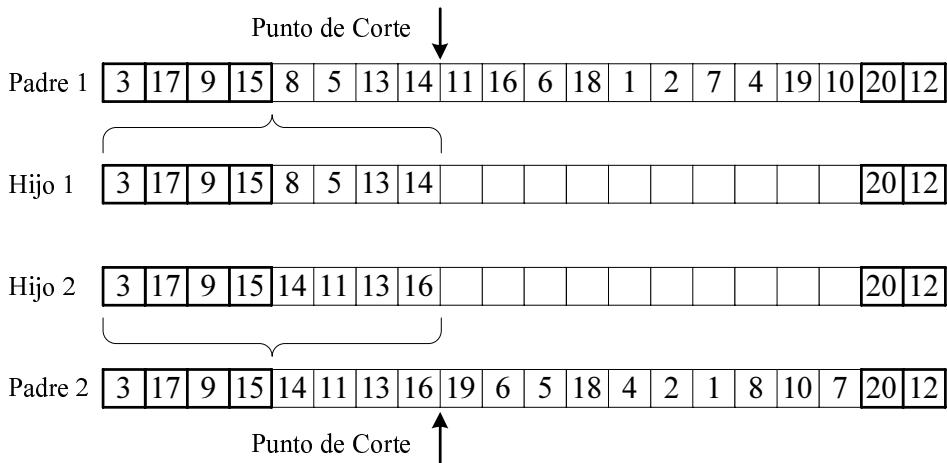
Como vemos, los bloques se respetan en los hijos, y para este ejemplo de cruce, obtenido mientras se resolv  a el problema ta001 de Taillard, los hijos generados tienen un C_{max} menor que los padres. Esto es porque los buenos bloques de los padres se han respetado y el resto de posiciones siguen la estrategia del

cruce de un punto por orden, que es un cruce que funciona bien y da buenos resultados. De hecho, si no existen posiciones con alelos idénticos en ambos padres, el operador SJOX se comporta exactamente igual al operador OP. Esto es una ventaja importante dado que permite aprovechar la potencia del operador OP en las primeras etapas del GA y al mismo tiempo evita los problemas de rotura de bloques en las últimas etapas del algoritmo.

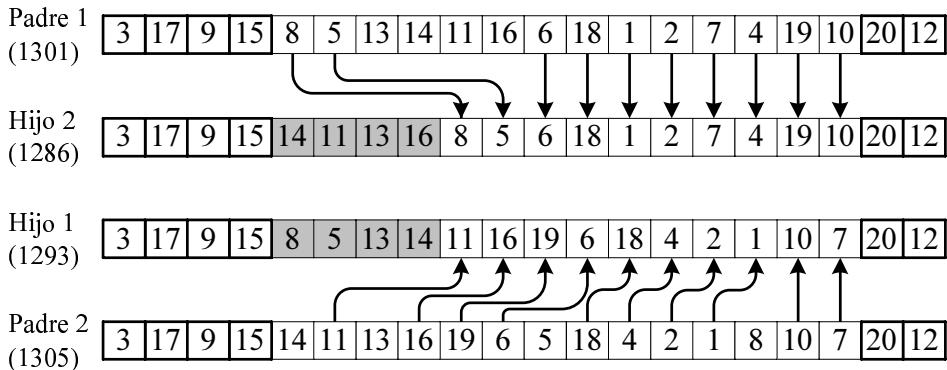
Un examen más detenido del operador SJOX nos lleva a la siguiente cuestión: puede haber situaciones donde tenemos varias posiciones no consecutivas con alelos idénticos en ambos padres, no está claro que copiar estas posiciones idénticas alternativas sea recomendable ya que no constituyen un bloque. Estas coincidencias pueden ser debido simplemente al azar. Para dar solución a este problema se plantea el operador “*Similar Block Order Crossover*” o SBOX. En este nuevo operador, los hijos solamente heredan bloques de trabajos idénticos en ambos padres, entendiendo por bloque la aparición de al menos dos posiciones consecutivas con idénticos alelos en ambos padres. De esta manera sólo se modifica el primer paso del cruce SJOX. Un ejemplo, aplicado también al problema ta001 de Taillard se muestra en la Figura 4.19



- (a) Primero, los bloques de al menos dos trabajos con idénticos alelos en los dos padres se pasan directamente a los hijos.



(b) En segundo lugar se hereda la parte anterior al punto de corte del padre directo.



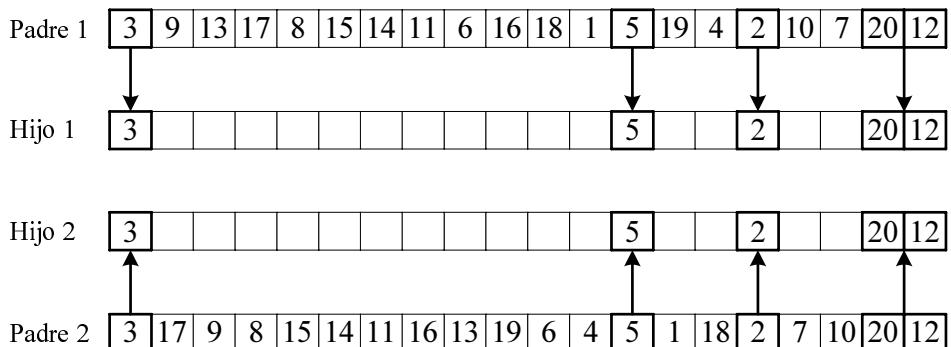
(c) Finalmente se heredan las posiciones faltantes del otro parente en su orden relativo.

Figura 4.19 – Operador de cruce “Similar Block Order Crossover” (SBOX).

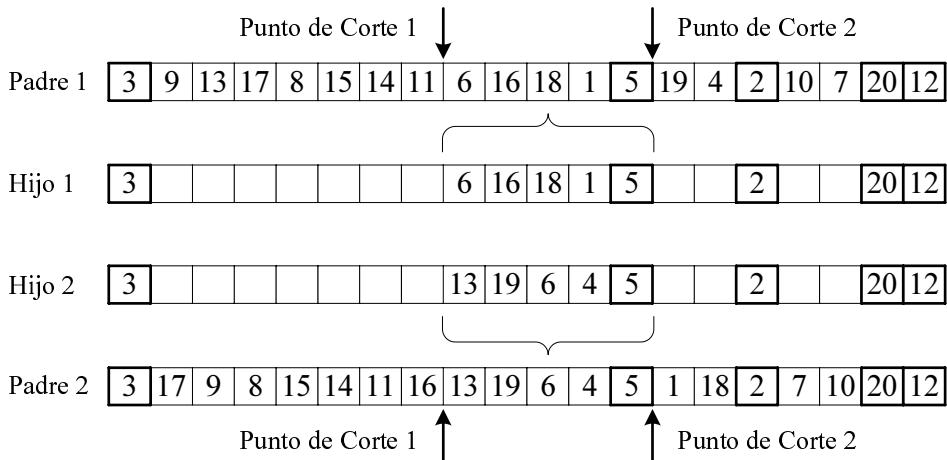
A partir del ejemplo vemos como en el primer paso (Figura 4.19(a)) las posiciones 7, 12 y 14 contienen el mismo alelo en ambos padres (13, 18 y 2 respectivamente), aún así estas posiciones no se han copiado al ser simplemente ocurrencias no consecutivas y no constituir un bloque. Se podría argumentar que

en algunos casos y para un valor de n superior (por ejemplo 100), dos posiciones consecutivas con idénticos alelos en ambos padres tampoco se deberían considerar como un bloque. Se hicieron algunas simulaciones para determinar si un operador SBOX que considerase bloques a partir de tres trabajos era mejor que el SBOX aquí considerado. El resultado no mejoraba las prestaciones del operador, así que consideraremos ocurrencias de dos trabajos consecutivos como un bloque.

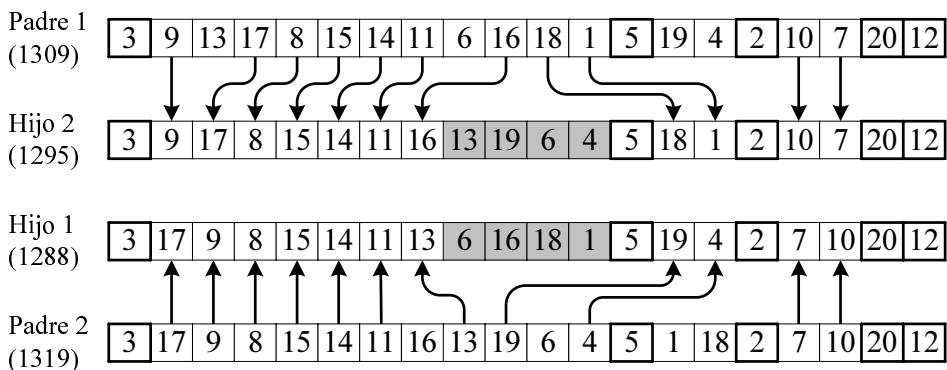
Los otros dos operadores de cruce que proponemos son modificaciones al operador SJOX y SBOX donde se consideran dos puntos de cruce en vez de uno. El operador SJ2OX o “*Similar Job 2-Point Order Crossover*” se diferencia del operador SJOX en el segundo paso, donde se extraen dos puntos de cruce y la sección entre ambos se copia a los hijos desde el padre directo. El resto de pasos son idénticos al operador SJOX. Un ejemplo de este operador puede verse en la Figura 4.20



(a) Las posiciones con idénticos alelos en los dos padres se pasan directamente a los hijos.



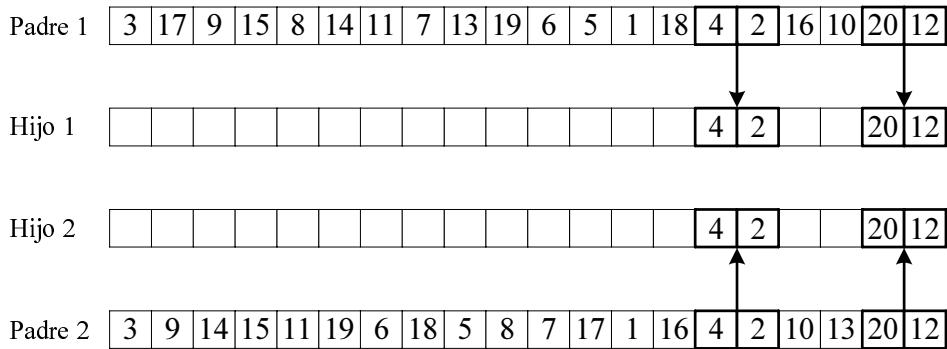
(b) Se hereda la parte que queda entre los dos puntos de cruce del padre directo.



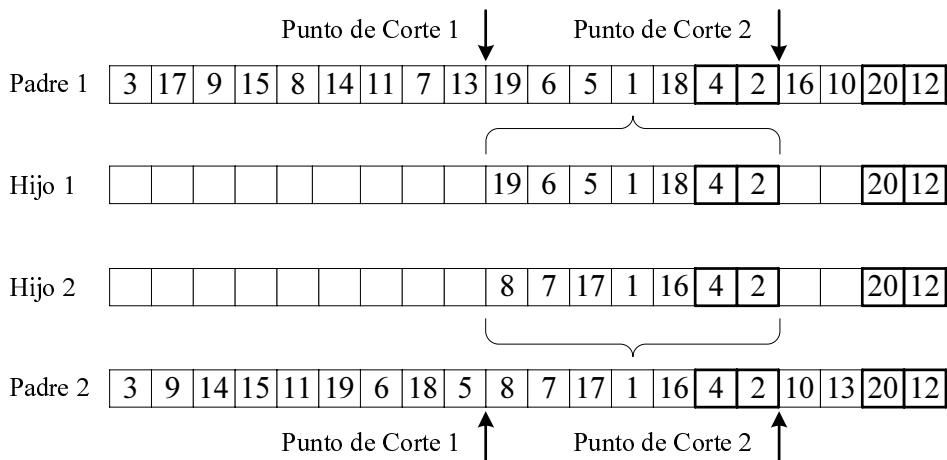
(c) Se heredan las posiciones faltantes del otro parent en su orden relativo.

Figura 4.20 – Operador de cruce “Similar Job 2-Point Order Crossover” (SJ2OX).

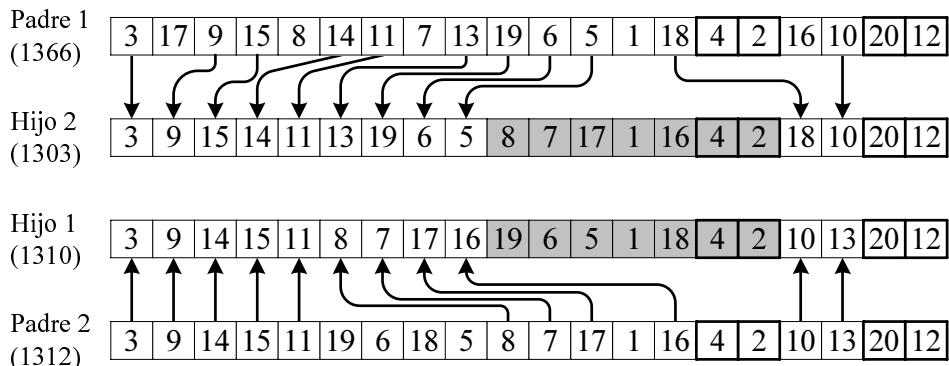
Al último operador de cruce considerado lo llamamos “*Similar Block 2-Point Order Crossover*” (SB2OX) y junta las ideas de los operadores SBOX y SJ2OX. Es decir, se trata del operador SBOX pero considerando dos puntos de cruce en vez de uno. La Figura 4.21 muestra un ejemplo de este último operador propuesto.



(a) Los bloques con al menos dos posiciones de alelos idénticos en ambos padres se copian directamente a los hijos.



(b) Se hereda la parte que queda entre los dos puntos de cruce del parente directo.



(c) Se heredan las posiciones faltantes del otro parente en su orden relativo.

Figura 4.21 – Operador de cruce “*Similar Block 2-Point Order Crossover*” (SB2OX).

Estos cuatro nuevos operadores de cruce (SJOX, SBOX, SJ2OX y SB2OX) se basan en la identificación de los bloques constructivos y se comportan como los operadores de cruce de un punto por orden (OP) y dos puntos por orden (TP) cuando no existen bloques o posiciones con alelos idénticos en ambos padres.

4.3.4. Mutación y operador de reinicialización

En el algoritmo genético propuesto se consideraron inicialmente los tres operadores de mutación comentados anteriormente (ver Sección 4.2.5). Los resultados de pruebas preliminares dieron una clara ventaja a la mutación por desplazamiento o “*SHIFT mutation*” sobre las otras dos. La diferencia resultó ser tan clara que decidimos utilizar sólo este tipo de mutación. Reeves por un lado y Murata, Ishibuchi y Tanaka por otro llegaron también a la misma conclusión con respecto a este operador de mutación. Además, como se ha comentado, este operador está íntimamente relacionado con el vecindario *I* visto en el capítulo anterior, donde en varios trabajos se observó que proporciona mejores resultados que el vecindario *E*, en el que están basados los otros dos tipos de operador de mutación (SWAP y POSITION).

El algoritmo genético propuesto incorpora un nuevo operador al que llamamos

de reinicialización. En un buen algoritmo genético es de esperar que la población converja hacia soluciones quasi óptimas en el transcurso de las generaciones. El problema es que esa convergencia puede producirse hacia óptimos locales. En principio, sería fácil escapar de un óptimo local mediante el operador de mutación, pero es posible que todos los vecinos I del óptimo local sean peores que éste, por lo que sólo la mutación no sirve. El cruce tampoco sirve de mucho si todos los individuos son muy parecidos entre sí. Si esta situación llega a producirse el GA se “estancará” y no se producirán mejoras en la población, hecho que se reflejará cuando el mejor C_{max} de la población no varíe. En la construcción del GA pudimos comprobar que este problema se daba con frecuencia. La Figura 4.22 muestra una gráfica de como evoluciona el mejor C_{max} de la población (trazo rojo) y el C_{max} medio de la población o \bar{C}_{max} (trazo verde) conforme avanza el número de generaciones evaluadas.

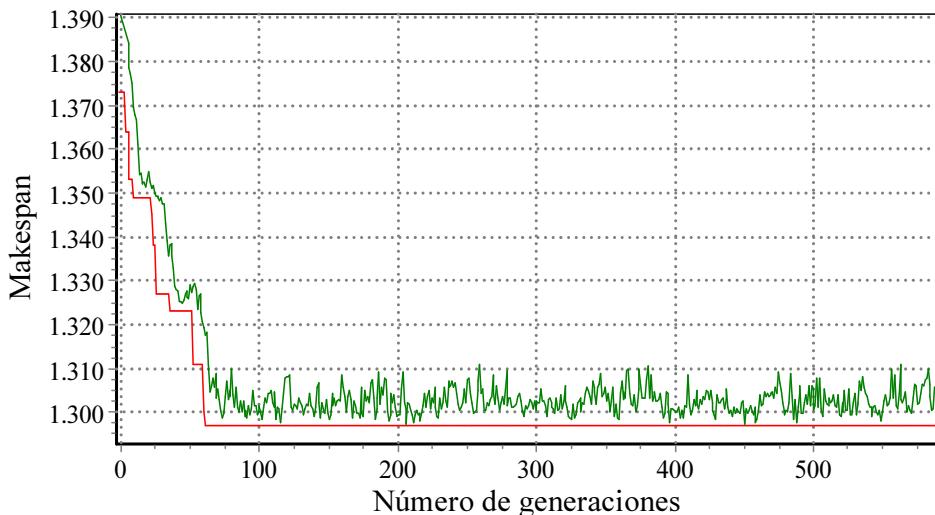


Figura 4.22 – Evolución del mejor C_{max} (trazo rojo) y del C_{max} medio (trazo verde) de la población frente al número de generaciones. Problema ta001 de Taillard.

La solución óptima del problema ta001 de Taillard es 1278. El GA ha evaluado un total de 589 generaciones con una población de 50 individuos cada una. La

mejor solución alcanzada es de 1297, lo cual no se aleja demasiado de la solución óptima (un 1,48 %). El problema es que parece ser que esta solución es un óptimo local muy fuerte, dado que el GA no ha sido capaz de mejorar esta solución a partir de la generación 61. Esto quiere decir que el GA ha permanecido estancado casi un 90 % del tiempo.

En Alcaraz, Maroto y Ruiz (2003) se desarrolló un operador de reemplazo que buscaba minimizar este problema utilizando dos probabilidades, llamadas $P_{replacement}$ y $P_{exchange}$. Este operador determina, de acuerdo a la probabilidad $P_{replacement}$, si se lleva a cabo el procedimiento de reemplazo. En caso afirmativo, la probabilidad $P_{exchange}$ determina el ratio de la población que se intercambia por individuos nuevos generados aleatoriamente. El operador de reinicialización que se propone en esta Tesis Doctoral utiliza una aproximación distinta, demostrando, como veremos, un buen comportamiento. El funcionamiento es el siguiente:

1. En cada generación g , almacenar el mínimo C_{max} : C_{max_g} .
2. Si $C_{max_g} = C_{max_{g-1}}$ entonces incrementamos el contador de generaciones sin mejora en el C_{max} : $countmak = countmak + 1$. En cambio, si $C_{max_g} < C_{max_{g-1}}$, hacemos $countmak = 0$.
3. Si $countmak > G_r$ entonces aplicamos el procedimiento de reinicialización siguiente:
 - Ordenar la población ascendentemente por el C_{max} de cada individuo.
 - Saltar el primer 20 % de los individuos de la lista ordenada (los mejores individuos).
 - Del 80 % de individuos restantes, el primer 50 % se reemplazan por simples mutaciones de desplazamiento (mutación SHIFT) de individuos extraídos del primer 20 % (mutaciones de los mejores). El 25 % se reemplazan por individuos nuevos generados a partir de la heurística NEH modificada (NEH_m) vista en la Sección 4.3.1 y el 25 % restante se reemplaza con individuos generados totalmente al azar.
 - $countmak = 0$.

De forma resumida, el operador reemplaza los 80 % peores individuos de la población mediante mutaciones de los 20 % mejores, y nuevos individuos, generados a partir de la heurística NEH_m y aleatoriamente, y esto siempre que el GA lleve un número de generaciones superior a G_r sin mejorar el mínimo C_{max} . G_r es un parámetro de control que habrá que calibrar. El hecho de no reemplazar los mejores individuos es simplemente para no perder la buena información que se ha obtenido actualmente en el proceso genético. Se introduce una buena porción de mutaciones de los mejores individuos, buscando escapar de los óptimos locales. También se introducen nuevos individuos, para fomentar el que todas las regiones del espacio de búsqueda se puedan explorar. El funcionamiento de este operador resultó ser muy bueno. A continuación (Figura 4.23) se muestra la gráfica de la evolución del mismo GA que en la Figura 4.22 pero añadiendo el operador de reinicialización.

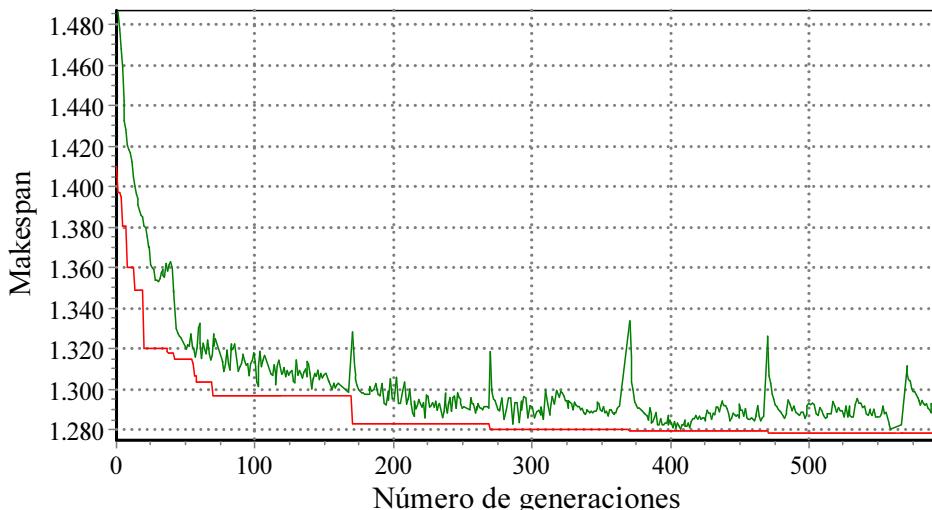


Figura 4.23 – Evolución del mejor C_{max} (trazo rojo) y del C_{max} medio (trazo verde) de la población frente al número de generaciones aplicando operador de reinicialización. Problema ta001 de Taillard.

Podemos observar que, igual que pasaba en el caso anterior, en torno a la generación 65 se alcanza una solución con un C_{max} de 1297. En este caso el parámetro G_r se ha fijado a 100, y dado que el GA no evoluciona durante 100 generaciones vemos como aproximadamente en la generación 175 se produce un pico en el \bar{C}_{max} que indica que el proceso de reinicialización se ha efectuado y al introducir individuos aleatorios en la población el \bar{C}_{max} aumenta. El efecto de la aplicación del operador es inmediato, podemos ver como el mínimo C_{max} baja de 1297 hasta 1286. En este momento el GA vuelve a estancarse durante otras 100 generaciones hasta que en la generación 275 se vuelve a aplicar el operador de reinicialización e inmediatamente el mínimo C_{max} baja hasta 1280. El operador vuelve a aplicarse en las generaciones 375 y 475 donde se alcanza la solución óptima de 1278. Claramente, este operador aumenta la eficacia del GA.

4.3.5. Esquema generacional

Como esquema generacional entendemos al proceso mediante el cual los nuevos individuos reemplazan a los antiguos en la población. Como ya se vio en el capítulo anterior, por norma general, los hijos generados tras el cruce (y posterior mutación) reemplazan directamente a los padres que se seleccionaron para dicho cruce. Reeves (1995) utilizó el esquema generacional tipo “*steady state*” donde los hijos reemplazan no a los padres, sino a un individuo escogido al azar de entre aquéllos que tienen un valor de adecuación inferior a la media de la población. De igual manera Reeves y Yamada (1998) utilizan un GA con el esquema “*steady state*”, pero en este caso los hijos generados reemplazan a los dos peores individuos de la población. Se ha demostrado en varios estudios (ver Davis, 1996 y Vavak y Fogarty, 1996) que en algunos problemas los GA tipo “*steady state*” funcionan mejor que los GA estándar. No obstante, existe un problema con este tipo de algoritmos que puede llegar a ser importante. Si la presión del operador de selección es alta, normalmente los mejores individuos se seleccionarán a menudo. Si además tenemos en cuenta que en muchas ocasiones los hijos generados serán muy parecidos a los padres (por no haberse llevado a cabo el cruce, o que habiéndose realizado los cambios sean pequeños), ocurrirá que, en muy pocas generaciones, podemos tener una población prácticamente

homogénea, dándose el problema de la convergencia prematura. Como se ha comentado, Reeves utilizó el esquema de mutación adaptativa para mitigar esta situación. También, Reeves y Yamada, propusieron descartar aquellos hijos con un valor del C_{max} que ya estuviese en la población buscando evitar que existan varias copias del mismo individuo en la población.

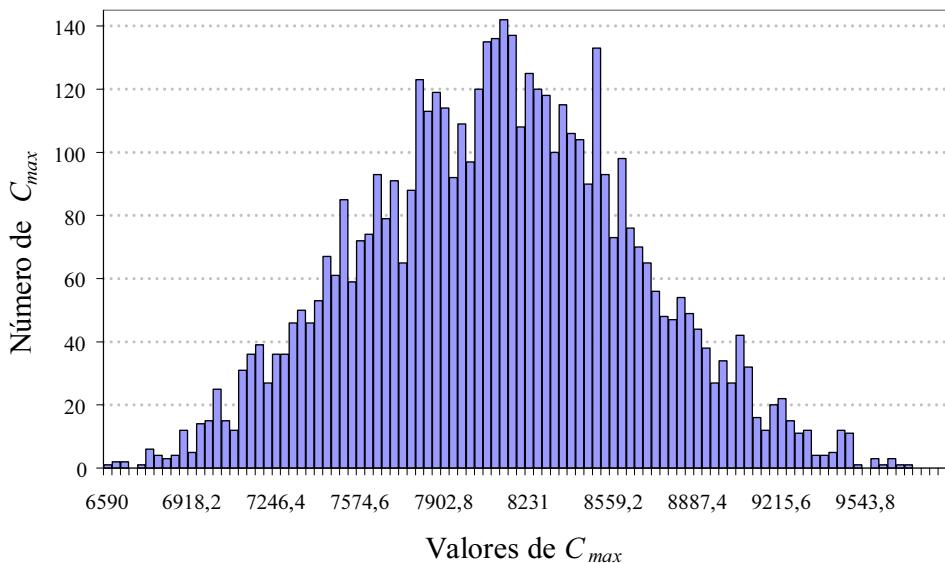
Aunque estas dos soluciones resuelven en buena medida el problema, sobre todo la solución de Reeves y Yamada, es posible afinar un poco más teniendo en cuenta que en general pueden existir muchas soluciones con idéntico valor de C_{max} y diferente secuencia. Por ejemplo, dos soluciones pueden coincidir en los bloques de trabajos que constituyen el camino crítico y no coincidir en el resto de posiciones. Si se da este caso es muy probable que ambas soluciones tengan el mismo valor de C_{max} . Para ilustrar este extremo, hemos realizado un extenso estudio con los ocho problemas de Carlier (car1-car8) obtenidos de la librería de problemas de investigación operativa OR-Library accesible en <http://mscmga.ms.ic.ac.uk/jeb/orlib>. Para cada problema hemos generado todas las posibles secuencias de permutación ($n!$) y hemos calculado los correspondientes valores del C_{max} . Todos estos valores se han almacenado y agrupado. La elección de este conjunto de problemas viene motivada por el hecho de ser el único conjunto completo de pruebas para el que es factible generar y resolver todas las posibles soluciones. La Tabla 4.1 muestra, para los ocho problemas de Carlier, el número total de posibles soluciones, el número de C_{max} distintos y el número de C_{max} óptimos, entre otros datos.

Problema	n	m	Soluciones posibles ($n!$)	Valores de C_{max} distintos ($T_{C_{max}}$)	Ratio $\left[\frac{n!}{T_{C_{max}}}\right]$	número de soluciones óptimas
car7	7	7	5.040	1.693	3	1
car8	8	8	40.320	1.996	20	1
car6	8	9	40.320	2.873	14	1
car5	10	6	3.628.800	4.119	881	3
car1	11	5	39.916.800	4.150	9.619	8.106
car3	12	5	479.001.600	4.667	102.636	18
car2	13	4	6.227.020.800	4.562	1.364.976	9.690
car4 ¹	14	4	87.178.291.200	5.030	17.331.668	561.256

Tabla 4.1 – Estudio de las soluciones y valores de C_{max} para los problemas de Carlier (car1-car8).

Por comodidad, se han ordenado los problemas por orden creciente del número de trabajos (n). Lo primero y más importante es notar que el número de valores de C_{max} distintos, lo que hemos denotado por $T_{C_{max}}$, es muy inferior al número de soluciones posibles. De esta manera, sacando el ratio $\frac{n!}{T_{C_{max}}}$ sabemos, de media, cuántas soluciones distintas nos podemos encontrar con el mismo valor de C_{max} . Resulta interesante observar que conforme aumenta el valor de n en los problemas, aumenta considerablemente este ratio, mientras que el número de C_{max} distintos aumenta, aunque mucho más despacio. El extremo se da en el problema Car4, de 14 trabajos, donde tenemos tan solo 5.030 distintos valores de C_{max} para más de 87.178 millones de posibles soluciones. De esta manera tenemos que en cada valor distinto de C_{max} hay de media más de 17 millones de posibles secuencias. Realmente, esto es sólo una media, la distribución de los valores de C_{max} y el número de secuencias que contiene cada uno se muestran en la Figura 4.24 en forma de histograma para el problema car7 de Carlier.

¹Se necesitaron más de 15 días de tiempo de CPU para resolver este problema.



viduos de la población para ver si realmente es distinto es una tarea prohibitiva desde el punto de vista computacional. Para ahorrar trabajo lo que proponemos es buscar idénticos valores de C_{max} en la población y si se encuentra alguno entonces comparar posición a posición. Esto se hace así porque sí es posible, como se ha visto, que dos individuos distintos tengan igual valor de C_{max} , pero dos individuos iguales no pueden tener distinto valor de C_{max} . De igual manera, si comparamos posición a posición, en el instante que encontramos una posición con distintos alelos podemos interrumpir la búsqueda, de esta manera, de media sólo tendremos que hacer $\mathcal{O}(n/2)$ comparaciones en el peor caso posible en que dos individuos solamente se diferencien en una posición.

Para aumentar la presión del proceso genético (dado que permitimos muchas más soluciones), no nos contentamos con hacer que los hijos reemplacen a los peores individuos de la población, sino que además imponemos la restricción de que sean mejores que éstos o en otro caso se descartarán. La Figura 4.25 muestra un sencillo diagrama de flujo que determina si un nuevo individuo (hijo) entra o no en la población.

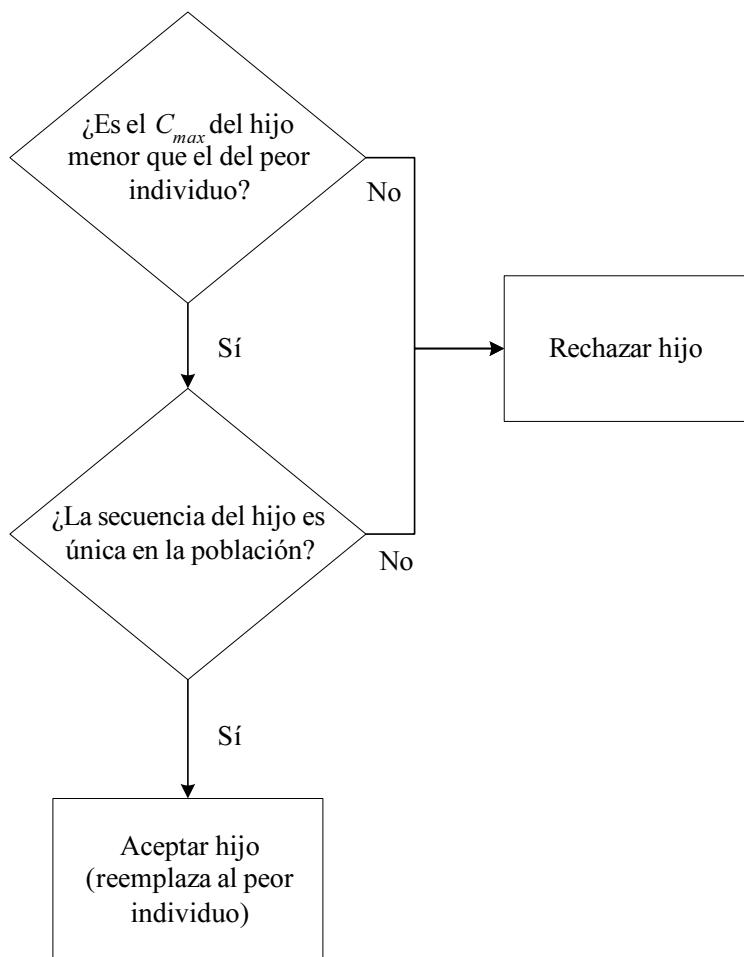


Figura 4.25 – Diagrama de flujo con el esquema generacional propuesto.

4.3.6. Evaluación experimental

Es bien conocido que la eficacia de un GA depende mucho del tipo de representación, de cómo se realiza la selección y de los operadores de cruce y de mutación, así como de las probabilidades con las que estos operadores se aplican (ver Goldberg, 1989, Michalewicz, 1996, Alcaraz, 2001 y Alcaraz y Maroto,

2001). Aun así, muchos autores deciden todos estos operadores y parámetros haciendo “simulaciones” donde normalmente se evalúa un factor cada vez. Este sistema se utiliza por ejemplo en Reeves (1995) o en Chen, Vempati y Aljaber (1995) y en Murata, Ishibuchi y Tanaka (1996a). Otros autores, como por ejemplo Ponnambalam, Aravindan y Chandrasekaran (2001) directamente fijan los parámetros y operadores a un determinado nivel sin dar más detalles.

El problema de la parametrización y calibración de los GAs ha sido ampliamente estudiado, ya en 1985, De Jong determinó que un área de estudio importante en los GAs era (y es) precisamente la calibración de los parámetros. Algunos autores han propuesto esquemas adaptativos que eliminan la necesidad de determinar el valor de algún parámetro o la decisión por uno u otro operador. Por ejemplo, Baker (1985) propone un esquema de selección adaptativo, lo que elimina la necesidad de decantarse por un tipo u otro de operador de selección. Fogarty (1989) realizó un estudio donde la probabilidad de mutación es adaptativa y por tanto no es necesario fijarla de antemano. El autor profundizó este estudio en Smith y Fogarty (1996) en el caso de un GA de tipo “*steady state*”. En el trabajo de Hong y Wang (1998) se elimina la necesidad de fijar el operador de cruce y la probabilidad de cruce, dado que el propio GA va ajustando dinámicamente las probabilidades de dos tipos de operadores de cruce que se aplican simultáneamente. Ochoa, Harvey y Buxton (1999) estudian la relación entre las probabilidades de cruce y de mutación, obteniendo algunos resultados interesantes. Otros autores van incluso más lejos y codifican las probabilidades de mutación y de cruce en el propio genoma del individuo (ver Tuson y Ross, 1996).

Los problemas con estas soluciones son muchos, primero y más importante, en todos los trabajos citados anteriormente, los GAs obtenidos tienen menos parámetros, pero nunca se consigue un GA “sin parámetros” con lo que el problema no se elimina del todo. Segundo, los GAs resultantes son menos eficaces, (en algunos casos mucho más ineficaces) que las versiones con más parámetros cuando éstas se calibran correctamente. Tercero, los GAs adaptativos son complejos, dado que no sólo tienen que controlar la evolución de los individuos sino también la aplicación de los esquemas adaptativos, y esto hace que consuman mucho más tiempo del requerido y finalmente, en cuarto lugar, tenemos que no es fácil diseñar un esquema adaptativo que se comporte bien ante cualquier entrada de datos para

un problema dado y menos para cualquier problema en general. Debido a todo esto los GAs más comunes y los más eficaces no son adaptativos y siguen necesitando de una correcta parametrización de los operadores y probabilidades.

Pocos autores han estudiado la forma de llevar a cabo esta parametrización. En Alcaraz (2001) se utiliza el test no paramétrico de los rangos con signo de Wilcoxon para estudiar los niveles de los factores así como algunas interacciones. Este tipo de tests son muy utilizados en la práctica cuando no es posible asumir la hipótesis de normalidad en los datos.

En esta Tesis Doctoral haremos uso extensivo de un tipo de tests paramétricos mucho más potentes: el análisis de la varianza o ANOVA, enmarcado dentro del diseño de experimentos. De manera general, el diseño de experimentos o “*Design of Experiments*” (DOE) es una metodología organizada y estructurada que permite obtener el comportamiento y la relación entre los factores e interacciones que afectan a una variable en un proceso (ver Montgomery, 2000). La aplicación del DOE para la parametrización de los GAs no es nueva, Bagchi y Deb (1996) aplicaron por primera vez el diseño experimental a la calibración de algoritmos genéticos y posteriormente, en Jain, Bagchi y Wagneur (2000) se aplicó un sencillo experimento factorial completo para la parametrización de GAs aplicados al problema del taller de flujo. No obstante, los autores parecen obviar un aspecto muy importante del ANOVA, que es, como se ha comentado, un test *paramétrico*, y la veracidad de los resultados depende fuertemente del cumplimiento de tres hipótesis básicas que afectan a los datos que son: normalidad, homocedasticidad (igualdad de varianza) e independencia del residuo. Aparte de no comprobar estas hipótesis, los autores realizan un plan factorial muy simple (plan 2^3) de ocho experiencias, con lo que en realidad es muy poca la información que del experimento se puede obtener.

Dado que el GA propuesto es muy rápido (como se verá más adelante), podemos realizar un elevado número de experiencias en un tiempo relativamente corto. Concretamente, para el GA propuesto tenemos que determinar qué tipo de selección se utiliza, el operador de cruce y la probabilidad con la que se aplica, la probabilidad de mutación, el tamaño de la población y el número de generaciones G_r sin mejora en el C_{max} tras las cuales se aplica el procedimiento de reinicialización. Cabe recordar, que existen otros parámetros que ya se han fijado

de antemano, como el parámetro B_i utilizado en la inicialización de la población y el tipo de mutación por desplazamiento. Para el resto de factores evaluaremos los siguientes niveles o variantes:

- Tipo de operador de selección, 2 variantes: Torneo y Ranking.
- Operador de cruce, 9 variantes: OP, OX, PMX, SB2OX, SBOX, SJ2OX, SJOX, TP y UOB.
- Probabilidad de cruce (P_c), 6 niveles: 0, 0,1, 0,2, 0,3, 0,4 y 0,5.
- Probabilidad de mutación (P_m), 5 niveles: 0, 0,005, 0,01, 0,015 y 0,02.
- Tamaño de la población (P_{size}), 4 niveles: 20, 30, 40 y 50.
- Generaciones antes de reinicialización (G_r), 3 niveles: 25, 50 y 75.

Todas las combinaciones de los niveles y variantes de los factores anteriores resultan en un total de $2 \cdot 9 \cdot 6 \cdot 5 \cdot 4 \cdot 3 = 6.480$ diferentes combinaciones y por tanto 6.480 algoritmos genéticos distintos. Para analizar estos factores realizamos un diseño factorial completo. Para cada algoritmo, se resuelven los 120 problemas de Taillard (1993) con el criterio de parada fijado a 10.000 evaluaciones del C_{max} . Se escogió este número de evaluaciones por ser un buen compromiso entre la velocidad de cálculo y precisión, dado que normalmente el GA ya ha alcanzado la convergencia para cuando se alcanzan este número de evaluaciones. La variable respuesta en el experimento es la media del incremento porcentual sobre la solución óptima o menor cota superior conocida para los 120 problemas de Taillard (1993), o lo que es lo mismo, el *IPSON* Total. Para aumentar la precisión del experimento, y dado que en cada ejecución del GA es posible obtener resultados distintos (convergencia hacia distintos óptimos locales), hemos realizado dos réplicas del diseño factorial, con lo que tenemos un total de 3 ejecuciones distintas de todos los algoritmos. Con esto habremos resuelto $6.480 \cdot 3 \cdot 120 = 2.332.800$ problemas, evaluando un total de $2.332.800 \cdot 10.000 = 23.328.000.000$ secuencias. Con este amplio experimento esperamos obtener una buena calibración de los parámetros del GA.

El procedimiento para obtener los resultados se describe a continuación: se ha

utilizado un cluster de 4 ordenadores tipo PC/AT con procesadores AMD Athlon 1600+XP (1400 MHz) y 512 Mbytes de memoria RAM. Los 6.480 distintos algoritmos genéticos se distribuyen aleatoriamente entre los 4 ordenadores, por lo que cada ordenador ejecuta un total de 1.620 algoritmos con los que se resuelven los 120 problemas de Taillard. A medida que se van resolviendo los algoritmos se van mostrando los resultados en una tabla de Excel que al final de las ejecuciones se consolida con los datos de los 4 ordenadores. El proceso se repite para obtener las dos réplicas. Después los datos se exportan a Statgraphics Plus para Windows versión 5.0 (ver <http://www.statgraphics.com>), donde se analiza el experimento mediante la opción de ANOVA multifactor.

La codificación de los factores es la siguiente: P_c =Cross_Prob, tipo de operador de cruce=Cross_Type, P_m =Mut_Prob, P_{size} =Pop_Size, número de generaciones sin mejora en el C_{max} tras las que se aplica el operador de reinicialización=restart y tipo de operador de selección=Select_Type.

Como se ha comentado, es importante comprobar las hipótesis del ANOVA. La hipótesis de normalidad se puede verificar con un gráfico de probabilidad Normal tras realizar el experimento y guardar los residuos. Este gráfico se muestra en la Figura 4.26.

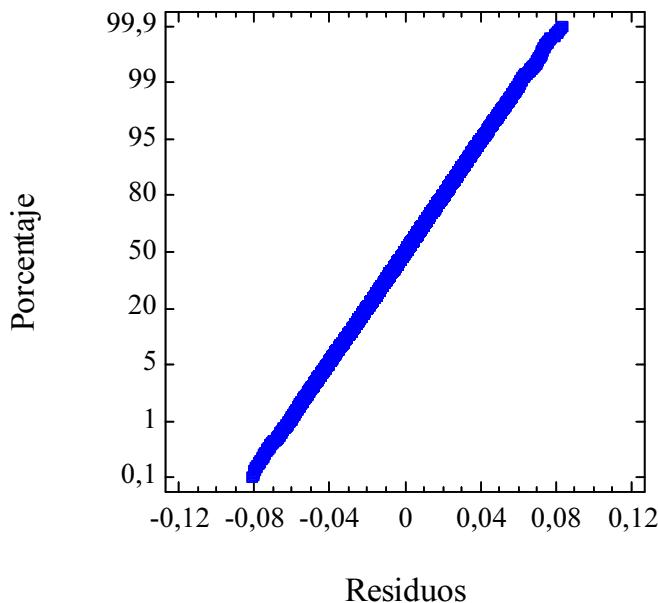


Figura 4.26 – Gráfico de probabilidad Normal para comprobar la hipótesis de normalidad en el ANOVA.

Como se puede observar, se cumple la hipótesis de normalidad. Adicionalmente, mediante un ajuste a la distribución Normal, el test Chi-cuadrado proporcionó un p-valor de 0,283495, o lo que es lo mismo, es difícil descartar la hipótesis de que los residuos se distribuyen como una Normal. La hipótesis de homocedasticidad se puede comprobar graficando los residuos frente a los niveles o variantes de los distintos factores y también obteniendo la gráfica de los residuos frente a los valores previstos del *IPSUM* total. Las Figuras 4.27 y 4.28 muestran estos dos análisis. Las gráficas de los residuos frente a los niveles o variantes para el resto de factores del experimento se pueden consultar en el Anexo A.

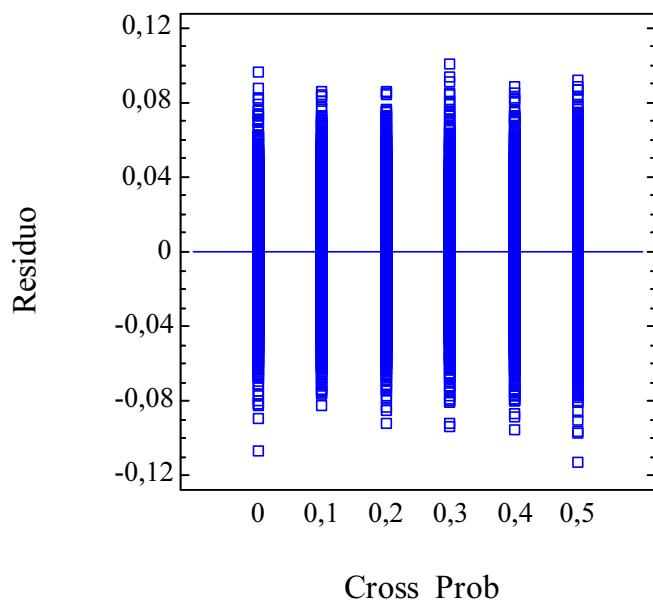


Figura 4.27 – Gráfico de los residuos frente a los niveles del factor probabilidad de cruce (Cross_Prob) para comprobar la hipótesis de homocedasticidad en el ANOVA.

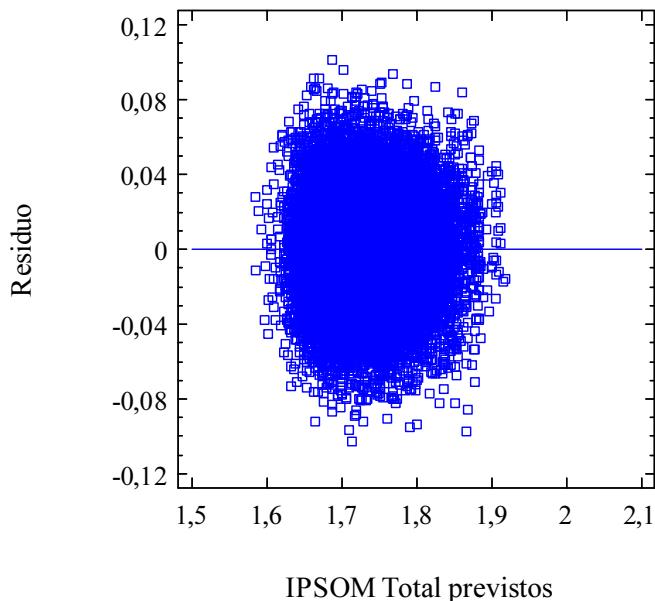


Figura 4.28 – Gráfico de los residuos frente los valores previstos de *IPSON* total para comprobar la hipótesis de homocedasticidad en el ANOVA.

Como vemos, no existen motivos para pensar que la varianza no es constante en el experimento. La hipótesis más importante en el ANOVA es la de independencia, es decir, que no exista un factor oculto que esté afectando de una forma sistemática a las pruebas. En principio, dado que las pruebas se ejecutan aleatoriamente en los ordenadores, y que el entorno de computación se reinicia para cada algoritmo, es difícil que existan problemas de independencia. Graficando los residuos frente al número u orden de ejecución de las pruebas se puede ver si existe estructura en el residuo y por tanto una falta de independencia. La Figura 4.29 muestra este gráfico.

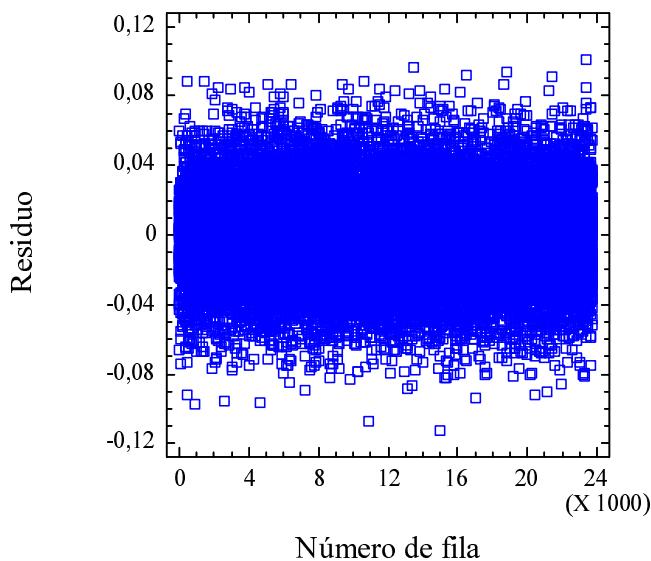


Figura 4.29 – Gráfico de residuos frente al orden de ejecución de las pruebas para comprobar la hipótesis de independencia en el ANOVA.

Se puede observar como no existe ningún tipo de patrón en el residuo, por lo que no hay razones para no asumir esta hipótesis. Hemos visto como se han cumplido las tres hipótesis para el experimento. Por otra parte, dada la gran cantidad de datos y la naturaleza aleatoria del GA desarrollado, es fácil que las hipótesis se cumplan.

Llegados a este punto podemos analizar los resultados del experimento mediante la tabla del ANOVA que se muestra en la Tabla 4.2.

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	3,12506	5	0,625013	883,45	0,0000
B:Cross_Type	5,78227	8	0,722784	1021,65	0,0000
C:Mut_Prob	17,8034	4	4,45086	6291,23	0,0000
D:Pop_Size	0,0662939	3	0,022098	31,24	0,0000
E:Restart	2,13957	2	1,06978	1512,12	0,0000
F:Select_Type	12,5083	1	12,5083	17680,31	0,0000
INTERACTIONS					
AB	1,30727	40	0,0326818	46,20	0,0000
AC	1,20115	20	0,0600576	84,89	0,0000
AD	0,0942268	15	0,00628179	8,88	0,0000
AE	1,40892	10	0,140892	199,15	0,0000
AF	1,15512	5	0,231024	326,55	0,0000
BC	0,29953	32	0,00936033	13,23	0,0000
BD	0,114871	24	0,00478631	6,77	0,0000
BE	0,0229379	16	0,00143362	2,03	0,0088
BF	0,204433	8	0,0255541	36,12	0,0000
CD	0,0314885	12	0,00262405	3,71	0,0000
CE	0,556373	8	0,0695467	98,30	0,0000
CF	4,06082	4	1,01521	1434,98	0,0000
DE	0,00243114	6	0,000405191	0,57	0,7524
DF	0,89417	3	0,298057	421,30	0,0000
EF	0,11672	2	0,0583598	82,49	0,0000
RESIDUAL	13,5912	19211	0,00070747		
TOTAL (CORRECTED)	66,4866	19439			

All F-ratios are based on the residual mean square error.

Tabla 4.2 – Tabla ANOVA con los resultados del experimento del GA propuesto.

Observando la tabla vemos que existe un primer problema. Disponemos de 19.440 datos y como vemos, tenemos 19.211 grados de libertad residuales. Los experimentos donde el tamaño de la muestra es tan grande presentan algunos problemas a la hora de interpretar los resultados. Como vemos, el “*p-value*”, que normalmente sirve como medida de significación estadística, aquí es de poca utilidad, ya que prácticamente todos los *p-values* son cero. Esto se debe a que

realmente el ANOVA realiza un contraste de hipótesis sobre las medias y cuando el p-value es menor que un α determinado (la probabilidad de rechazar la hipótesis nula cuando es cierta) significa que realmente existen diferencias **estadísticamente significativas** entre las medias de los niveles o variantes para un factor o interacción. Cuando el tamaño de la muestra es grande, o como en nuestro caso, muy grande, es muy probable que se detecten diferencias significativas entre las medias de los niveles o variantes para un factor aunque estas diferencias sean muy pequeñas o próximas a cero. De hecho, cuando el tamaño de la muestra tiende a infinito, cualquier diferencia distinta de cero entre las medias de dos niveles de un factor, por infinitesimal que sea, resultará significativa. Este extremo se ilustra con detalle en Montgomery y Runger (1994), donde se marca una clara diferenciación entre la **significación estadística** y la **significación real o ingenieril**, es decir, una interacción entre tres factores puede resultar significativa estadísticamente, pero puede ocurrir que las diferencias en el valor de la variable respuesta para los distintos niveles de la interacción sean muy pequeños y por tanto la significación estadística carezca de una significación real. Montgomery y Runger recomiendan, cuando el tamaño de muestra es grande, fijarse en la magnitud del ratio F o “*F-Ratio*”, que es el ratio entre la varianza explicada por el factor o interacción y la varianza residual del experimento. De esta manera, un “*F-Ratio*” alto para un factor indicará que éste afecta más a la variable respuesta. Siguiendo estos resultados, procederemos a analizar los resultados del experimento de la siguiente manera: ordenaremos todos los factores e interacciones entre los factores por orden decreciente del “*F-Ratio*” e iremos fijando los factores de acuerdo a los gráficos de las medias para cada factor o interacción. Finalizaremos el proceso cuando todos los factores simples estén fijados a un nivel o variante. Para ello utilizaremos los gráficos de medias e intervalos LSD. En el experimento hemos obviado las interacciones de tres y más factores puesto que prácticamente todos los “*F-Ratio*” eran inferiores a 1 y por tanto hay escasa utilidad real en el estudio de este tipo de interacciones.

Fijándonos en la tabla del ANOVA, el factor más influyente es `Select_Type`, el gráfico de las medias para este factor se muestra en la Figura 4.30.

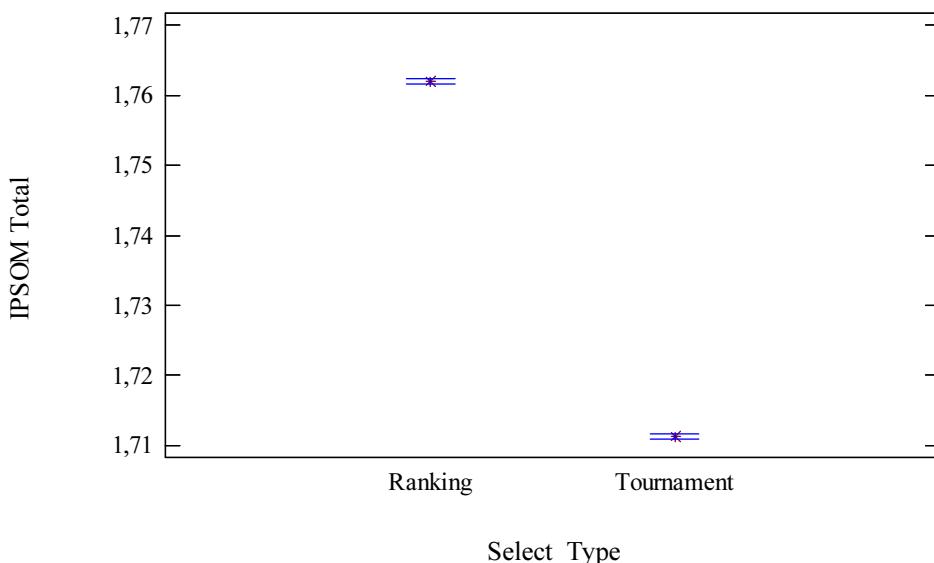


Figura 4.30 – Gráfico de medias e intervalos LSD al 95 % para el factor tipo de selección (Select_Type).

Vemos que existe una clara diferencia entre los dos tipos de selección. La selección por torneo binario tiene una presión más elevada que la selección por ranking y eso se ha notado en el *IPSON* total que es mucho más reducido para la selección por torneo. Este tipo de selección tiene la ventaja añadida de que no es necesario hacer ordenaciones de la población, por lo que es muy rápida. El segundo factor en orden de magnitud del “*F-Ratio*” es el factor *Mut_Prob*, que se muestra en la Figura 4.31.

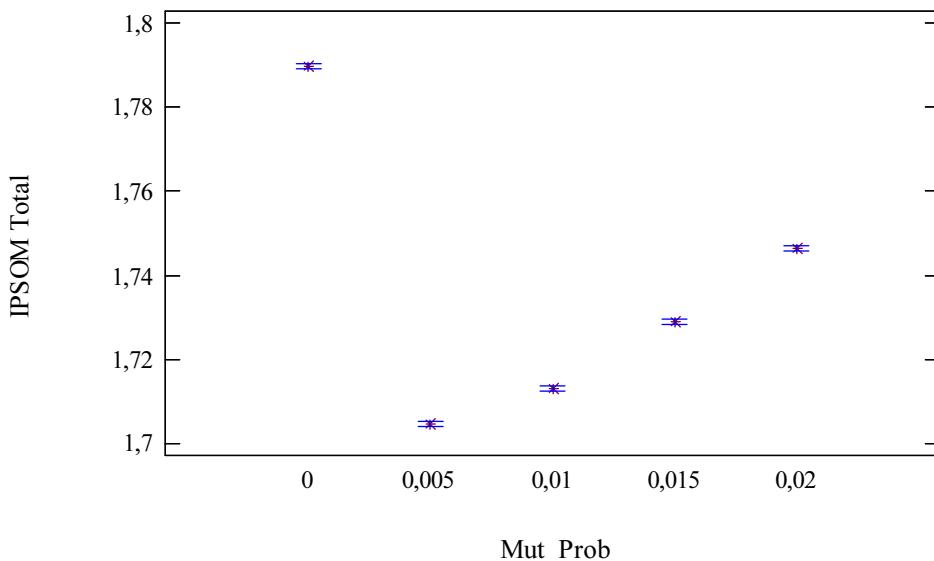


Figura 4.31 – Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de mutación (Mut_Prob).

Resulta interesante observar el comportamiento de la probabilidad de mutación. Vemos como una mutación alta del 2 % ($P_m=0,02$) resulta ser poco recomendable, si reducimos paulatinamente P_m vamos obteniendo mejores resultados, lo que podría llevar a la falsa conclusión de que el GA propuesto no necesita el operador de mutación. Si nos fijamos en la gráfica, al eliminar la mutación ($P_m=0$) obtenemos un GA que es claramente peor. Luego podemos concluir diciendo que el GA propuesto necesita la mutación, pero a un ratio muy reducido: 0,5 %. El tercer factor por orden de importancia es la interacción entre los dos factores estudiados anteriormente, la Figura 4.32 muestra las medias para esta interacción.

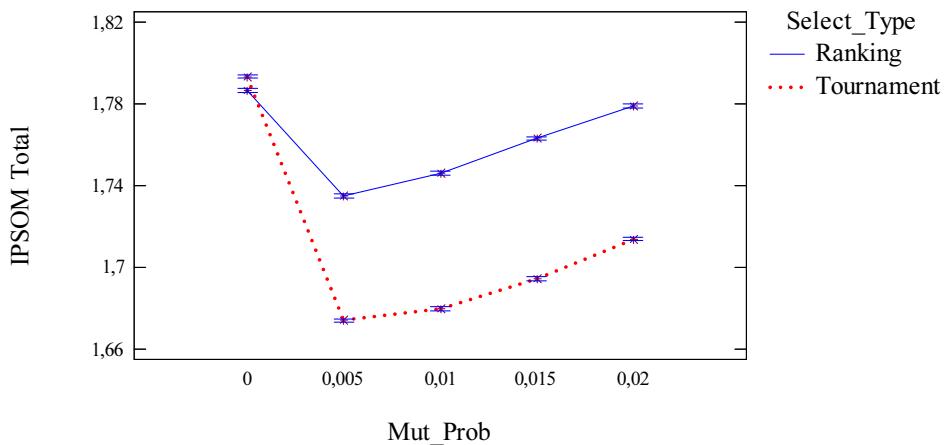


Figura 4.32 – Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores tipo de selección (Select_Type) y probabilidad de mutación (Mut_Prob).

Vemos que las medias se comportan como era de esperar después de haber fijado ya los dos factores que intervienen en la interacción. Concretamente, el mejor rendimiento se obtiene con la selección por torneo y $P_m=0,005$. La razón por la que esta interacción tiene un “F-Ratio” tan alto se debe al cambio de comportamiento cuando alcanzamos $P_m=0$. Vemos que en este caso sería recomendable utilizar la selección por ranking, dado que proporciona mejores resultados. Después de fijar estos dos factores y comprobar la interacción tenemos que el factor que sigue en importancia es Restart, que se muestra en la Figura 4.33.

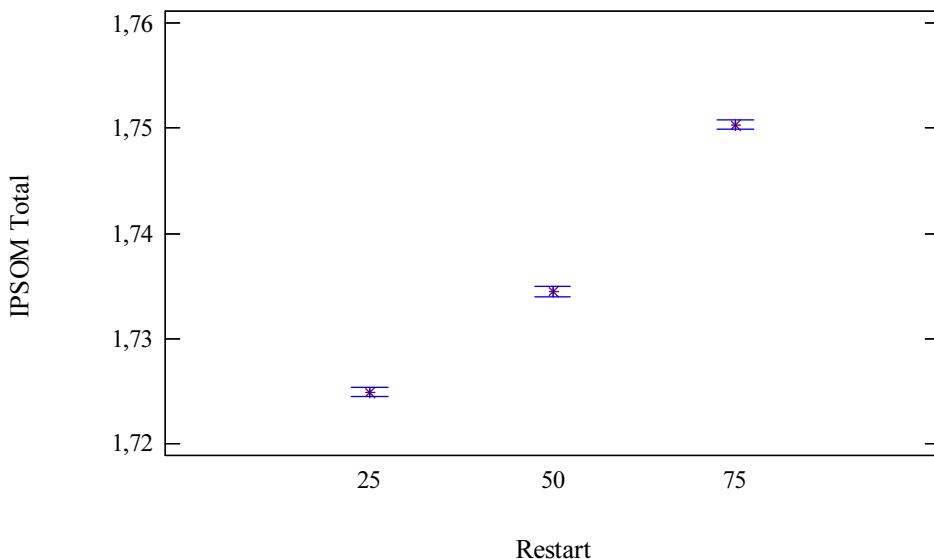


Figura 4.33 – Gráfico de medias e intervalos LSD al 95 % para el factor operador de reinicialización (Restart).

Se puede observar que el GA funciona mejor cuantas más reinicializaciones se produzcan, esto es, cuanto menos generaciones transcurran una vez el valor del C_{max} se ha estancado. En principio deberíamos ampliar el experimento para seguir reduciendo los niveles del factor `Restart`. El problema es que el operador de reinicialización consume tiempo, ya que requiere de ejecuciones de la heurística modificada NEH_m . Parece obvio pensar que con un menor valor del factor `Restart` se obtendrán mejores resultados dado el perfil de la gráfica, pero también a un mayor coste computacional, por lo que el factor `Restart` se fija a 25.

El siguiente factor por importancia es el tipo de operador de cruce o `Cross_Type`, cuyas medias para las nueve variantes se muestran a continuación en la la Figura 4.34.

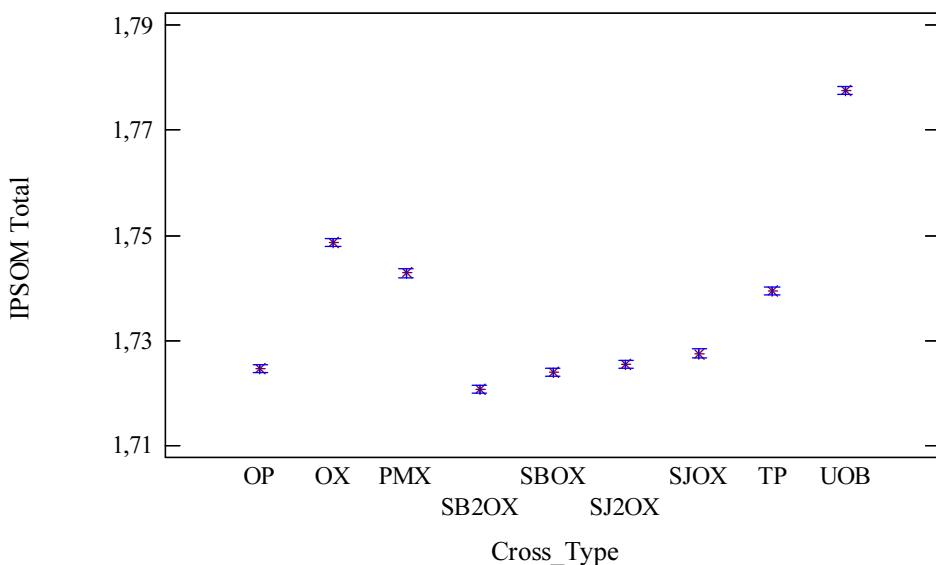


Figura 4.34 – Gráfico de medias e intervalos LSD al 95 % para el factor tipo de cruce (Cross_Type).

El resultado del experimento es muy interesante en lo que respecta a este factor. Ya se ha comentado que el operador PMX está reconocido como uno de los más eficaces para el taller de flujo y también para el TSP. Podemos observar como los cuatro cruces propuestos en esta Tesis Doctoral mejoran el comportamiento del cruce PMX, que es uno de los más eficaces publicados en la literatura. Los operadores OX y UOB son los peores, especialmente este último. El operador de dos puntos por orden y el de un punto por orden se comportan muy bien, en especial el operador de un punto (ya utilizado por Reeves, 1995). No obstante, el mejor cruce es el nuevo SB2OX propuesto en esta Tesis Doctoral. Los otros tres cruces SBOX, SJ2OX y SJOX también presentan un buen comportamiento, aunque este último es ligeramente peor que el cruce OP y entre los cruces OP, SJ2OX y SBOX no parece haber diferencias estadísticamente significativas. También se puede observar como la consideración de bloques de dos trabajos mejora los dos tipos de cruce, es decir, el operador SB2OX resulta mejor que el SBOX y el operador SJ2OX es mejor que el SJOX. Con estos resultados podemos afirmar

que los operadores propuestos, en especial el operador de cruce SB2OX, son hasta la fecha los más eficaces para el taller de flujo de permutación.

El siguiente factor a considerar es la probabilidad de cruce o P_c , cuyo comportamiento se puede observar en la Figura 4.35.

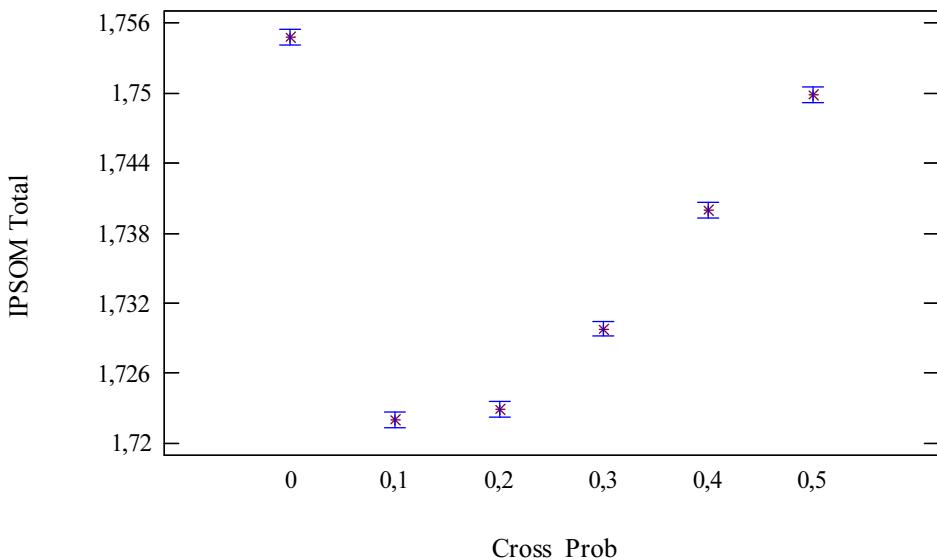


Figura 4.35 – Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de cruce (Cross_Prob).

Como se puede ver, el comportamiento es muy parecido al factor Mut_Prob, ya que conforme reducimos la probabilidad de cruce se obtienen mejores GAs, pero eliminar el cruce del todo ($P_c=0$) resulta en un GA muy poco eficaz. Lo que la gráfica no aclara es si resulta más interesante una probabilidad de cruce del 10 o del 20 %. Este aspecto se resolverá cuando estudiemos las interacciones que afectan al factor Cross_Prob. Lo siguiente a considerar es la interacción entre los factores Pop_Size y Select_Type (Figura 4.36).

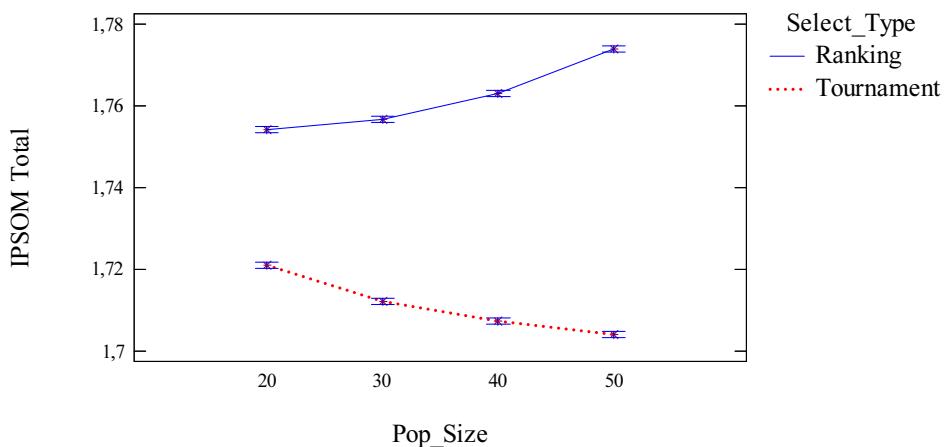


Figura 4.36 – Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores tamaño de la población (Pop_Size) y tipo de selección (Select_Type).

Este es un claro ejemplo de interacción fuerte entre dos factores. Como se puede observar, la selección por torneo es claramente superior a la selección por ranking, independientemente del tamaño de la población, pero el comportamiento del tamaño de la población no es el mismo según un tipo u otro de selección. Mientras que en la selección por torneo aumentar la población tiene un efecto positivo, en la selección por ranking ocurre justamente lo contrario. A partir de la información de esta gráfica fijamos el tamaño de la población o P_{size} a 50. Notar que el factor Pop_Size por sí solo es muy poco importante como se observa en la tabla del ANOVA.

Ahora solo quedaría deshacer el “empate” entre las probabilidades de cruce del 10 y 20 %, para ello recorremos las interacciones que nos quedan por estudiar (por orden del “F-Ratio”) hasta que encontramos la interacción entre los factores Cross_Prob y Select_Type (Figura 4.37).

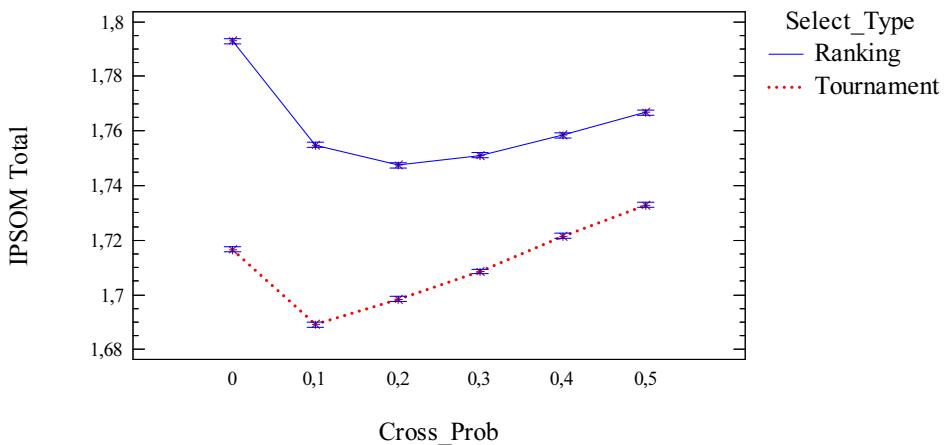


Figura 4.37 – Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores probabilidad de cruce (Cross_Prob) y tipo de selección (Select_Type).

Como se puede ver, en esta interacción, se evidencia que una probabilidad de cruce del 10 % resulta más eficaz.

Por tanto, el algoritmo genético propuesto es el siguiente:

- Tipo de operador de selección: Torneo.
- Operador de cruce: SB2OX.
- Probabilidad de cruce (P_c): 0,1.
- Probabilidad de mutación (P_m): 0,005.
- Tamaño de la población (P_{size}): 50.
- Generaciones antes de reinicialización (G_r): 25.

Como conclusión a la parametrización del GA mediante diseño de experimentos debemos decir que se puede considerar una herramienta muy potente. De los 19.440 algoritmos evaluados, el más eficaz reportó un *IPSON* total de 1,56 %, mientras que el peor un 2,05 %. Esto quiere decir que el mejor es un 31 % más

eficaz que el peor algoritmo, luego la importancia de una correcta parametrización es evidente.

4.4. Nuevo algoritmo genético híbrido

Al principio del capítulo indicamos que se proponían dos algoritmos genéticos. El primer algoritmo genético ya se ha expuesto con detalle. El segundo se trata de una hibridación del primero con una forma de búsqueda local.

Como ya vimos en la Sección 3.1.5 del Capítulo 3, muchos autores han propuesto la hibridación de técnicas metaheurísticas con búsqueda local o incluso con otros métodos metaheurísticos (por ejemplo el GA + simulated annealing de Murata, Ishibuchi y Tanaka, 1996a). Luego esta propuesta no es nueva. Sin embargo, un análisis más exhaustivo de los distintos métodos híbridos propuestos arroja las siguientes conclusiones: muchos de los métodos resultan ser muy lentos, al ser muy exhaustivos. Por ejemplo, en el citado GA de Murata, Ishibuchi y Tanaka, se aplica una fase de “mejora” a todos los individuos al principio de cada generación y antes de efectuar la selección. La fase de mejora consiste en unas iteraciones de un método de búsqueda local descendente, un simulated annealing o un tabu search. Evidentemente, aplicar indiscriminadamente esta fase de mejora a todos los individuos de la población resulta muy costoso en términos computacionales. Además, parece obvio pensar que perder tiempo “mejorando” los peores individuos de la población no va a producir, por término medio, una mejora significativa o que se encuentre una nueva mejor solución. Otros algoritmos híbridos aplican un esquema similar, donde simplemente se “juntan” dos metaheurísticas y no se aprovecha lo mejor de cada una (ver por ejemplo Jain, Bagchi y Wagneur, 2000 o Tang y Liu, 2002). Nosotros proponemos una nueva hibridación basada en una nueva probabilidad, a la que llamamos probabilidad de mejora (P_{enh}), donde, al final de cada generación, después de crear los hijos mediante el cruce y la mutación y antes de incorporarlos a la población, éstos se “mejoran” de acuerdo a la citada probabilidad P_{enh} . Esto es, se obtiene un número aleatorio uniforme entre 0 y 1 y si el número es menor que P_{enh} entonces el hijo se mejora. De esta manera podemos controlar el nivel de hibridación del algoritmo, si hacemos $P_{enh}=1,0$ entonces el esquema de hibridación coincidirá con las propuestas anteriores.

En general, para la fase de mejora existen muchas propuestas, pero para mantener la simplicidad del GA híbrido (al que nos referiremos como HGA), vamos a aplicar una fase de hibridación mediante búsqueda local descendente. Por búsqueda local nos referimos al examen, total o parcial, de algún tipo de vecindario en los hijos. Las heurísticas de mejora de Dannenbring, RACS y RAES (ver Sección 3.1.4.2) son un ejemplo de este tipo de métodos de búsqueda local en el vecindario E restringido. Vimos en el capítulo anterior cómo la heurística NEH de Nawaz, Enscore y Ham (1983) es la más eficiente y eficaz. De esta manera, y con las mejoras de Taillard (1990), podemos aplicar una sencilla forma de búsqueda local, que es parecida al procedimiento Búsqueda_Local del método ILS de Stützle (1998). En un paso de la búsqueda local extraemos un trabajo del hijo, que al ser una secuencia completa tendrá n posiciones. Este trabajo que se ha extraído se inserta en todas las posibles n posiciones de la secuencia parcial de $n - 1$ trabajos y nos quedamos con la mejor como resultado. Se trata de aplicar una búsqueda local que consiste en aplicar la última iteración del tercer paso de la heurística NEH. Esta forma de búsqueda local conlleva la evaluación de n secuencias y el cálculo de n valores de C_{max} , pero con las mejoras de Taillard se puede hacer este paso sin necesidad de calcular ningún C_{max} y de una manera muy rápida.

El número de iteraciones o de pasos de búsqueda local que se aplican en el HGA es también un parámetro de control que llamamos Enh_s (pasos de mejora). De esta manera, el algoritmo genético híbrido (HGA), tras la mutación y de acuerdo a la probabilidad P_{enh} , efectuará Enh_s pasos del citado método de búsqueda local para los hijos y después determinará si éstos entran en la población.

Los parámetros P_{enh} y Enh_s no se fijan entendiendo a la minimización del *IPSUM* total, ya que, evidentemente, la mejora será máxima cuando $P_{enh}=1,0$ y cuanto mayor sea Enh_s , sino que se fijan siguiendo criterios de eficiencia. Es necesario buscar un compromiso entre la calidad de las soluciones y el tiempo de cálculo.

El comportamiento inicial del HGA resultó ser decepcionante, para valores de P_{enh} y Enh_s en principio “razonables” desde el punto de vista de la eficiencia, el HGA se comportaba marginalmente mejor que la versión estándar (que llamamos

simplemente GA) mientras que los tiempos de cómputo eran superiores. Haciendo un análisis exhaustivo de la fase de búsqueda local pudimos comprobar como ésta realmente funcionaba, dado que era común que el hijo mejorase (en algunos casos significativamente) después de la búsqueda local. El problema es que muchas veces el hijo de partida no es bueno y el tiempo invertido en mejorarla no se traduce en una mejora general. En cambio, aumentar el parámetro P_{enh} a 1,0 sí parecía mejorar el comportamiento del HGA considerablemente (a base de disparar el tiempo de proceso). Un balance entre el esquema de hibridación básico, donde todos los individuos se mejoran y el esquema propuesto a partir del parámetro P_{enh} es necesario. ¿Por qué funciona mejor el esquema básico? De nuevo, tras estudiar los algoritmos resultaba ser que el esquema básico (o el HGA con $P_{enh}=1,0$) funcionaba mejor ya que en todas las generaciones todos los individuos, incluidos los de menor C_{max} , se mejoraban. Con este resultado propusimos que en todas las generaciones, el individuo con menor C_{max} se mejorase (pero solo éste). De nuevo, esto ralentizaba el HGA, por lo que finalmente se determinó aplicar la fase de búsqueda local al mejor individuo de la población con una probabilidad $2 \cdot P_{enh}$.

Tras esta mejora se consiguió aumentar la eficacia del HGA al tiempo que se mantuvo el tiempo de proceso en un nivel aceptable. No obstante, se observó que en algunas ejecuciones y para algunos problemas, el HGA se estancaba. El mejor individuo no mejoraba y por tanto se “desperdiciaban” numerosas iteraciones de búsqueda local intentando mejorar el individuo con menor C_{max} . Creemos que este problema es debido a la existencia, como ya se ha comentado en secciones anteriores, de óptimos locales muy fuertes de los cuales no es fácil escapar utilizando movimientos en el vecindario I (que es, al fin y al cabo, la búsqueda local que se aplica). De esta forma desarrollamos un sistema de turnos de búsqueda local que funciona de la siguiente manera: se crea una lista ordenada por el C_{max} de cada individuo y para cada uno se almacena el número de pasos de búsqueda local que se han aplicado sin éxito (sin reducir el C_{max} del individuo). De forma que cuando se superan un número máximo de pasos de mejora “infructuosos” mejoramos el segundo individuo con menor C_{max} . De una forma esquemática, la mejora por búsqueda local funciona como sigue:

1. Ordenar la población ascendentemente por el C_{max} de cada individuo, almacenar la lista ordenada en ℓ . $S(\ell_{(j)})$ es el número de pasos de búsqueda local que se han aplicado al individuo que ocupa la posición j de ℓ sin mejorar el C_{max} . Inicializar $j = 1$ y $mult = 10$.
2. Si $S(\ell_{(j)}) < Enh_s \cdot mult$ entonces:
 - $\ell'_{(j)} = \text{Búsqueda_Local}(\ell_{(j)})$.
 - Si $C_{max}(\ell'_{(j)}) < C_{max}(\ell_{(j)})$ entonces hacemos $S(\ell_{(j)}) = 0$, en caso contrario $S(\ell_{(j)}) = S(\ell_{(j)}) + Enh_s$. Ir al paso 5.
3. Si $S(\ell_{(j)}) \geq Enh_s \cdot mult$ entonces $j = j + 1$ y volver al punto 2.
4. Si $j > P_{size}$ entonces hacemos $mult = mult \cdot 2$, $j = 1$ y vamos al paso 2.
5. FIN.

Por ejemplo y siguiendo el esquema anterior, podemos arrancar el HGA y en un momento dado se alcanza el final de una generación, por lo que se realiza una búsqueda local en el mejor individuo. Suponemos que se obtiene un número aleatorio inferior a $2 \cdot P_{enh}$ y por tanto se ejecuta el procedimiento de búsqueda local. En este momento supongamos que $Enh_s = 5$ y que $mult = 10$ por lo que buscaremos el individuo con menor C_{max} de la población y llamaremos al procedimiento de `Búsqueda_Local` un total de 5 veces. Supongamos también que tras realizar estos pasos 10 veces (50 iteraciones o $Enh_s \cdot mult$ veces) no se ha conseguido reducir el C_{max} del mejor individuo. En este caso pasaríamos al segundo mejor individuo, al que se le permitiría un “turno” de 50 iteraciones de búsqueda local, si este segundo tampoco se mejorase en estas 50 iteraciones se pasaría al tercero y así hasta llegar al último individuo. Se podría dar el caso que todos los individuos fuesen óptimos locales fuertes y por ello deberíamos “alargar” el turno, con lo que si el último individuo ha consumido sus 50 pasos de búsqueda local sin mejoras haríamos $mult = mult \cdot 2$, con lo que se permitirían un total de 100 pasos de búsqueda local, como todos los individuos habrían consumido 50, tendrían 50 pasos adicionales de búsqueda local. La serie que sigue la variable $mult$ es de 10, 20, 40, 80, 160,... por lo que cada vez se permiten el

mismo número de pasos adicionales que pasos se llevan.

Con este esquema de turnos se consiguió un algoritmo genético híbrido rápido y además muy eficaz.

4.5. Evaluación de los algoritmos genéticos propuestos

Una vez se han detallado ambos algoritmos genéticos GA y HGA, vamos a compararlos con los mejores algoritmos heurísticos y metaheurísticos de la evaluación que se llevó a cabo en la Sección 3.2 del capítulo anterior. Concretamente, los mejores métodos que se consideran para la evaluación de los dos GAs propuestos son la heurística NEH de Nawaz, Enscore y Ham (1983) con las mejoras de Taillard (1990), el simulated annealing de Osman y Potts (1989), los algoritmos genéticos de Reeves (1995) y de Chen, Vempati y Aljaber (1995), y la búsqueda local iterativa de Stützle (1998). Todos estos métodos ya se evaluaron en la citada sección y aquí lo que hacemos es añadir los algoritmos genéticos propuestos a la evaluación. Recordemos que la variable que se pretende minimizar es el *IPSOM* total para los 120 problemas de Taillard. El criterio de parada se ha fijado igualmente a 50.000 evaluaciones del C_{max} y se han realizado cinco ejecuciones independientes de cada algoritmo. Para el algoritmo genético híbrido (HGA) se han fijado los parámetros P_{enh} a 0,05 (un 5 %) y Enh_s a 25 pasos. Los resultados se muestran en la Tabla 4.3.

Problema	NEH	SAOP	GAChen	GAReev	ILS	GA	HGA
20x5	3,35	1,39	3,82	0,70	0,30	0,28	0,05
20x10	5,02	2,66	4,89	1,92	0,94	0,77	0,30
20x20	3,73	2,31	4,17	1,53	0,87	0,61	0,18
50x5	0,84	0,69	2,09	0,26	0,13	0,07	0,02
50x10	5,12	4,25	6,60	2,58	2,16	1,98	1,36
50x20	6,20	5,13	8,03	3,76	3,45	2,84	2,26
100x5	0,46	0,40	1,32	0,18	0,11	0,10	0,06
100x10	2,13	1,88	3,75	1,08	0,71	0,81	0,60
100x20	5,11	5,21	7,94	3,94	3,30	2,97	2,34
200x10	1,43	1,56	2,70	0,82	0,45	0,53	0,39
200x20	4,37	4,83	7,07	3,33	2,72	2,65	2,20
500x20	2,24	3,40	4,61	1,83	1,30	1,48	1,27
Media	3,33	2,81	4,75	1,83	1,37	1,26	0,92

NEH=Algoritmo de Nawaz, Enscore y Ham (1983), SAOP=Simulated annealing de Osman y Potts (1989), GAChen=GA de Chen, Vempati y Aljaber (1995), GAReev=GA de Reeves (1995), ILS=Búsqueda local iterativa de Stützle (1998), GA=GA propuesto, HGA=GA híbrido propuesto

Tabla 4.3 – Incremento porcentual medio por encima del óptimo o mínima cota superior conocida (*IPSONM*) de los mejores métodos heurísticos, metaheurísticos y los algoritmos genéticos propuestos.

Podemos observar que el comportamiento de ambos algoritmos genéticos es muy bueno. Concretamente, el algoritmo GA supera al mejor método metaheurístico hasta el momento, el método ILS en muchos grupos de problemas. Por ejemplo, en el grupo de problemas de 50 trabajos y 20 máquinas, uno de los más duros del benchmark, el algoritmo GA es un 21,48 % mejor que el ILS. En otros casos, sin embargo, el GA es peor, aunque de media, el GA es casi un 9 % mejor que el ILS. Si consideramos el algoritmo HGA la situación cambia radicalmente. Podemos decir que el método HGA propuesto es el más eficaz de todos los métodos existentes hasta el momento para el taller de flujo de permutación habiendo utilizado el banco de datos más difícil y extenso que se conoce, que son los problemas de Taillard. En algunos casos, como por ejemplo en el grupo de problemas de 20

trabajos y 5 máquinas, donde el método ILS tiene un *IPSONM* del 0,30 %, el método HGA tiene un *IPSONM* de 0,05 %, o lo que es lo mismo, es un 600 % mejor. Adicionalmente, el método HGA domina al ILS en todos los grupos de problemas y resulta ser, de media un 48,9 % mejor, lo cual, como vimos en el Capítulo 3, resulta ser una mejora considerable.

Si comparamos los algoritmos genéticos propuestos con los mejores GAs existentes también se obtienen muy buenos resultados, concretamente, el GA es poco más del 45 % mejor que el algoritmo GAReev y éste es casi el doble peor que el HGA. Comparando detenidamente el GA y el HGA con el ILS obtenemos las Tablas 4.4 y 4.5.

Problema	Número de veces solución mejor por ILS	Número de veces solución igual entre ILS y GA	Número de veces solución mejor por GA
20x5	3	1	6
20x10	3	0	7
20x20	3	0	7
50x5	2	1	7
50x10	4	0	6
50x20	2	0	8
100x5	4	0	6
100x10	7	0	3
100x20	1	0	9
200x10	8	1	1
200x20	3	0	7
500x20	10	0	0
Total	50	3	67

ILS=Búsqueda local iterativa de Stützle (1998), GA=GA propuesto

Tabla 4.4 – Comparación entre los métodos metaheurísticos ILS y GA para los problemas de Taillard.

Problema	Número de veces solución mejor por ILS	Número de veces solución igual entre ILS y HGA	Número de veces solución mejor por HGA
20x5	0	2	8
20x10	0	0	10
20x20	0	0	10
50x5	0	2	8
50x10	0	0	10
50x20	0	0	10
100x5	2	0	8
100x10	2	1	7
100x20	0	0	10
200x10	4	0	6
200x20	1	0	9
500x20	4	0	6
Total	13	5	102

ILS=Búsqueda local iterativa de Stützle (1998), HGA=GA híbrido propuesto

Tabla 4.5 – Comparación entre los métodos metaheurísticos ILS y HGA para los problemas de Taillard.

Como podemos ver, no parece haber una diferencia clara entre el método ILS y el GA, parece que ambos métodos son más o menos equivalentes. ILS domina en algunos grupos de problemas mientras que GA lo hace en otros. Al final parece que el método GA es algo mejor. Lo que sí parece claro es que el método HGA presenta un claro mejor comportamiento que el ILS, dado que domina en todos los grupos de problemas. De 120 problemas, el método HGA es mejor que el ILS en 102, mientras que ILS consigue ser mejor que HGA en 13 casos.

Como ya hicimos en el capítulo anterior, resulta muy interesante comparar los tiempos de CPU que necesita cada uno de los métodos. En este caso, se ha utilizado un ordenador con un procesador AMD Athlon 1600+XP (1400 MHz) y 512 Mbytes de memoria RAM para las anteriores ejecuciones. La Tabla 4.6

muestra los tiempos, en segundos que ha necesitado cada método para evaluar los 50.000 C_{max} . Recordemos que la heurística NEH es muy eficiente y necesita menos de 1 segundo de media para evaluar hasta los problemas más grandes, de ahí que no aparezca en la tabla.

Problema	SAOP	GAChen	GAReev	ILS	GA	HGA
20x5	<1	<1	1,08	4,01	2,02	16,13
20x10	<1	<1	1,17	4,09	2,03	11,92
20x20	<1	<1	1,35	4,63	2,38	12,46
50x5	<1	<1	1,82	6,38	2,55	14,79
50x10	<1	<1	2,08	9,94	3,08	16,75
50x20	1,04	1,45	2,53	11,82	4,42	33,52
100x5	<1	1,79	3,97	15,31	4,00	21,49
100x10	1,10	2,26	4,49	18,79	5,69	31,35
100x20	2,09	3,24	5,54	24,04	8,70	59,42
200x10	2,29	5,97	12,91	33,73	12,62	67,91
200x20	4,59	8,18	15,16	41,80	19,36	88,35
500x20	39,48	55,30	101,71	192,03	134,50	286,74
Media	4,42	6,77	12,82	30,55	16,78	55,07

SAOP=Simulated annealing de Osman y Potts (1989), GAChen=GA de Chen, Vempati y Aljaber (1995), GAReev=GA de Reeves (1995), ILS=Búsqueda local iterativa de Stützle (1998), GA=GA propuesto, HGA=GA híbrido propuesto

Tabla 4.6 – Tiempos de proceso (en segundos) utilizados por los mejores métodos metaheurísticos y los algoritmos genéticos propuestos.

Como podemos observar, no solo el GA es ligeramente mejor que el ILS sino que además es mucho más rápido, concretamente algo más de un 83 %. Después de este resultado podemos concluir diciendo que el método GA es preferible al método ILS, puesto que es ligeramente más eficaz y también más rápido. Como hemos visto, el método HGA es muy eficaz, pero la alta calidad de las soluciones lleva aparejada un alto coste computacional. El método ILS necesita algo más de 3 minutos para evaluar 50.000 C_{max} mientras que el método HGA necesita casi 5

minutos (para los problemas de mayor tamaño con 500 trabajos y 20 máquinas). Realmente el método HGA es mejor pero más lento ¿Y si establecemos como criterio de parada el tiempo transcurrido en vez del número de evaluaciones del C_{max} ? de esta manera podremos comparar los algoritmos en idénticas condiciones, es decir, aquellos algoritmos lentos terminarán habiendo evaluado menos C_{max} y los rápidos tendrán la oportunidad de evaluar más secuencias y probablemente obtener mejores soluciones. Establecer un criterio de parada de tiempo homogéneo para todos los tamaños de problema no sería demasiado coherente, por ejemplo, 10 segundos parece demasiado para los problemas más pequeños de 20 trabajos y 5 máquinas mientras que es quizás poco para los problemas de 500 trabajos. Vamos a establecer un criterio de parada que sigue la siguiente expresión:

$$\text{Criterio Terminación} = \frac{n \cdot 60.000}{1.000} \text{ milisegundos}$$

La anterior expresión se puede simplificar, obviamente, a $n \cdot 60$, pero tal cual está permite ver más claramente que se permiten 60 segundos por cada 1.000 trabajos. De esta manera, para los problemas de 20 trabajos permitiremos un total de $20 \cdot 60.000 / 1.000 = 1.200$ milisegundos, o lo que es lo mismo, 1,2 segundos, mientras que para los problemas de 500 trabajos estaremos permitiendo 30.000 milisegundos o 30 segundos. Si nos fijamos, todos estos tiempos son inferiores a los que aparecen en la Tabla 4.6, por lo que es de esperar que los IPSOM totales aumenten, de todas formas, pensamos que estos tiempos son suficientemente reducidos como para permitir el uso de estos métodos en la práctica.

Establecer como criterio de parada el tiempo transcurrido y además medirlo en milisegundos requiere de una precisa programación que permita interrumpir los métodos cuando se haya alcanzado el máximo tiempo permitido. Para las pruebas hemos realizado también un total de cinco ejecuciones independientes para cada uno de los métodos. Los resultados se muestran en la Tabla 4.7.

Problema	SAOP	GAChen	GAReev	ILS	GA	HGA
20x5	1,47	3,67	0,71	0,29	0,29	0,20
20x10	2,57	5,03	1,97	1,26	0,95	0,55
20x20	2,22	4,02	1,48	1,04	0,56	0,39
50x5	0,52	2,31	0,23	0,12	0,07	0,06
50x10	3,65	6,65	2,47	2,38	1,91	1,72
50x20	4,97	7,92	3,89	4,19	3,05	2,64
100x5	0,42	1,18	0,18	0,12	0,10	0,08
100x10	1,73	3,91	1,06	0,85	0,84	0,70
100x20	4,90	7,82	3,84	3,92	3,12	2,75
200x10	1,33	2,70	0,85	0,54	0,54	0,50
200x20	4,40	7,01	3,47	3,34	2,88	2,59
500x20	3,48	5,62	1,98	1,82	1,65	1,56
Media	2,64	4,82	1,84	1,65	1,33	1,15

SAOP=Simulated annealing de Osman y Potts (1989), GAChen=GA de Chen, Vempati y Aljaber (1995), GAReev=GA de Reeves (1995), ILS=Búsqueda local iterativa de Stützle (1998), GA=GA propuesto, HGA=GA híbrido propuesto

Tabla 4.7 – IPSOM de los mejores métodos heurísticos, metaheurísticos y los algoritmos genéticos propuestos. Criterio de parada fijado a $(n \cdot 60000)/1000$ milisegundos.

Esta última tabla nos proporciona mucha información, por ejemplo, tenemos que el HGA es un 15,65 % mejor que el GA cuando se les permite evolucionar durante el mismo intervalo de tiempo. Este resultado demuestra que el la hibridación llevada a cabo en el método HGA es adecuada, en el sentido en que conseguimos mejores resultados en el mismo intervalo de tiempo. Sin el esquema de mejora propuesto en la sección anterior, muy probablemente el HGA hubiese sido inferior al resto de algoritmos al invertir tiempo explotando soluciones pobres. Comparando los GAs propuestos con el resto de métodos obtenemos, si cabe, una situación incluso mejor a la obtenida anteriormente. El método GA es un 24 % mejor que el ILS y poco más del 38 % mejor que el GAReev. El método HGA es casi un 45,5 % mejor que el ILS y un 60 % mejor que GAReev. Las Tablas 4.8 y 4.9 muestran una comparación detallada entre los GAs propuestos y el método ILS.

Problema	Número de veces solución mejor por ILS	Número de veces solución igual entre ILS y GA	Número de veces solución mejor por GA
20x5	4	0	6
20x10	2	0	8
20x20	0	0	10
50x5	0	3	7
50x10	2	0	8
50x20	0	0	10
100x5	3	1	6
100x10	4	0	6
100x20	0	0	10
200x10	7	0	3
200x20	1	0	9
500x20	1	0	9
Total	24	4	92

ILS=Búsqueda local iterativa de Stützle (1998), GA=GA propuesto

Tabla 4.8 – Comparación entre los métodos metaheurísticos ILS y GA para los problemas de Taillard. Criterio de parada fijado a $(n \cdot 60000)/1000$ milisegundos.

Problema	Número de veces solución mejor por ILS	Número de veces solución igual entre ILS y HGA	Número de veces solución mejor por HGA
20x5	3	0	7
20x10	0	0	10
20x20	0	0	10
50x5	1	0	9
50x10	1	0	9
50x20	0	0	10
100x5	1	2	7
100x10	2	0	8
100x20	0	0	10
200x10	5	0	5
200x20	0	0	10
500x20	1	0	9
Total	14	2	104

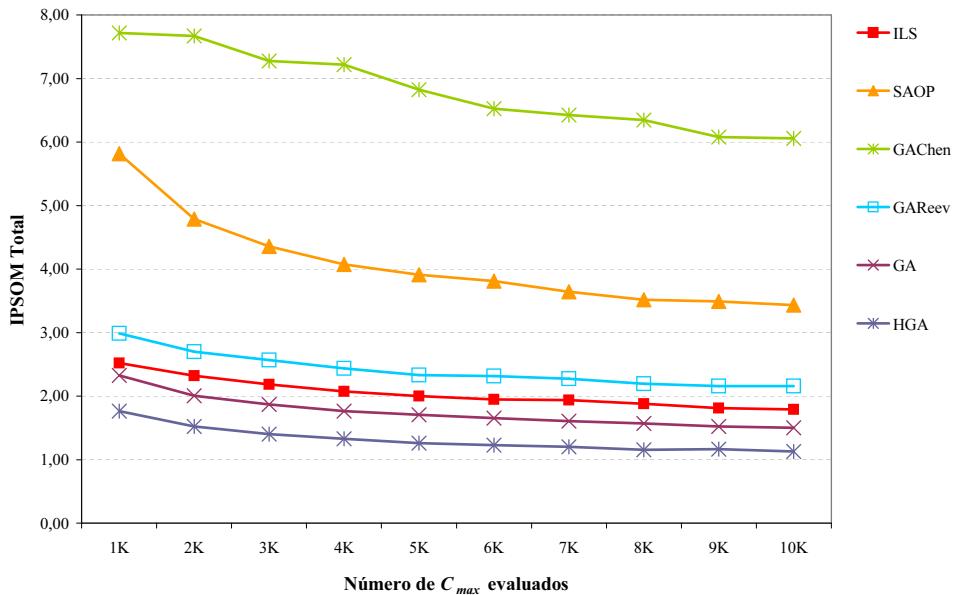
ILS=Búsqueda local iterativa de Stützle (1998), HGA=GA híbrido propuesto

Tabla 4.9 – Comparación entre los métodos metaheurísticos ILS y HGA para los problemas de Taillard. Criterio de parada fijado a $(n \cdot 60000)/1000$ milisegundos.

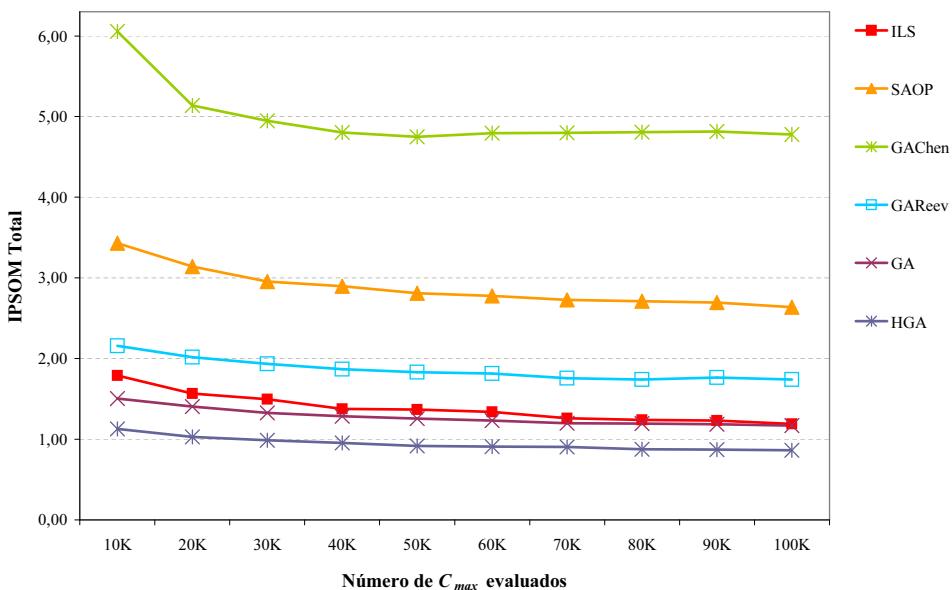
Podemos ver que ambos algoritmos genéticos, en especial el HGA, dominan al método ILS también con el criterio de parada del tiempo. Además se puede observar que la posición de HGA no ha empeorado frente a ILS, ni en la Tabla 4.7 ni en la Tabla 4.9, lo que significa que probablemente el ILS necesita muchas iteraciones para encontrar buenas soluciones y que con 50.000 C_{max} no ha llegado a alcanzar los mejores resultados.

Por último, y para asegurarnos de que los algoritmos genéticos propuestos presentan un mejor comportamiento que el resto de métodos para un amplio rango de evaluaciones del C_{max} , hemos incluido el método GA y HGA en el estudio que se realizó al final del capítulo anterior donde se variaba el número de C_{max} .

evaluados desde 1.000 hasta 10.000 en incrementos de 1.000 y desde 10.000 hasta 100.000 en incrementos de 10.000. La Figura 4.38 muestra el resultado del estudio.



(a) Desde 1.000 hasta 10.000 “makespans”.



(b) Desde 10.000 hasta 100.000 “makespans”.

Figura 4.38 – Evolución del *IPSUM* total frente al número de C_{max} evaluados para los mejores métodos metaheurísticos y los algoritmos genéticos propuestos.

Podemos observar que ambos algoritmos genéticos propuestos consiguen mejorar al método ILS en todo el rango de C_{max} evaluados. No obstante, para 100.000 evaluaciones del C_{max} , el método ILS se acerca ya bastante al método GA, lo que viene a corroborar la idea anterior de que el método ILS requiere de un número elevado de iteraciones para mejorar los resultados. De todos modos, es destacable que el método GA siga siendo ligeramente mejor para 100.000 evaluaciones del C_{max} que el ILS, dado que no se basa en ningún método de búsqueda local y además es mucho más rápido.

4.6. Conclusiones del capítulo

Este capítulo presenta una revisión de la aplicación de los algoritmos genéticos al problema del taller de flujo o “*flowshop*” de permutación, problema denotado como $F/prmu/C_{max}$. En esta revisión se ha comentado la estructura general y los principales operadores útiles para obtener buenas soluciones para el citado problema. Asimismo se ha presentado un algoritmo genético cuyas principales características son una inicialización de la población mediante el uso de heurísticas competitivas, cuatro nuevos operadores de cruce que se basan en la idea de respetar los bloques constructivos en los padres, un operador de reinicialización que busca evitar el estancamiento del proceso de evolución y un nuevo esquema generacional que al mismo tiempo aumenta la presión de la evolución y evita la convergencia prematura.

El conjunto de parámetros y operadores se ha evaluado mediante un amplio diseño de experimentos y un posterior análisis de la varianza que ha permitido obtener una calibración precisa en el algoritmo genético propuesto. Tras esta calibración se ha presentado otro algoritmo genético que básicamente es una hibridación del algoritmo anterior con búsqueda local. La característica novedosa de la hibridación es el control que se realiza sobre los pasos de búsqueda local que se han realizado sobre cada individuo de la población durante la ejecución del algoritmo, de manera que se evita invertir tiempo en individuos que no se han podido mejorar tras un número elevado de pasos.

Estos dos algoritmos se han comparado con los cinco mejores métodos resultado de la evaluación del Capítulo 3 utilizando el conjunto de problemas de Taillard (1993). La comparativa se ha realizado teniendo en cuenta dos criterios de parada: el número de C_{max} evaluados y el tiempo transcurrido. Los resultados indican que ambos algoritmos propuestos presentan una mejor eficacia que el resto de métodos. Concretamente, el algoritmo genético (GA) es entre un 9 % y un 24 % mejor que el segundo mejor método y el algoritmo genético híbrido (HGA) entre un 46 % y un 49 %. Un estudio adicional indica que estas diferencias se mantienen conforme se aumenta el número máximo de C_{max} que se permiten evaluar.

5

CAPÍTULO

EL TALLER DE FLUJO CON TIEMPOS DE CAMBIO DE PARTIDA

En los capítulos anteriores se ha estudiado en profundidad el problema del taller de flujo o $F//C_{max}$ y concretamente, la versión de permutación. Este problema no recoge algunas circunstancias que son relevantes en muchos problemas reales. Al principio del Capítulo 3 se enunciaron hipótesis de partida sobre las cuales se fundamentan los desarrollos posteriores. Una de esas condiciones es que los tiempos de cambio de partida o son muy pequeños y se pueden despreciar, o son independientes de la secuencia y no separables y por tanto pueden incluirse en los p_{ij} . Esta simplificación es muy fuerte, dado que existen muchas configuraciones productivas donde es fácil que esta condición no se cumpla. En el Capítulo 1 se describió el proceso de producción de baldosas cerámicas como un taller de flujo, pero realmente, después de producir un tipo de baldosa en una línea habrá que realizar cambios y ajustes en la misma si se quiere producir otro tipo distinto de baldosa. El tiempo que se tarda en ajustar la línea para el nuevo producto depende de muchas cosas, pero muy importantemente, del tipo de baldosa que se estaba

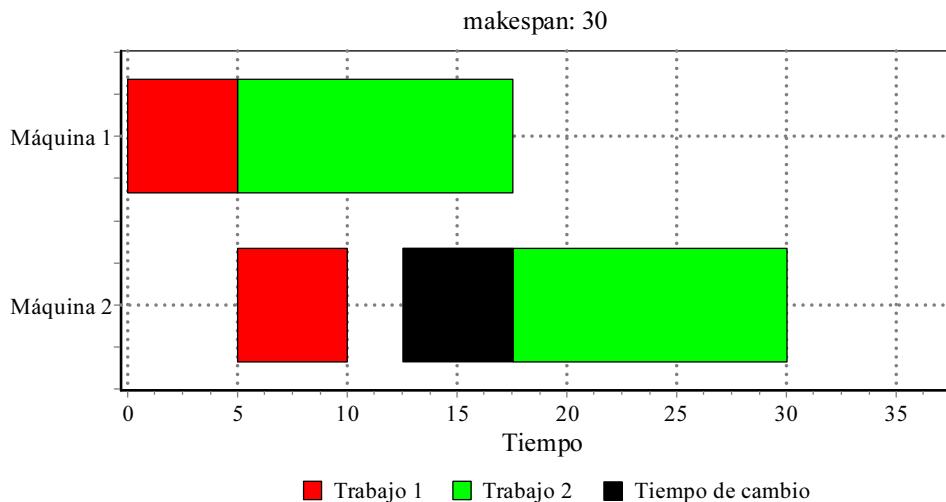
produciendo y del que se pretende fabricar. Por poner un ejemplo, si los dos tipos de azulejos tienen el mismo formato y molde, no será necesario ajustar la prensa y el tiempo de cambio será nulo, es decir, el molde y la configuración de la prensa sirven. Si además ambos azulejos tienen una decoración y colores muy similares, habrá que hacer pocos cambios en la línea de esmaltado. Sin embargo, si ambos productos tienen pocos puntos en común se pueden necesitar horas para preparar la línea de producción. El tiempo necesario para realizar los cambios puede llegar a ser muy elevado. Las industrias químicas son un claro ejemplo, ya que después de producir un tipo de producto en un reactor, es posible que se necesite una limpieza profunda del mismo, incurriendo en un alto tiempo de cambio de partida, por el contrario, el siguiente producto a producir puede ser muy parecido al que justo se ha terminado y por tanto necesitar menos tiempo de limpieza.

De manera general, las operaciones que se realizan en las máquinas y que no están estrictamente relacionadas con el tiempo de proceso de los trabajos se conocen como “ajustes” o “cambios” y estos ajustes llevan un tiempo que se llama de cambio, de preparación, de ajuste o más generalmente, de cambio de partida. Estas operaciones de cambio pueden consistir en la configuración de la maquina para procesar el siguiente trabajo en la secuencia, reunir las herramientas necesarias, limpieza de la máquina y muchas otras situaciones. Se pueden hacer diversas clasificaciones de los problemas de programación de la producción atendiendo a la naturaleza de los tiempos de cambio de partida y que pasamos a comentar a continuación:

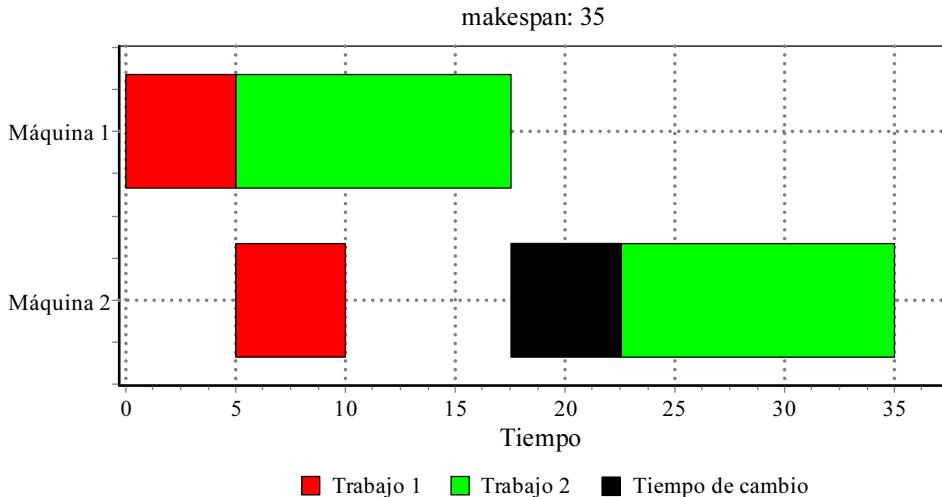
Primero, los tiempos de cambio pueden ser *independientes de la secuencia* o *dependientes de la secuencia*. Se dice que un tiempo de cambio de partida es dependiente de la secuencia si la duración de este tiempo en las máquinas depende del trabajo que se estaba procesando y del que se pretende procesar. En cambio, diremos que el tiempo de cambio de partida es independiente de la secuencia si sólo depende del trabajo que se pretende procesar.

Segundo, los tiempos de cambio pueden ser *anticipativos* (también llamados separables o “*detached*”) o *no anticipativos*, (no separables o “*attached*”). Un tiempo de cambio de partida es anticipativo cuando se puede hacer el cambio antes de que llegue el trabajo a la máquina si es que está ociosa. Un tiempo

de cambio de partida es no anticipativo cuando se requiere que el trabajo esté físicamente en la máquina. Un ejemplo puede ser la acción de fijar una pieza en la máquina y calibrar la misma, para esta operación se requiere que la pieza (trabajo) haya llegado a la máquina. En la Figura 5.1 podemos ver un ejemplo de secuencia de producción con 2 trabajos y 2 máquinas donde existen tiempos de cambio de partida en la segunda máquina. Se puede observar el efecto de los tiempos de cambio separables o anticipativos y no separables o no anticipativos. Evidentemente, es mucho mejor que los tiempos sean anticipativos ya que de esta manera se puede aprovechar el tiempo ocioso que puede existir entre los trabajos en las máquinas para realizar los ajustes, aunque esto, como se ha comentado, depende del sistema productivo.



(a) Tiempos de cambio de partida anticipativos o separables.



(b) Tiempos de cambio de partida no anticipativos o no separables.

Figura 5.1 – Diferencia entre tiempos de cambio de partida anticipativos y no anticipativos para un ejemplo de secuencia con 2 trabajos y 2 máquinas.

A partir de las dos anteriores clasificaciones tenemos las siguientes cuatro posibilidades de acuerdo con la naturaleza de los tiempos de cambio de partida:

- no anticipativos e independientes de la secuencia
- no anticipativos y dependientes de la secuencia
- anticipativos e independientes de la secuencia
- anticipativos y dependientes de la secuencia

Evidentemente, en el primer caso, si los tiempos de cambio no dependen de la secuencia y además hay que esperar a que el trabajo llegue a la máquina para poder realizarlos, entonces se puede incluir el tiempo de ajuste en el p_{ij} y resolver el problema como si de un taller de flujo regular se tratase.

También existe una tercera clasificación de los tiempos de cambio de partida que afecta a los trabajos y que se conoce como clasificación por *lote* ("batch")

o *clase*. En este caso, los trabajos se agrupan en clases o lotes (también llamados a veces familias) por su afinidad y entonces se incurre en un tiempo de cambio de partida cuando se cambia de una clase a otra y en un tiempo de cambio de partida, normalmente menor, cuando se cambia entre dos trabajos dentro de una familia.

De la clasificación anterior, el grado más alto de dificultad se da cuando los tiempos de cambio son anticipativos y dependientes de la secuencia. El presente capítulo se centrará en este caso: el problema del taller de flujo con tiempos de cambio de partida separables y dependientes de la secuencia. Como ya vimos en el Capítulo 2, este problema se denota como $F/S_{sd}/C_{max}$ y para cada máquina i se define una matriz S_{ijk} con número de filas y número de columnas igual a n , que representa el tiempo que se necesita para ajustar la máquina i cuando se quiere procesar el trabajo k después de haber procesado el trabajo j , esto es, cuando k sigue a j en la secuencia. De nuevo, el criterio de optimización más común es la minimización del C_{max} , en el que nos centraremos. Hay que tener en cuenta que el cálculo del C_{max} para este problema no se realiza como se vio en la Sección 3.1 del Capítulo 3, dado que ahora es necesario incluir la información de los tiempos de cambio de partida dependientes de la secuencia o S_{ijk} . Adicionalmente, prácticamente toda la literatura se centra en las secuencias de permutación, por lo que realmente estamos hablando del problema $F/S_{sd}, prmu/C_{max}$, al que algunos autores llaman “*SDST Flowshop*” (de “*Sequence-Dependent Set-up*”). Aquí también utilizaremos, por brevedad, esta nomenclatura. Recordemos que tenemos una permutación o secuencia de n trabajos a la que llamamos π . El trabajo que ocupa la posición k de la secuencia se denota por $\pi_{(k)}$, luego el C_{max} para el SDST flowshop se calcula como sigue:

$$\begin{aligned} O_{i,\pi(j)} s &= \max \left\{ O_{i,\pi(j-1)} f + S_{i,\pi(j-1),\pi(j)}; O_{i-1,\pi(j)} f \right\} \\ O_{i,\pi(j)} f &= O_{i,\pi(j)} s + p_{i,\pi(j)}, \quad i = (1, \dots, m), j = (1, \dots, n) \end{aligned}$$

Donde $O_{0j}f = 0, j = (1, \dots, n)$ y $O_{i0}f = 0, i = (1, \dots, m)$. Una vez calculados estos instantes de comienzo y finalización para cada trabajo se calculan

también los tiempos de finalización:

$$C_{\pi(j)} = O_{m,\pi(j)} f, \quad j = (1, \dots, n)$$

El C_{max} se calcula de la siguiente manera:

$$C_{max} = \max \left\{ C_{\pi(1)}, C_{\pi(2)}, \dots, C_{\pi(n)} \right\}$$

Que como vimos, en el caso del taller de flujo y también en el taller SDST se puede calcular como :

$$C_{max} = O_{m,\pi(n)} f$$

Es importante comentar que algunos autores diferencian también entre tiempos y costes de cambio de partida, de manera que es posible que en algunos entornos productivos hayan algunos tipos de cambio que sean más cortos que otros (en tiempo), pero mucho más caros. En estos casos los costes de cambio se deben considerar explícitamente. Normalmente, el coste del cambio y el tiempo del cambio están íntimamente relacionados, por lo que aquí no haremos tal distinción.

5.1. La complejidad del Taller SDST

La primera impresión que se puede tener es que el taller SDST no es muy distinto al taller de flujo de permutación estándar, pero en realidad se puede decir que el taller SDST es mucho más complejo y difícil de resolver. De hecho Gupta (1986), demostró que el taller SDST es \mathcal{NP} -Completo incluso en el caso en que $m = 1$, o sea, con una sola máquina. Esto contrasta con el taller de flujo estándar, donde para dos máquinas la ya citada regla de Johnson proporciona una solución óptima en un tiempo polinomial. Adicionalmente, Gupta y Darrow (1986) demostraron, en el caso de $m = 2$ y cuando sólo existen tiempos de setup en una máquina (sea la primera o la segunda), que el SDST flowshop también es \mathcal{NP} -Completo. También en este mismo trabajo los autores demostraron que las secuencias de permutación ya no dominan en ningún caso a las secuencias generales cuando el criterio de optimización es el C_{max} . Recordemos que en los casos $m = 2$ y $m = 3$ las secuencias de permutación dominan a las generales

en el taller de flujo estándar. Todo esto indica que el taller SDST es mucho más complejo que el taller estándar y si además tenemos en cuenta que muchas de las propiedades que se han investigado para el taller estándar no se cumplen para el taller SDST, tendremos un nuevo problema, que aunque relacionado con el taller estándar, resulta notablemente más difícil.

Se podría pensar, que aunque el problema $F/S_{sd}/C_{max}$ es más complejo que el F/C_{max} , en la práctica no es necesario tener en cuenta los tiempos de cambio de partida para obtener buenas secuencias de fabricación utilizando los métodos vistos en el Capítulo 3. Desgraciadamente, esto no es cierto. En 1969, Wilbrecht y Prescott compararon un total de siete reglas de prioridad en un problema $J/S_{sd}/C_{max}$, los resultados indicaron una clara ventaja de las reglas que consideraban explícitamente los tiempos de cambio de partida. Aquéllas reglas que no los consideraban resultaron tener un comportamiento muy inferior. White y Wilson (1977) estudiaron también el problema en el caso de una única máquina y en un estudio independiente posterior, Kim y Bobrowski (1994) evaluaron cuatro reglas de prioridad también para el mismo problema, llegando a la misma conclusión. Luego parece evidente que el taller SDST se debe estudiar separadamente y que hay que proponer métodos adecuados para obtener buenas soluciones que tengan en cuenta los tiempos de cambio de partida de manera explícita.

5.2. Métodos exactos para la programación de la producción en el taller SDST

Como ya ocurriera en el taller de flujo estándar, se han propuesto numerosos algoritmos y métodos de carácter exacto para el taller SDST. En esta sección comentamos los más importantes.

Corwin y Esogbue (1974) desarrollaron un algoritmo basado en programación dinámica para el taller SDST con $m = 2$ y donde los tiempos de cambio son dependientes de la secuencia en sólo una de las dos máquinas, e independientes de la secuencia en la otra máquina. Con independencia del caso, los algoritmos propuestos solamente resuelven de manera óptima problemas de entre 12 y 14 trabajos. Uskup y Smith (1975) desarrollaron un algoritmo de bifurcación y aco-

tación para el mismo problema de dos máquinas pero con la existencia de fechas de finalización. El criterio de optimización considerado es la minimización de los costes de cambio de partida donde éstos son directamente proporcionales a los tiempos de cambio. El algoritmo consigue resolver problemas de hasta 30 trabajos, pero únicamente en el caso de dos máquinas. La aplicación de los métodos de bifurcación y acotación a problemas con tiempos de cambio de partida dependientes de la secuencia no se ha limitado solamente al taller de flujo. Por ejemplo, en Gupta (1982), se resolvió el problema del taller de trabajo con este mismo criterio de optimización.

Gupta (1975) presentó un algoritmo de optimización mediante búsqueda lexicográfica. El método presentado es muy general y permite incorporar numerosas restricciones y condiciones en el problema, entre ellas los tiempos de cambio de partida dependientes de la secuencia. El criterio que se busca optimizar en este trabajo es una compleja función del coste total de oportunidad, de la cual se puede derivar el C_{max} .

Srikar y Ghosh (1986) desarrollaron una formulación de programación lineal entera mixta o MILP para el caso de minimización del C_{max} y m máquinas. El modelo propuesto presenta la característica de necesitar menos variables que las formulaciones basadas en el TSP y por tanto es más rápido. Sin embargo, los autores no pudieron resolver de manera óptima problemas más allá de 6 trabajos y 6 máquinas. Posteriormente, Stafford y Tseng (1990), realizaron correcciones al modelo de Srikar y Ghosh y propusieron también modelos para el problema $F/S_{sd}, prmu, nwt/C_{max}$, entre otros. En este caso, el tamaño máximo de problema que se pudo resolver fue de 7 trabajos y 5 máquinas, necesitando más de seis horas de tiempo de CPU en un ordenador tipo PC/AT de la época.

Más recientemente, Ríos-Mercado y Bard (1998a) desarrollaron un algoritmo de tipo ramificación y corte. Este algoritmo, a pesar de ser muy complejo, no permitió resolver problemas con más de 8 trabajos y 6 máquinas en menos de una hora de tiempo de CPU en una estación de trabajo tipo Sun. En un desarrollo posterior (ver Ríos-Mercado y Bard, 1999b), los autores propusieron un algoritmo de bifurcación y acotación con el que mejoraron ligeramente el método anterior y consiguieron resolver de forma heurística problemas de hasta 10 trabajos (con una distancia al óptimo de 1 %). También recientemente, Tseng y Stafford

(2001), propusieron dos modelos MILP para el taller de flujo SDST y el taller $F/S_{sd}, prmu, nwt/C_{max}$. En todo caso los autores pudieron resolver problemas de hasta 9 trabajos y 9 máquinas en aproximadamente cinco minutos de tiempo de CPU en un ordenador tipo PC/AT con un procesador de 800 MHz. Ya en 2002, Stafford y Tseng presentaron otros dos modelos similares, con algunas mejoras sobre los anteriores, sin conseguir resolver problemas de un tamaño superior. Como podemos observar, el considerable esfuerzo realizado para desarrollar métodos exactos, que de manera general y en un tiempo razonable, permitan obtener soluciones para el problema del taller SDST ha sido infructuoso. Debido a esto, se han propuesto varios algoritmos heurísticos que se comentan en la siguiente sección.

5.3. Métodos heurísticos y metaheurísticos para la programación de la producción en el taller SDST

Gupta y Darrow (1986) propusieron cuatro métodos heurísticos para el problema con dos máquinas. En este estudio se determinó que conforme aumenta el ratio entre los tiempos de cambio y los tiempos de proceso, las heurísticas propuestas empeoran drásticamente. Realmente, los autores proponen dos métodos heurísticos basados en la regla de Johnson. Los otros dos métodos se basan en los dos primeros más una técnica de búsqueda local.

En Gupta (1986) se propone un algoritmo basado en una formulación del TSP asimétrico para el problema $F/S_{sd}, prmu, nwt/C_{max}$, o taller SDST donde los trabajos no esperan entre las máquinas, así como para el taller SDST con espacio de almacenamiento limitado entre máquinas. El modelo de tipo TSP resultante propuesto puede resolverse con cualquier método exacto o aproximado existente para el TSP y el autor propone una heurística diseñada por él mismo (ver Gupta, 1978).

Szwarc y Gupta (1987) propusieron un algoritmo heurístico para un caso especial de taller SDST donde se consideran los tiempos de cambio de partida dependientes de la secuencia, separables y aditivos. Por aditivos se entiende que una parte del tiempo depende del trabajo que se está procesando y la otra parte del trabajo

que sigue en la secuencia. Los autores proporcionan un algoritmo óptimo para el caso $m = 2$ y una extensión heurística para el caso general con m máquinas.

Una de las primeras heurísticas generales para el caso de m máquinas se debe a Simons (1992). En este caso se proponen cuatro métodos, dos de ellos son modificaciones de los métodos MINIT y MICOT de Gupta (1972), con adaptaciones para tener en cuenta la existencia de tiempos de cambio de partida dependientes de la secuencia. Las otras dos heurísticas parten del trabajo de Stinson y Smith (1982). Como vimos, en este trabajo se resuelve el problema del taller de flujo a partir del cálculo de una matriz de costes para los trabajos, estos costes son las distancias del problema TSP que se resuelve aplicando una versión del algoritmo de Vogel para problemas de transporte. Simons propone una primera heurística llamada TOTAL donde la matriz de costes la forman la suma de todos los tiempos de proceso y de cambio de partida para los trabajos. En la segunda heurística, llamada SETUP, la matriz de costes sólo contiene la suma de los tiempos de cambio de partida. Tras la evaluación, las heurísticas TOTAL y SETUP dominaron holgadamente a las adaptaciones de los métodos MINIT y MICOT.

Das, Gupta y Khumawala (1995) desarrollaron una heurística basada en unos índices de ahorro. Estos índices seleccionan un trabajo no secuenciado para asignarlo a una posición de una secuencia parcialmente construida. El índice de ahorro considera tres conceptos: ahorro en el tiempo de cambio de partida en la secuencia, ahorro en la duración de la secuencia a partir de los tiempos de proceso y ahorro en los tiempos de finalización de los trabajos. La eficacia de esta heurística se comprobó resolviendo un grupo de 240 problemas para los que se conocía la solución óptima y se calculó la desviación de la heurística con respecto a estos óptimos. En estos casos la heurística nunca se desvió más del 6 %.

Kim, Lim y Park (1996) presentaron un extenso estudio sobre un problema real de producción de tarjetas con circuitos impresos (PCB). El sistema productivo es muy específico y se puede definir como un $F/S_{sd}, prmu, nwt/\bar{T}$, aunque tiene otros aspectos más concretos para los cuales no abarca la notación que aquí utilizamos. Los autores proponen varias heurísticas, todas nuevas, para resolver el problema, entre las que están una adaptación de la heurística NEH, algoritmos tipo simulated annealing, búsqueda tabú y de búsqueda local descendente. De entre todos, los algoritmos tipo SA parecen dominar a los demás.

Parthasarathy y Rajendran (1997) propusieron una metaheurística basada en simulated annealing para el taller SDST aplicado a un problema real donde se consideran unas ponderaciones para los trabajos. Los objetivos de optimización considerados son la minimización del máximo retraso ponderado de los trabajos ($WT_{max} = \max\{w_1T_1, w_2T_3, \dots, w_nT_n\}$) y la minimización del retraso ponderado acumulado para todos los trabajos ($\sum_{j=1}^n w_jT_j$).

Ríos-Mercado y Bard (1998b) propusieron dos métodos para el taller SDST, uno de ellos heurístico, y el segundo metaheurístico. El método heurístico se basa en la adaptación del método de inserción NEH de Nawaz, Enscore y Ham (1983) para la consideración de los tiempos de cambio de partida, en la adaptación, los autores tuvieron en cuenta las mejoras de Taillard y a la heurística resultante la llamaron NEHT-RMB. La propuesta metaheurística se basa en el método GRASP (“*GReedy Adaptive Search Procedure*”), que aúna el uso de heurísticas voraces (o “*greedy*”), la aleatorización y la búsqueda local. Los autores compararon estos dos métodos con la heurística SETUP de Simons. Los resultados fueron mixtos, para ratios cercanos a uno entre el tiempo de cambio y el tiempo de proceso, la heurística SETUP domina a las propuestas, en el caso en el que los tiempos de cambio son menores a los tiempos de proceso, las dos heurísticas, y especialmente el método GRASP resultaron mejores, aunque este último método es considerablemente más lento. En un estudio posterior, (ver Ríos-Mercado y Bard, 1999a), los autores desarrollaron un método basado en las heurísticas de Simons. El método desarrollado, al que llaman HYBRID, se basa en un parámetro θ que pondera, en la función de coste para el TSP, el peso que tienen los tiempos de cambio y los tiempos de proceso, de manera que para distintos valores de θ se obtienen distintas secuencias. Concretamente, la función de coste para cada par de trabajos se calcula de la siguiente manera:

$$C_{jk} = \theta \cdot R_{jk} + (1 - \theta) \cdot S_{jk}$$

donde $\theta \in [0, 1]$, y S_{jk} es la suma de los tiempos de cambio entre los trabajos j y k para toda máquina i , es decir: $S_{jk} = \sum_{i=1}^m S_{ijk}$. R_{jk} es el residuo de secuenciar j antes que k y se calcula de acuerdo al trabajo de Stinson y Smith. En las pruebas realizadas por los autores, HYBRID domina a la heurística SETUP y al método

GRASP para valores pequeños de m mientras que GRASP resulta mejor para problemas grandes.

Norman (1999) propuso varios métodos para un problema particular, el taller SDST donde existe una capacidad de almacenamiento limitada entre las máquinas o $F/S_{sd}, prmu, block/C_{max}$. En este caso el autor probó métodos de tipo búsqueda tabú para el problema. De manera parecida, Allahverdi y Aldowaisan (2001), resuelven un tipo específico de taller SDST donde los trabajos no pueden esperar entre las máquinas y donde el criterio de optimización considerado es la suma de los tiempos de finalización de los trabajos, es decir, el problema considerado es el $F/S_{sd}, prmu, nwt/\sum_{j=1}^n C_j$. Los autores discuten una serie de métodos que proporcionan la solución óptima en el caso en que $m = 2$ y cuando se cumplen una serie de condiciones especiales. También proponen cinco heurísticas diferentes para el problema de m máquinas. En este caso los tiempos de cambio de partida que se consideran son del tipo aditivo.

5.4. Métodos para otros problemas con tiempos de cambio de partida

Existe una literatura extensa que trata problemas que no se pueden clasificar como talleres SDST, pero no por ello se trata de trabajos menos importantes. Un ejemplo claro es el taller de flujo con tiempos de cambio separables e independientes de la secuencia, comúnmente conocido como taller SIST. Este problema se denota por $F/S_{nsd}/C_{max}$ en el caso en el que se pretende minimizar el C_{max} y ha recibido interés por parte de la comunidad científica desde hace varias décadas. Por ejemplo, Yoshida y Hitomi (1979) extendieron el algoritmo clásico de Johnson para resolver el problema $F2/S_{nsd}/C_{max}$ (dos máquinas). El algoritmo propuesto, al igual que el de Johnson, proporciona una solución óptima para el caso considerado. Por otra parte, Stafford y Tseng (2001) presentaron un modelo MILP para el problema del taller SIST, de nuevo, el máximo problema que se resuelve es de 9 trabajos, como ya ocurriera en los modelos para el problema del taller SDST.

Cuando los tiempos de proceso y de cambio se consideran independientes de la se-

cuencia, es posible hilar más fino y hacer una separación entre el tiempo de cambio o preparación, el tiempo de proceso y el tiempo de limpieza o “*removal time*”, que se denota por R_{nsd} . Estos tres tipos de tiempos se consideran explícitamente en Sule (1982), donde se adapta el algoritmo de Yoshida y Hitomi para considerar los tiempos de limpieza independientes de la secuencia. Posteriormente, los autores propusieron métodos heurísticos para este mismo problema en el caso en que $m = 3$ (ver Sule y Huang, 1983). Szwarc (1983) consideró la existencia de “lapsos” en las máquinas asociados a cada trabajo j . Si por ejemplo el lapso l_{ij} para el trabajo j en la máquina i es superior al p_{ij} querrá decir que el trabajo, después de procesarse, debe esperar por un tiempo igual a $l_{ij} - p_{ij}$. Estos casos se dan cuando es necesario esperar tras una operación de pintado o esperar a que el producto se enfrié después de salir de una máquina, por poner un ejemplo. Puede ocurrir que $l_{ij} < p_{ij}$, en este caso se permite un “solape”, es decir, se puede procesar el trabajo en la siguiente máquina de la secuencia sin haberse terminado todavía en la máquina actual. Este puede ser el caso en el que los trabajos realmente son lotes grandes de producción formados por piezas, donde no es necesario que todas las piezas hayan terminado en una máquina para empezar a procesarlas en la siguiente.

Proust, Gupta y Deschamps (1991) consideraron, al igual que Sule, el problema $F/S_{nsd}, prmu, R_{nsd}/C_{max}$ y propusieron un algoritmo de tipo bifurcación y acotación así como cuatro métodos heurísticos para el caso general. Tiempo después, Chandrasekharan y Ziegler (1997) publicaron un trabajo donde se muestran tres algoritmos heurísticos para el mismo problema. Los autores comparan estos algoritmos con los cuatro heurísticos propuestos por Proust, Gupta y Deschamps, resultando mejores pero considerablemente más lentos. Otro tipo de problemas son aquellos en los que se consideran tiempos de cambio de partida pero sólo entre grupos de trabajos, lo que antes hemos llamado clases. Por ejemplo Sotskov, Tautenhahn y Werner (1996) proporcionan varios algoritmos basados en técnicas de inserción de trabajos para este problema con los criterios de C_{max} y $\sum_{j=1}^n C_j$. Danneberg, Tautenhahn y Werner (1999) consideran también un problema muy similar donde además existen decisiones de lotificación.

Otro tipo de problemas, que se verán con más en detalle en el Capítulo 6 son los talleres híbridos, donde en cada etapa del proceso productivo, en vez de

encontrarnos con una sola máquina, tenemos un grupo de máquinas, muchas veces de idénticas características, que pueden procesar la tarea. En Gupta y Tunc (1991) se proponen una serie de métodos heurísticos que resuelven el problema del taller SIST con dos etapas de producción donde hay una única máquina en la primera etapa y un número concreto de máquinas idénticas en la segunda etapa. Li (1997) trabaja con un problema parecido de dos etapas pero añade consideraciones de clases de trabajos y lotificación.

A modo de resumen, la Tabla 5.1 muestra los trabajos más destacados sobre técnicas exactas, heurísticas y metaheurísticas para problemas con tiempos de cambio de partida dependientes de la secuencia y separables, los talleres SDST.

Año	Autor/es	Acrónimo	Tipo	Comentarios
1974	Corwin y Esogbue		Exacto	$m = 2$, programación dinámica
1975	Uskup y Smith		Exacto	$m = 2, d_j, nwt$ bifurcación y aco-tación
	Gupta		Exacto	búsqueda lexicográfica, limitado a $n = 6$
1986	Srikar y Ghosh		Exacto	MILP, limitado a $n = 6$
	Gupta y Darrow		Aprox.	$m = 2$, basado en la regla de John-son
	Gupta		Aprox.	basado en el TSP, nwt
1987	Szwarc y Gupta		Exacto /	tiempos de cambio aditivos, exacto
			Aprox.	para $m = 2$, aproximado para m
1990	Stafford y Tseng		Exacto	MILP, limitado a $n = 7$
1992	Simons	S_TOTAL, S_SETUP	Aprox.	basadas en TSP
1995	Das, Gupta y Khumawala	SI_DGK	Aprox.	índice de ahorro
1996	Kim, Lim y Park		Aprox.	aplicación real, metaheurísticas, criterio \bar{T}
1997	Parthasarathy y Rajendran		Aprox.	aplicación real, simulated annealing, criterios WT_{max} y $\sum_{j=1}^n w_j T_j$
1998	Ríos-Mercado y Bard		Exacto	bifurcación y acotación, limitado a $n = 8$
	Ríos-Mercado y Bard	NEHT_RMB, RMB_GRASP	Aprox.	heurístico basado en NEH, meta-heurístico GRASP
1999	Ríos-Mercado y Bard		Exacto	bifurcación y acotación, limitado a $n = 10$
	Ríos-Mercado y Bard	RMB_HYB	Aprox.	basado en Simons
	Norman		Aprox.	<i>block</i> , búsqueda tabú
2001	Tseng y Stafford		Exacto	MILP, nwt , limitado a $n = 9$
	Allahverdi y Aldowaisan		Aprox.	<i>no - wait</i> , criterio $\sum_{j=1}^n C_j$
2002	Stafford y Tseng		Exacto	MILP, nwt , limitado a $n = 9$

Tabla 5.1 – Métodos exactos, heurísticos y metaheurísticos para el problema del taller SDST.

Esta tabla recoge los resultados más importantes sobre el taller SDST, una revisión excelente y actual sobre talleres de flujo con tiempos de cambio puede verse en Cheng, Gupta y Wang (2000), otros trabajos presentan revisiones del estado del arte sobre problemas de otros tipos en los que se consideran tiempos de cambio, por ejemplo Yang y Liao (1999) y Allahverdi, Gupta y Aldowaisan (1999).

Como podemos observar, los métodos exactos no resultan viables, ya que o están diseñados para $m = 2$ o cuando consideran m máquinas no es posible resolver problemas de más de 10 trabajos. A partir de la revisión anterior, hay una serie de trabajos que consideran heurísticas generales para el taller SDST con el criterio de optimización C_{max} y que permiten su evaluación. Estas heurísticas son las SETUP y TOTAL de Simons, que denotaremos por S_SETUP y S_TOTAL, la heurística basada en índices de ahorro de Das, Gupta y Khumawala y que llamaremos SI_DGK, la mejora a la heurística NEH y la metaheurística GRASP de Ríos-Mercado y Bard (NEHT_RMB y RMB_GRASP) y por último la heurística HYBRID basada en el TSP de los mismos autores y que llamaremos RMB_HYBRID. El resto de trabajos proponen métodos “*ad-hoc*” para problemas reales o para variantes específicas del taller SDST (muchas veces *no-wait, block* o con criterios de optimización distintos al C_{max}) que hacen difícil su adaptación al taller SDST y por ello no se considerarán en las secciones siguientes.

5.5. Adaptación del algoritmo genético al taller SDST

En esta sección proponemos adaptar el algoritmo genético que se expuso en la Sección 4.3 del Capítulo 4 al taller SDST. Que nosotros sepamos, no existe ningún GA propuesto para este problema, la única referencia que tenemos de algoritmos genéticos aplicados a problemas de programación de la producción con tiempos de cambio de partida dependientes de la secuencia corresponde a Rubin y Ragatz (1995) donde desarrollaron un algoritmo basado en GAs para un problema de una única máquina, el $1/S_{sd} / \sum_{j=1}^n T_j$, por lo demás, no se han encontrado trabajos en este aspecto.

Una de las ventajas que tiene el algoritmo GA propuesto para el taller de flujo estándar en el capítulo anterior es que no utiliza propiedades específicas del taller de flujo o condiciones de dominancia que no se cumplen en el caso del taller SDST. Muchos de los algoritmos de tipo bifurcación y acotación propuestos para el $F/prmu/C_{max}$ utilizan estas propiedades y condiciones y por tanto no se pueden aplicar al $F/S_{sd}, prmu/C_{max}$, sobre todo cuando los tiempos de cambio son anticipativos. Otros algoritmos heurísticos, como por ejemplo el método de mejora de Ho y Chang (1991) o las heurísticas de pendiente de Palmer (1965) o Hundal y Rajgopal (1988) se basan en propiedades de los trabajos o de las máquinas en un taller de flujo que no se cumplen en un taller SDST. Ni siquiera la sencilla regla de Johnson funciona en el taller SDST y no se conocen adaptaciones de la regla que funcionen cuando los tiempos de cambio son dependientes de la secuencia (aunque sí para el taller SIST, como hemos visto). Comentamos ahora los cambios realizados al GA para contemplar la resolución del problema del taller SDST.

5.5.1. Representación genética, evaluación e inicialización

Para el algoritmo genético adaptado al taller SDST, que llamaremos GA_{sd} utilizamos la misma representación genética ordinal que en el GA desarrollado y visto en las Secciones 4.2.1 y 4.3.1, dado que los tiempos de cambio no suponen un problema en este caso. La función de evaluación es muy parecida y también se utiliza el mismo mapeado y escalado. Evidentemente, para evaluar el C_{max} de cada individuo hay que utilizar otras ecuaciones, también vistas en este mismo capítulo, ya que obviamente, hay que considerar los tiempos de cambio de partida. Para la inicialización de la población sí es necesario hacer cambios importantes. Ya vimos en el Listado 4.4 que el GA inicializa el primer individuo con la heurística NEH de Nawaz, Enscore y Ham y luego hasta el $B_i\%$ de la población se utiliza la heurística NEH_m . También vimos como el resto de individuos $(100 - B_i\%)$ son secuencias aleatorias. El problema es que la heurística NEH original no considera los tiempos de cambio de partida. Se podría modificar esta heurística para que hiciera los cálculos del C_{max} en cada paso utilizando las fórmulas necesarias en este caso. El problema es que el método NEH original es “lento” y se hace

necesario, para una inicialización eficiente, utilizar las mejoras de Taillard. Estas mejoras aceleran la heurística pero de nuevo se pierde la consideración de los tiempos de cambio de partida. Afortunadamente, Ríos-Mercado y Bard (1998b) extendieron esta heurística NEH mejorada para la consideración de los tiempos de cambio sin perder velocidad, por lo que el algoritmo GA_{sd} utilizará esta versión de la heurística NEH, llamada, como hemos comentado, NEHT_RMB. De igual manera modificamos esta heurística tal y como hicéramos con la original NEH para obtener distintos individuos iniciales. A esta heurística de Ríos-Mercado y Bard modificada la llamaremos NEHT_RMB_m. De esta forma, la inicialización completa se lleva a cabo tal y como muestra el Listado 5.1.

```

function Inicializacion(var P: Population);
begin
    //El primer individuo se genera por la heurística NEH estándar con las
    //mejoras de Taillard y la adaptación de Ríos-Mercado y Bard (1998b)
    Population[1]:=NEHT_RMB;
    //Hasta el  $B_i\%$  de los individuos se generan con la heurística NEHT_RMB
    //modificada
    for l := 2 to round( $B_i \cdot P_{size}$ ) do
        Population[l]:=NEHT_RMBm;
    //El resto de individuos hasta llenar la población se generan al azar
    for m := round( $B_i \cdot P_{size}$ )+1 to P_size do
        Population[m]:=RANDOM;
    end;

```

Listado 5.1 – Proceso de inicialización para el GA_{sd} .

5.5.2. Selección y operadores de cruce

Los operadores de selección no requieren modificaciones, ya que se basan en los valores de adecuación de los individuos que, una vez modificada la función de evaluación, ya incorporan la consideración de los tiempos de cambio de partida. Utilizaremos los mismos operadores de selección para el algoritmo GA_{sd} que para el GA, concretamente la selección por torneo binario y la selección por ranking. Recordar que cuando se utiliza la selección por torneo se aplica el mapeado lineal. De nuevo, los cruces que se consideran son los mismos que se expusieron en la Sección 4.3.3, concretamente, los cruces propuestos para el algoritmo GA_{sd} son los siguientes:

- “*Generalized Position Crossover*” (GPX) (Mattfeld, 1996).
- “*Partially Matched Crossover*” (PMX) de Goldberg y Lingle (1985), (modificado por Cotta y Troya, 1998).
- “*Uniform Order Based Crossover*” (UOB), (Spears y De Jong, 1991 y Syswerda, 1996).
- *cruce de un punto por orden* (OP), (Michalewicz, 1996).
- “*Order Crossover*” (OX), (Davis, 1985a).
- *cruce de dos puntos por orden* (TP), (Michalewicz, 1996).
- “*Similar Job Order Crossover*” (SJOX).
- “*Similar Block Order Crossover*” (SBOX).
- “*Similar Job 2-Point Order Crossover*” (SJ2OX).
- “*Similar Block 2-Point Order Crossover*” (SB2OX).

Hemos decidido incluir el operador GPX porque en simulaciones iniciales no resultaba ser parecido al operador TP como ocurriera con el experimento realizado para el taller de flujo estándar del capítulo anterior. En este caso el operador GPX tiene un comportamiento diferente, mejor en unos casos y peor en otros, de ahí que para este experimento se considere adicionalmente este operador, por lo que tendremos 10 operadores de cruce distintos. Recordemos que los últimos cuatro tipos de cruce son los desarrollados en esta Tesis Doctoral. No ha sido necesario hacer modificaciones en ninguno de los cruces ya que solo operan con la representación ordinal y no hacen uso de características concretas del taller de flujo que no permitan su extensión al taller SDST.

5.5.3. Mutación, reinicialización y esquema generacional

De nuevo, el operador de mutación que se ha implementado es la mutación por desplazamiento o “*SHIFT mutation*”.

La consideración de los tiempos de cambio de partida resulta en algoritmos más

lentos y que consumen mucha más memoria, dado que para el cálculo del C_{max} hay que ir sumando en cada caso los tiempos de cambio y la matriz que contiene estos tiempos tiene un tamaño de $m \cdot n^2$, que para valores grandes de m y n puede ser costosa de manejar. Aún así, hemos decidido mantener el operador de reinicialización, dado que demostró tener un rendimiento muy bueno (ver Sección 4.3.4). Las únicas modificaciones que es necesario realizar en este operador conciernen a la aplicación de la heurística modificada NEH_m cuando se reemplazan el 25 % del 80 % de peores individuos. En este caso utilizaremos la heurística $NEHT_RMB_m$ comentada anteriormente para considerar los tiempos de cambio de partida.

En cuanto al esquema generacional no es necesario hacer ninguna modificación dado que es independiente del tipo de problema a resolver. El esquema generacional propuesto en la Sección 4.3.5 es muy general y aplicable a cualquier tipo de problema, dado que lo único que necesita es una forma de poder evaluar los individuos de la población y también de poder comparar cuando dos individuos son diferentes. Todo esto está ya disponible con la función de evaluación y la representación genética ordinal en el algoritmo GA_{sd} , por lo que no se requieren cambios.

5.5.4. Evaluación de los parámetros del algoritmo genético

Al cambiar el problema y la función de evaluación los parámetros del algoritmo genético obtenidos en la Sección 4.3.6 no tienen por qué ser los mejores para el nuevo problema. Es por esto que planteamos de nuevo un experimento muy parecido al ya realizado para parametrizar y calibrar el GA. En este caso calibraremos el nuevo algoritmo genético GA_{sd} . La selección de operadores y parámetros, así como sus correspondientes niveles o variantes se muestra a continuación:

- Tipo de operador de selección, 2 variantes: Torneo y Ranking.
- Operador de cruce, 10 variantes: GPX, OP, OX, PMX, SB2OX, SBOX, SJ2OX, SJOX, TP y UOB.
- Probabilidad de cruce (P_c), 6 niveles: 0, 0,1, 0,2, 0,3, 0,4 y 0,5.
- Probabilidad de mutación (P_m), 5 niveles: 0, 0,005, 0,01, 0,015 y 0,02.

- Tamaño de la población (P_{size}), 4 niveles: 20, 30, 40 y 50.
- Generaciones antes de reinicialización (G_r), 3 niveles: 25, 50 y 75.

Todas las posibles combinaciones de los niveles y variantes de los parámetros y operadores dan como resultado un total de $2 \cdot 10 \cdot 6 \cdot 5 \cdot 4 \cdot 3 = 7.200$ diferentes combinaciones y por tanto 7.200 algoritmos genéticos distintos. Realizaremos un diseño factorial completo.

Para poder evaluar los algoritmos se hace necesario un conjunto de pruebas que contenga tiempos de cambio de partida dependientes de la secuencia. Tras una búsqueda exhaustiva en la literatura y también por Internet, no se pudieron encontrar conjuntos de pruebas estándar. Por tanto, en Vallada, Ruiz y Maroto (2003) se han propuesto un total de 480 problemas de ejemplo para el taller SDST. La construcción de estos problemas es muy sencilla: existen cuatro grandes grupos de 120 problemas cada uno, las características de los problemas, número de máquinas, número de trabajos y tiempos de proceso son exactamente iguales a los problemas de Taillard (1993) comentadas en la Sección 3.2 del Capítulo 3, a los que se añaden las matrices S_{ijk} con los tiempos de cambio de partida de cada problema. En el primer grupo de problemas el ratio entre los tiempos de proceso y los tiempos de cambio de partida es de 10, es decir, los tiempos de cambio son el 10 % de los tiempos de proceso. De esta manera, si en los problemas de Taillard los p_{ij} son cantidades generadas a partir de una uniforme $U[1, 99]$, los tiempos de cambio en este primer grupo de problemas se distribuyen como una uniforme $U[1, 9]$. A este primer grupo se le denomina SSD10. Los otros tres grupos van disminuyendo el ratio entre los tiempos de proceso y de cambio, el grupo SSD50 tiene un ratio de 2, es decir los tiempos de cambio son el 50 % los de proceso, con lo que se distribuyen como una uniforme $U[1, 49]$. El grupo SSD100 tiene un ratio de 1, es decir, los tiempos de cambio y los de proceso son del mismo orden y ambos se distribuyen como una uniforme $U[1, 99]$. El último grupo, llamado SSD125 tiene un ratio de 0,8. En este caso los tiempos de cambio son un 25 % superiores a los de proceso, luego se distribuirán como una uniforme $U[1, 124]$. La idea de crear cuatro conjuntos de pruebas variando la magnitud de los tiempos de proceso surge después de examinar la literatura

donde muchos autores comprobaron que el ratio entre los tiempos de proceso y de cambio afecta a las prestaciones de los métodos propuestos. El formato que le hemos dado a los ficheros de datos es muy sencillo. La parte inicial del fichero, donde se especifica el número de trabajos y de máquinas permanece inalterada y al finalizar la descripción de todos los tiempos de proceso se definen los tiempos de cambio de partida con el siguiente formato:

$$\begin{array}{cccc}
 M0 & & & \\
 S_{111} & S_{112} & \cdots & S_{11n} \\
 S_{121} & S_{122} & \cdots & S_{12n} \\
 \vdots & \vdots & \vdots & \vdots \\
 S_{1n1} & S_{1n2} & \cdots & S_{1nn} \\
 M1 & & & \\
 S_{211} & S_{212} & \cdots & S_{21n} \\
 \vdots & \vdots & \vdots & \vdots \\
 S_{2n1} & S_{2n2} & \cdots & S_{2nn} \\
 \vdots & & & \\
 M(m-1) & & & \\
 S_{m11} & S_{m12} & \cdots & S_{m1n} \\
 \vdots & \vdots & \vdots & \vdots \\
 S_{mn1} & S_{mn2} & \cdots & S_{mnn}
 \end{array}$$

Hemos de tener en cuenta que $S_{ijk} = 0, i = (1, \dots, m)$ cuando $j = k$. Por ejemplo, la problema ta001 del grupo de problemas SSD10 queda como sigue:

20 5
0 54 1 79 2 16 3 66 4 58
0 83 1 3 2 89 3 58 4 56
0 15 1 11 2 49 3 31 4 20
0 71 1 99 2 15 3 68 4 85
0 77 1 56 2 89 3 78 4 53
0 36 1 70 2 45 3 91 4 35
0 53 1 99 2 60 3 13 4 53
0 38 1 60 2 23 3 59 4 41
0 27 1 5 2 57 3 49 4 69
0 87 1 56 2 64 3 85 4 13
0 76 1 3 2 7 3 85 4 86
0 91 1 61 2 1 3 9 4 72
0 14 1 73 2 63 3 39 4 8
0 29 1 75 2 41 3 41 4 49
0 12 1 47 2 63 3 56 4 47
0 77 1 14 2 47 3 40 4 87
0 32 1 21 2 26 3 54 4 58

0	87	1	86	2	75	3	77	4	18
0	68	1	5	2	77	3	51	4	68
0	94	1	77	2	40	3	31	4	28
SSD M0									
0	5	8	2	7	7	4	5	4	3
9	0	8	1	1	9	6	7	9	6
1	7	0	2	9	5	2	9	5	6
7	7	4	0	6	6	3	7	8	9
5	7	5	3	0	6	6	2	8	5
8	6	2	7	3	0	2	1	7	3
8	7	5	9	1	8	0	7	4	1
6	4	4	4	1	8	1	0	1	1
7	4	2	6	4	3	6	0	3	7
7	8	6	6	6	8	9	6	0	4
5	9	8	5	3	3	7	1	9	0
9	7	4	2	2	7	5	5	4	8
1	9	7	2	4	4	7	9	9	4
9	2	5	4	1	2	2	2	6	4
5	8	8	1	3	8	7	7	4	9
7	6	9	5	6	2	9	6	7	3
2	5	3	7	7	2	1	5	3	4
8	4	3	8	4	7	3	7	6	4
9	4	4	7	2	4	1	7	7	3
5	5	7	4	9	3	6	9	4	5
M1									
0	2	8	9	5	6	2	9	6	6
7	0	5	3	7	9	1	8	6	5
4	2	0	2	7	3	0	1	6	5
9	7	2	0	6	4	1	4	1	4
8	3	5	2	0	4	5	4	3	4
3	7	5	6	9	0	7	8	9	1
6	8	4	4	1	2	0	3	4	3
4	4	5	4	6	4	5	0	6	4
9	1	2	9	7	8	9	4	0	5
5	8	2	4	4	5	2	1	1	0
6	7	5	3	8	3	1	3	2	9
3	5	7	8	1	8	5	7	6	8
3	8	4	9	2	2	2	6	2	1
7	7	5	5	9	7	9	8	3	8
8	8	2	7	6	9	8	8	3	7
9	4	3	6	6	9	3	4	3	8
5	6	1	8	2	9	1	6	6	3
9	2	7	5	7	9	5	5	1	6
8	1	8	3	5	9	7	8	2	6
7	6	9	6	8	7	2	5	9	9
M2									
0	6	8	2	5	3	7	1	4	3
6	0	8	7	2	7	3	6	2	8
5	7	0	6	2	9	5	8	4	3
3	3	6	0	8	4	2	9	5	5
3	4	5	3	0	4	9	4	5	3
6	1	6	1	1	0	4	1	7	6
3	6	3	9	3	1	0	2	1	3
7	1	1	9	7	5	8	0	8	4
1	1	8	9	1	6	6	5	0	9
1	8	1	1	1	3	4	9	1	0
1	3	8	5	9	1	9	5	8	1
2	5	5	3	5	6	6	3	9	8
2	6	2	3	1	3	7	6	8	1
7	2	8	7	4	2	6	2	6	9
9	1	9	3	3	4	7	3	6	2
2	7	6	9	9	6	4	5	4	7
8	9	2	9	2	9	5	8	6	7
1	8	5	8	8	2	6	2	6	7
1	7	3	4	5	5	5	3	7	6
6	9	5	1	9	2	9	4	1	3
M3									
0	7	1	8	1	5	6	7	4	4
5	0	1	5	8	6	9	3	5	8
6	4	0	7	4	1	7	7	6	8
1	4	5	0	8	3	4	9	8	9
3	2	3	9	0	5	8	8	5	5
8	2	8	2	5	0	1	7	8	4
1	3	2	4	3	3	0	4	8	5
9	8	2	2	5	4	6	0	9	3
1	1	8	4	5	7	5	2	0	1
1	7	1	8	4	5	7	1	3	3
6	9	5	1	9	2	9	4	1	3

3	5	1	8	1	9	4	9	7	0	2	7	1	3	2	3	5	8	2	2
9	9	5	2	9	6	3	7	9	6	0	8	5	4	5	2	5	3	5	3
3	2	9	1	5	7	4	4	5	5	6	0	8	4	7	9	1	1	5	1
5	8	2	4	2	6	6	4	4	7	9	2	0	9	6	4	5	3	5	3
6	1	8	6	3	1	4	7	6	1	5	8	6	0	8	4	2	7	9	8
1	3	8	6	7	1	1	2	7	8	7	3	2	1	0	9	1	7	4	9
9	3	2	8	9	9	2	8	8	2	6	2	9	1	9	0	6	3	1	4
6	7	5	1	9	5	4	5	1	6	8	6	3	5	3	3	0	5	1	8
4	8	4	2	3	6	1	8	3	7	1	4	5	3	4	2	6	0	7	5
6	1	4	9	6	7	3	5	1	2	2	3	1	3	6	7	3	6	0	3
2	6	3	7	9	8	4	3	1	2	4	6	1	8	8	7	3	1	6	0
M4	0	5	3	4	4	9	1	2	7	6	7	6	8	4	9	2	4	3	1
3	0	8	4	1	3	1	1	3	4	3	6	4	3	7	9	3	3	5	3
1	6	0	1	4	9	2	1	2	8	3	4	4	3	9	8	8	7	2	2
7	7	4	0	6	3	3	3	3	7	7	3	9	6	5	3	1	3	8	4
6	7	4	6	0	9	5	6	5	7	7	5	4	5	3	1	1	4	3	7
6	8	9	1	9	0	9	3	1	8	9	8	4	2	5	2	5	7	2	8
8	9	1	9	4	3	0	8	9	6	3	4	2	5	4	8	7	7	2	5
1	6	2	5	9	3	4	0	6	7	1	2	5	2	2	3	6	3	8	7
1	7	4	9	5	3	3	3	0	4	4	6	8	3	8	6	8	8	2	8
3	1	6	1	3	5	9	5	3	0	7	2	6	1	2	3	6	8	3	6
9	4	9	9	6	1	4	2	9	8	0	3	1	3	6	4	7	6	8	5
9	7	4	7	4	4	8	8	2	5	7	0	9	5	6	2	1	3	3	9
5	6	3	8	1	4	3	8	1	4	4	5	0	6	8	3	6	7	9	3
5	8	3	2	5	9	8	7	9	3	1	8	5	0	5	6	2	6	7	8
9	8	2	4	3	1	7	6	3	1	1	4	6	3	0	9	1	5	9	6
5	6	7	5	4	5	2	5	8	4	6	8	7	3	1	0	1	1	8	7
3	2	5	1	5	8	9	9	4	2	6	1	4	9	8	7	0	1	2	3
3	9	5	8	2	1	9	7	6	1	7	1	4	3	1	8	0	3	2	
9	5	6	1	6	7	7	9	6	8	8	6	7	2	2	5	9	8	0	2
8	1	3	9	8	1	9	7	3	4	6	7	9	7	1	9	1	7	6	0

Listado 5.2 – Problema ta001 de Taillard aumentado con los tiempos de cambio de partida dependientes de la secuencia al 10 % de los tiempos de proceso. Problema ta001_SSD10.

El resto de problemas se puede descargar de <http://www.upv.es/gio> y también se pueden obtener del CD-ROM anexo (ver Anexo C). Para estas pruebas no se conoce el óptimo puesto que, como hemos visto, no existe tampoco ningún algoritmo que pueda obtener la solución óptima ni tan siquiera para los problemas más pequeños de 20 trabajos. Para poder evaluar si alguno de los 7.200 algoritmos del experimento es bueno resolviendo los problemas, necesitamos conocer un valor de referencia. Si el valor óptimo no se puede conocer existen dos posibilidades: considerar el valor de referencia como la menor solución obtenida por los 7.200 algoritmos (en ese caso el mejor de los algoritmos tendría un incremento sobre esta solución del 0 %) o tomar como referencia una solución obtenida tras permitir un número muy elevado de iteraciones de un algoritmo dado. Creemos que la segunda opción es la más adecuada y más fácil de implementar. Obtuimos un conjunto de “buenas soluciones” para los 480 problemas tras ejecutar el algoritmo GA_{sd} con el criterio de parada fijado a la evaluación de 500.000 C_{max} y con los parámetros y operadores obtenidos en el experimento del capítulo anterior. Las

soluciones obtenidas pueden consultarse en la Sección B.1 del Anexo B. De esta manera, la variable respuesta para nuestro experimento está basada en el cálculo del porcentaje de incremento sobre la mejor solución obtenida que se calcula como:

$$IPSO = \frac{Heu_{sol} - Mejor_{sol}}{Mejor_{sol}} \cdot 100$$

Donde $Mejor_{sol}$ es la mejor solución que hemos obtenido para un problema dado, y Heu_{sol} es la solución obtenida por un algoritmo concreto. Tal y como venimos haciendo, llamaremos $IPSOM$ a la media del $IPSO$ sobre un grupo de problemas y al $IPSOM$ total a la media para los 120 problemas dentro de un grupo. Concretamente, para la realización de los experimentos evaluamos los primeros 90 problemas de cada grupo (hasta 100 trabajos y 20 máquinas), dado que los tiempos de proceso se elevan mucho para problemas más grandes. El criterio de parada para los algoritmos se fija en 5.000 evaluaciones del C_{max} .

Hemos realizado cuatro experimentos distintos, uno por cada grupo de problemas; SSD10, SSD25, SSD100 y SSD125. Con lo que, para cada experimento se han resuelto $7.200 \cdot 90 = 648.000$ problemas, y $648.000 \cdot 5.000 = 3.240.000.000$ secuencias. En general, uniendo los cuatro experimentos se han resuelto $648.000 \cdot 4 = 2.259.000$ problemas y $12.960.000.000$ secuencias. La metodología para la obtención de los resultados es idéntica a la del capítulo anterior, donde se ha utilizado el mismo cluster de cuatro ordenadores para hacer las pruebas. La codificación de los factores también es la misma.

Vamos a analizar el experimento realizado con el primer grupo de problemas SSD10. Lo primero, como ya hicimos en el capítulo anterior, es analizar el cumplimiento de las hipótesis del ANOVA. El gráfico de probabilidad Normal se puede observar en la Figura 5.2.

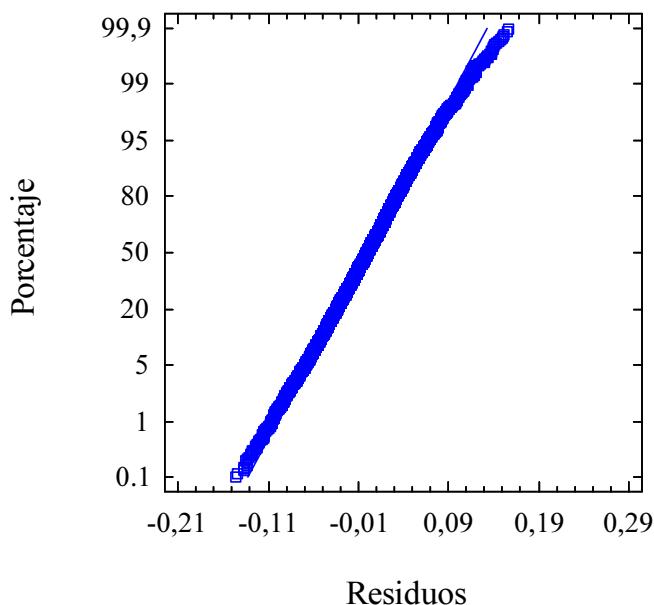


Figura 5.2 – Gráfico de probabilidad Normal para comprobar la hipótesis de normalidad en el ANOVA, experimento SSD10.

Podemos ver que de nuevo se cumple la hipótesis de normalidad. La segunda hipótesis a comprobar es la homocedasticidad. Las Figuras 5.3 y 5.4 muestran los residuos frente a los niveles del factor *Cross_Prob* y los residuos frente a los valores previstos del *IPSONM* total. El resto de gráficos de los residuos frente a los niveles o variantes para los demás factores pueden consultarse en el Anexo A, Sección A.2.

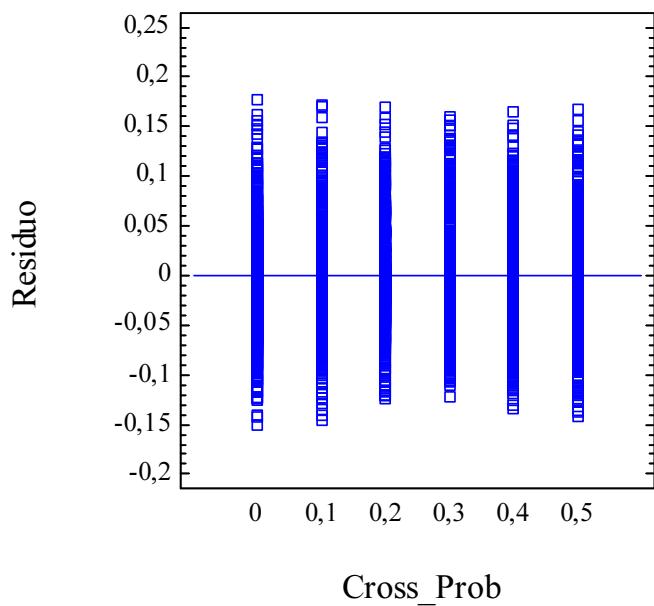


Figura 5.3 – Gráfico de los residuos frente a los niveles del factor Cross_Prob, experimento SSD10.

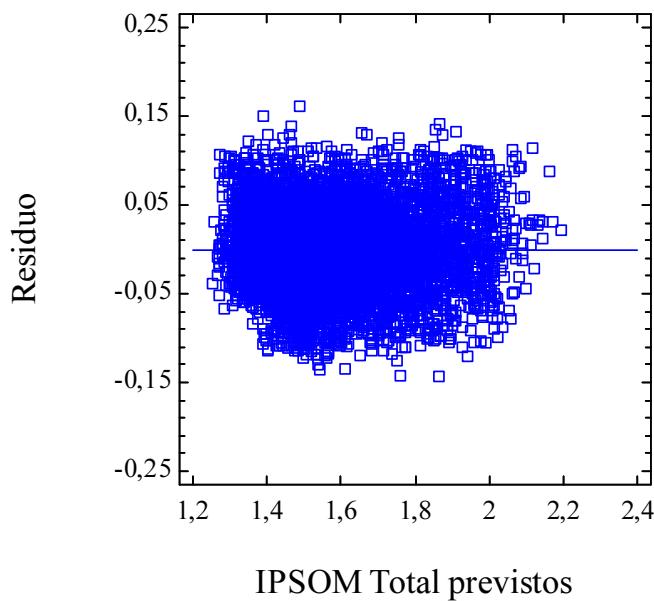


Figura 5.4 – Gráfico de los residuos frente los valores previstos de *IP SOM* total, experimento SSD10.

Como se puede observar en estas figuras y en las que aparecen en el anexo, se puede aceptar la hipótesis de homocedasticidad, dado que no se aprecian niveles o variantes en los factores con una mayor dispersión y tampoco se ve estructura en la Figura 5.4. Por último, la Figura 5.5 muestra los residuos frente al número u orden de ejecución de las pruebas para comprobar si hay una falta de independencia.

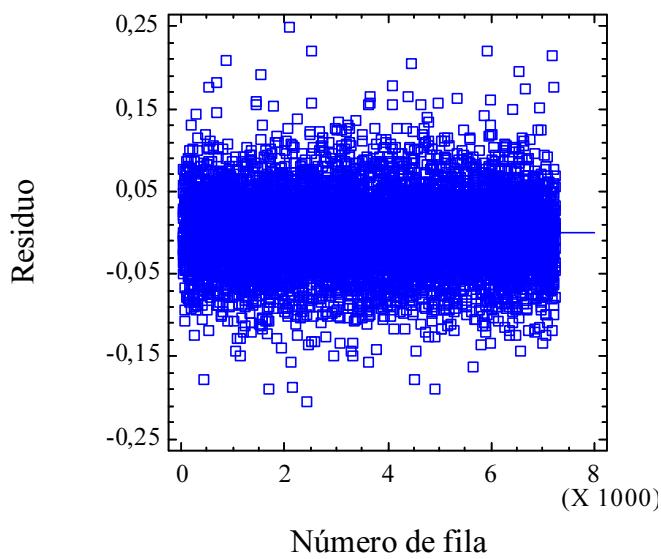


Figura 5.5 – Gráfico de residuos frente al orden de ejecución de las pruebas, experimento SSD10.

Podemos decir, sin temor a equivocarnos, que las tres hipótesis se cumplen y por tanto podemos proseguir con el análisis. Vamos a analizar los resultados del experimento mediante la tabla del ANOVA que se muestra en la Tabla 5.2.

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	4,39052	5	0,878103	406,34	0,0000
B:Cross_Type	73,7525	9	8,19472	3792,04	0,0000
C:Mut_Prob	22,9451	4	5,73628	2654,42	0,0000
D:Pop_Size	4,08417	3	1,36139	629,97	0,0000
E:Restart	25,3836	2	12,6918	5873,03	0,0000
F:Select_Type	19,5282	1	19,5282	9036,53	0,0000
INTERACTIONS					
AB	6,17796	45	0,137288	63,53	0,0000
AC	14,0883	20	0,704415	325,96	0,0000
AD	0,0632651	15	0,00421767	1,95	0,0150
AE	0,903764	10	0,0903764	41,82	0,0000
AF	0,93106	5	0,186212	86,17	0,0000
BC	7,07194	36	0,196443	90,90	0,0000
BD	4,98861	27	0,184763	85,50	0,0000
BE	8,37768	18	0,465427	215,37	0,0000
BF	3,16264	9	0,351404	162,61	0,0000
CD	0,651219	12	0,0542683	25,11	0,0000
CE	0,11873	8	0,0148412	6,87	0,0000
CF	3,81717	4	0,954292	441,59	0,0000
DE	0,152896	6	0,0254827	11,79	0,0000
DF	0,640667	3	0,213556	98,82	0,0000
EF	0,171441	2	0,0857204	39,67	0,0000
RESIDUAL	15,03	6955	0,00216103		
TOTAL (CORRECTED)	216,431	7199			

All F-ratios are based on the residual mean square error.

Tabla 5.2 – Tabla ANOVA con los resultados del experimento del algoritmo GA_{sd} propuesto, experimento SSD10.

Seguiremos el procedimiento ya utilizado anteriormente para analizar los resultados utilizando los valores del “F-Ratio”. A partir de la tabla, el factor más importante es Select_Type, cuyo gráfico de medias aparece en la Figura 5.6.

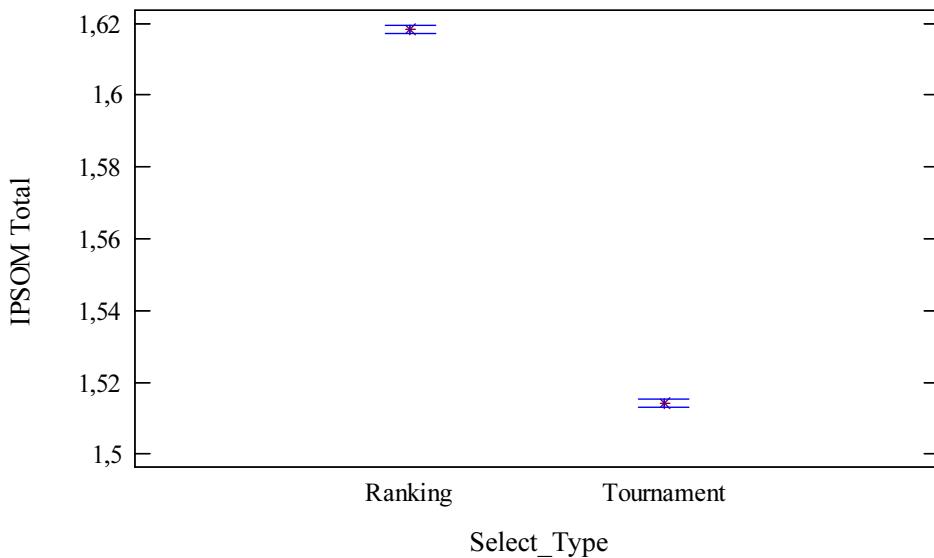


Figura 5.6 – Gráfico de medias e intervalos LSD al 95 % para el factor tipo de selección (Select_Type), experimento SSD10.

De nuevo hay una diferencia muy grande entre los tipos de selección. La selección por torneo binario resulta en un algoritmo mucho mejor que la selección por ranking. El segundo factor por orden de mayor “*F-Ratio*” es el factor Restart, que se muestra en la Figura 5.7.

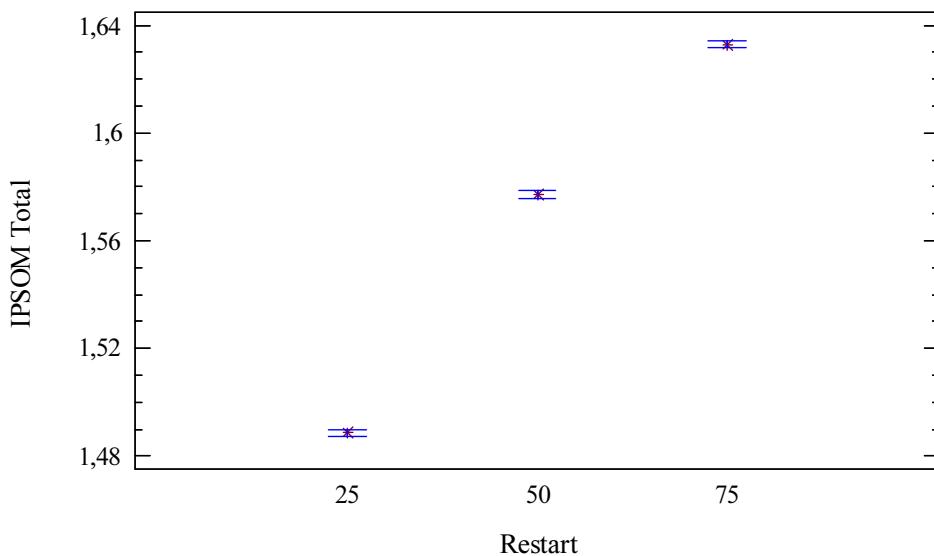


Figura 5.7 – Gráfico de medias e intervalos LSD al 95 % para el factor operador de reinicialización (Restart), experimento SSD10.

Al igual que con el algoritmo GA del capítulo anterior, reiniciar el algoritmo cada 25 generaciones sin mejorar el C_{max} resulta ser la opción más adecuada. El tercer factor a considerar es el Cross_Type:

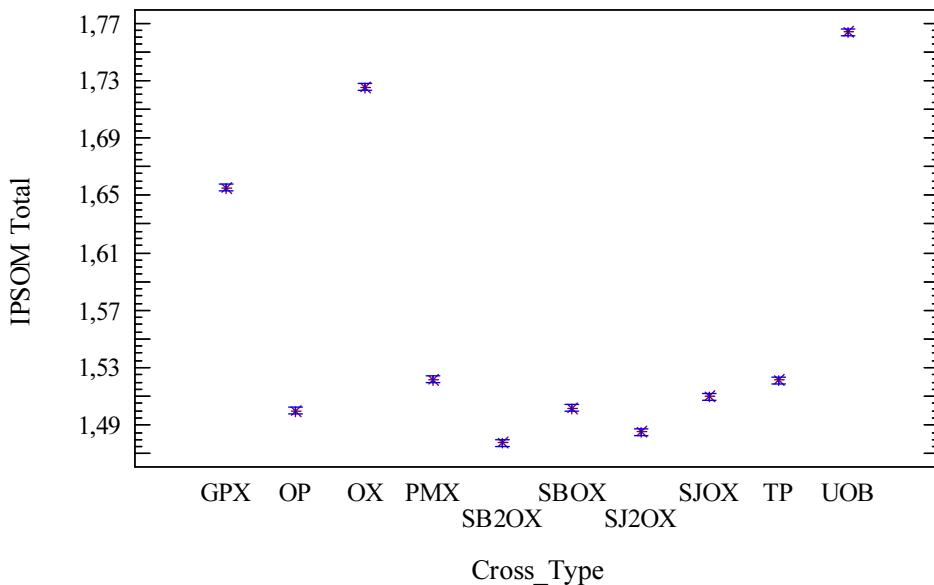


Figura 5.8 – Gráfico de medias e intervalos LSD al 95 % para el factor tipo de cruce (Cross_Type), experimento SSD10.

Podemos ver que las simulaciones iniciales que comentábamos antes sobre el operador GPX no iban desencaminadas, este operador, junto con el OX y UOB son los peores de los 10 considerados. Para poder ver mejor el comportamiento del resto de operadores podemos observar la Figura 5.9 una ampliación de esta gráfica donde se muestran los siete mejores operadores.

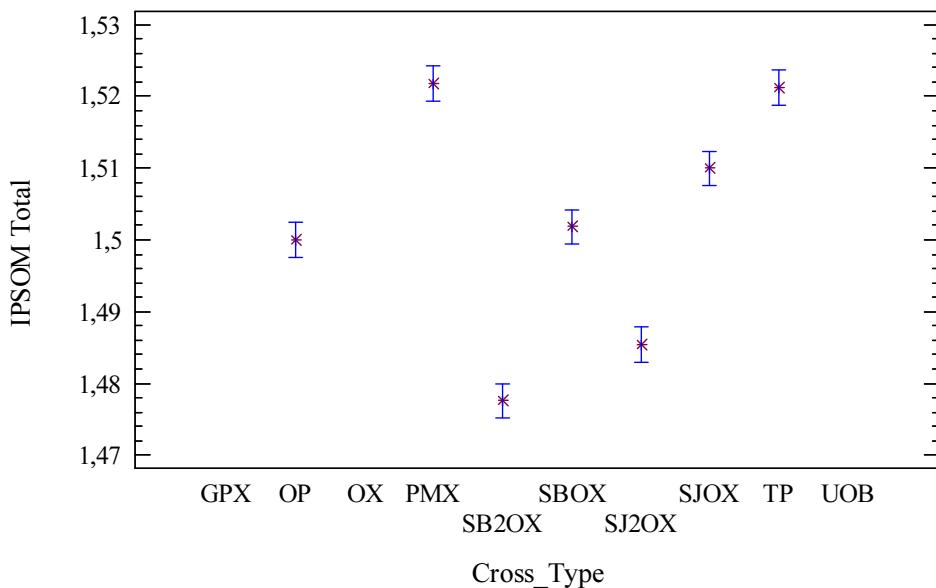


Figura 5.9 – Gráfico ampliado de medias e intervalos LSD al 95 % para el factor tipo de cruce (Cross_Type) donde se observan los mejores operadores, experimento SSD10.

De los siete mejores operadores, los cruces PMX y TP tienen un comportamiento similar, que resulta ser peor que los demás, después viene el operador SJOX. En tercer lugar se encuentra el grupo formado por los cruces SBOX y OP, que tienen un rendimiento parecido. En cuarto lugar tenemos el operador SJ2OX y por último, el mejor operador de los 10 considerados es el SB2OX, donde en este caso la diferencia es clara. Este operador no sólo ha demostrado ser el mejor en el problema del taller de flujo, sino también en el taller SDST, por lo que podemos decir que este operador constituye una importante aportación.

Pasamos ahora a comentar el siguiente operador, la probabilidad de mutación o Mut_Prob (Figura 5.10).

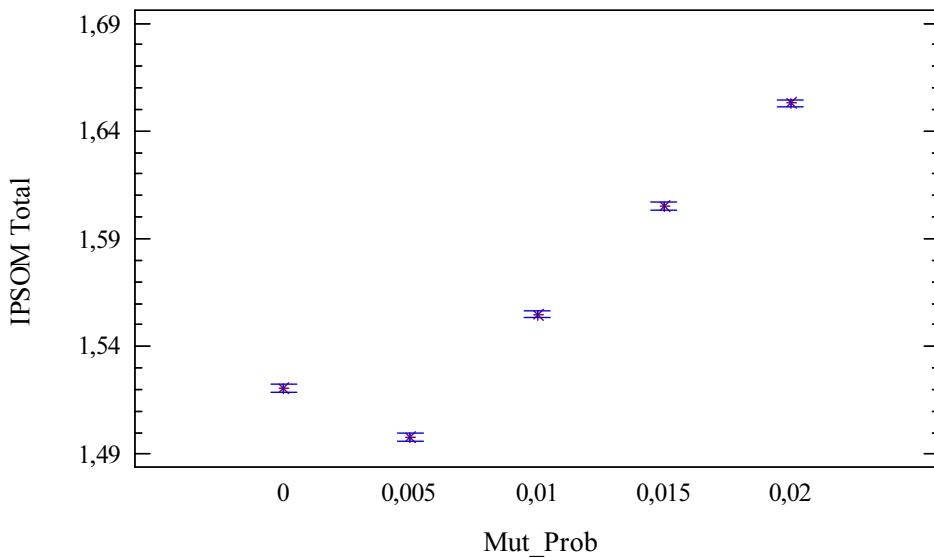


Figura 5.10 – Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de mutación (Mut_Prob), experimento SSD10.

Se puede ver que la probabilidad de mutación tiene un comportamiento similar en este caso que en el taller de flujo estándar. Un P_m de 0 no da buenos resultados, mientras que una probabilidad pequeña (0,005) funciona mejor. El siguiente parámetro que afecta al comportamiento del algoritmo genético propuesto por orden de importancia es en tamaño de la población (P_{size}) o Pop_Size que aparece en la Figura 5.11.

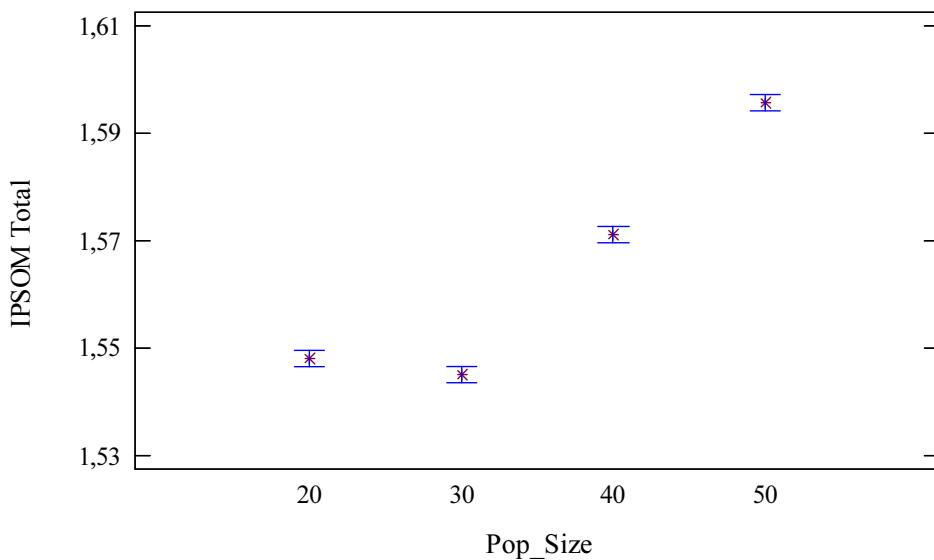


Figura 5.11 – Gráfico de medias e intervalos LSD al 95 % para el factor tamaño de población (Pop_Size), experimento SSD10.

Como vemos en la gráfica, el nivel óptimo para este factor difiere de lo obtenido hasta el momento para el taller de flujo estándar. En este caso un tamaño de población de 20 o 30 resulta ser mejor que el anterior tamaño de 50. Tras examinar el resto de factores e interacciones no fue posible encontrar una diferencia estadísticamente significativa entre los niveles 20 y 30, por ello, elegimos el nivel 30 por ser más cercano al anterior nivel encontrado en el taller de flujo estándar y por resultar en un algoritmo más rápido ya que una población de 30 resulta en menos generaciones del GA. En la lista ordenada por orden de magnitud de “*F-Ratio*”, el siguiente elemento es la interacción entre el tipo de selección y la probabilidad de mutación que se muestra en la Figura 5.12.

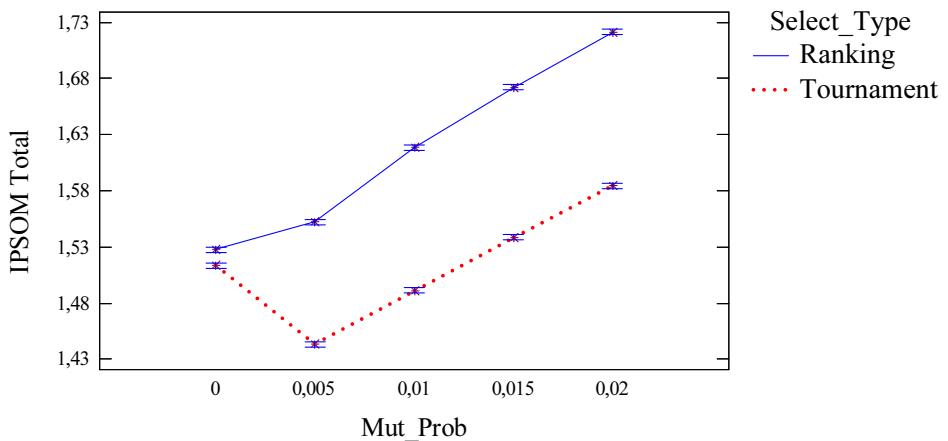


Figura 5.12 – Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores tipo de selección (Select_Type) y probabilidad de mutación (Mut_Prob), experimento SSD10.

El gráfico confirma los anteriores niveles encontrados para la probabilidad de mutación (0,005) y el tipo de selección por torneo binario. Sólo queda el factor de probabilidad de cruce fijar, graficamos sus medias en la Figura 5.13.

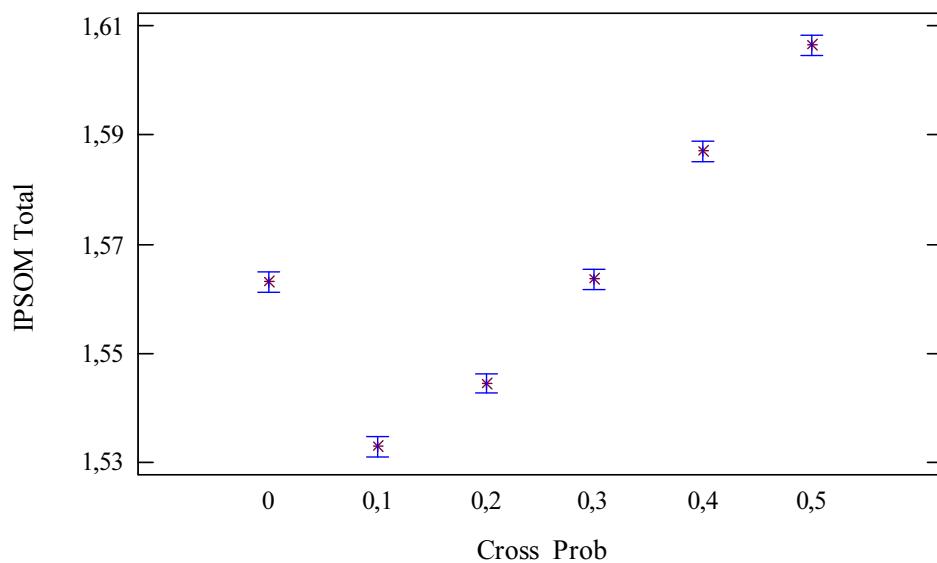


Figura 5.13 – Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de cruce (Cross_Prob), experimento SSD10.

De nuevo, al igual que teníamos con el taller de flujo estándar, una probabilidad de cruce muy baja, aunque no nula resulta lo más indicado.

Los otros 3 experimentos realizados para los grupos de problemas SSD50, SSD100 y SSD125 pueden consultarse en la Sección A.2 del Anexo A, donde se detallan todos los gráficos con las comprobaciones de las hipótesis, tablas ANOVA y los gráficos de medias. El resultado de la parametrización para cada uno de los cuatro grupos de problemas se muestra a continuación en la Tabla 5.3.

Parámetro	Grupo de problemas			
	SSD10	SSD50	SSD100	SSD125
Select	Tournament	Tournament	Tournament	Tournament
Restart	25	25	25	25
Mut_Prob	0,005	0,005	0,005	0,005
Cross_Type	SB2OX	SB2OX	SB2OX	SB2OX
Cross_Prob	0,1	0,1	0,1	0,1
Pop_Size	30	30	30	30

Tournament=Selección por torneo binario, SB2OX=Operador de cruce “*Similar Block 2-Point Order Crossover*”

Tabla 5.3 – Resultado de los experimentos y de la parametrización del algoritmo GA_{sd} propuesto.

Como se puede observar, los experimentos para los grupos de problemas SSD10, SSD50, SSD100 y SSD125 han proporcionado una idéntica parametrización del algoritmo GA_{sd} . Este es un resultado muy bueno, dado que por un lado, todos los parámetros y operadores coinciden, con la única excepción del tamaño de la población, con la parametrización obtenida en el taller de flujo estándar, y por otro lado, el ratio entre los tiempos de proceso y los tiempos de setup no afecta a la parametrización del algoritmo GA_{sd} . Por todo esto, podemos decir que el GA propuesto en el capítulo anterior y la adaptación propuesta para el taller SDST que se ha presentado tienen un comportamiento robusto, dado que los mejores niveles o variantes de los distintos parámetros y operadores se mantienen constantes. Esta propiedad es muy deseable en los GAs, dado que nos aseguramos que tras la parametrización hemos obtenido el mejor GA posible con independencia del problema o ejemplo concreto a resolver. Este resultado no es el mismo que se observa en los trabajos publicados, dado que las heurísticas y métodos presentados por varios autores, como por ejemplo Simons (1992), Das, Gupta y Khumawala (1995) o Ríos-Mercado y Bard (1999a), son mucho más sensibles a la magnitud de los tiempos de cambio de partida.

5.6. Adaptación del HGA al taller SDST

En el capítulo anterior se propuso una hibridación del GA mediante un tipo de búsqueda local. Esta hibridación resultó ser muy eficaz y por ello proponemos una aplicación del mismo esquema híbrido al algoritmo GA_{sd} presentado anteriormente. El nuevo GA híbrido resultante se denotará por HGA_{sd} .

Utilizaremos la probabilidad de aplicar la hibridación o probabilidad de mejora P_{enh} , que tiene idéntico funcionamiento al comentado en la Sección 4.4. De nuevo, y para no complicar demasiado el funcionamiento del algoritmo HGA_{sd} , la hibridación se hace con un método de búsqueda local descendente. Este método es muy parecido al utilizado en el HGA del capítulo anterior. Recordemos que éste se basa en la aplicación de una búsqueda local en el vecindario de inserción I , donde en una secuencia completa se extrae un trabajo al azar y se evaluaban las n posibles secuencias resultado de insertar el trabajo extraído en todas las posibles n posiciones. Para evaluar las n posibles secuencias se utiliza la aceleración de Taillard (1990) propuesta para la heurística NEH y que resulta muy rápida. El número de veces que se aplica este procedimiento se denotó por Enh_s , con lo que con ambos parámetros podemos controlar el nivel de hibridación del algoritmo HGA_{sd} . Como se ha comentado, las mejoras de Taillard (1990) no contemplan la consideración de los tiempos de cambio de partida dependientes de la secuencia, para ello hacemos uso de la extensión propuesta por Ríos-Mercado y Bard (1998b).

Para este algoritmo también utilizamos la técnica de mejorar el individuo con menor C_{max} de la población en cada generación con una probabilidad igual a $2 \cdot P_{enh}$. Asimismo, para evitar mejorar siempre los mismos individuos usamos el sistema de turnos comentado en el capítulo anterior. De esta manera, el algoritmo híbrido HGA_{sd} es muy parecido al algoritmo HGA propuesto para el taller de flujo estándar, al que se añaden los tiempos de cambio para el taller SDST. Tras realizar experimentos iniciales, determinamos que de nuevo, la mejor combinación de los parámetros P_{enh} y Enh_s , desde el punto de vista de las prestaciones frente al tiempo de computación, es de $P_{enh}=0,05$ y $\text{Enh}_s=25$.

5.7. Evaluación comparativa de heurísticas y metaheurísticas para el taller SDST

En esta sección vamos a realizar una extensa evaluación de métodos heurísticos y metaheurísticos para el problema del taller de flujo. Vamos a hacer dos clasificaciones: los métodos heurísticos y metaheurísticos propuestos específicamente para el taller SDST y los que se propusieron para el taller de flujo general pero que hemos adaptado al taller SDST.

En el primer grupo tenemos la heurística NEH con las mejoras de Taillard y la extensión al taller SDST de Ríos-Mercado y Bard (1998b), que llamaremos NEHT_RMB, las heurísticas S_TOTAL y S_SETUP de Simons (1992), el método RMB_HYB de Ríos-Mercado y Bard (1999a), la heurística SI_DGK de Das, Gupta y Khumawala (1995) y la metaheurística RMB_GRASP también de Ríos-Mercado y Bard (1998b). No se han propuesto, de manera general y que nosotros sepamos, otros métodos para el taller SDST aparte de estos seis que aquí evaluamos. En el segundo grupo hemos realizado adaptaciones de la regla aleatoria RANDOM presentada en el Capítulo 3 para la consideración de tiempos de cambio. De igual manera hemos adaptado las siguientes metaheurísticas, que por otra parte fueron las que demostraron un mejor rendimiento para el taller de flujo general: Método ILS de Stützle (1998), donde la inicialización y la fase de búsqueda local se hacen con la heurística NEHT_RMB y en la evaluación de las soluciones se consideran los tiempos de cambio. El GA de Reeves (1995) (GAReev), donde la adaptación realizada afecta a la inicialización (ahora por NEHT_RMB) y a la evaluación de los individuos en cada generación. El SA de Osman y Potts (1989), que también se ha adaptado, aunque en este caso, al ser la inicialización aleatoria sólo ha sido necesario cambiar la función de evaluación para considerar tiempos de cambio. Denotaremos a este algoritmo por SAOP. El tabu search de Widmer y Hertz (1989) (Spirit) también se ha adaptado. La inicialización propuesta por los autores no considera para nada la existencia de tiempos de cambio de partida así que proponemos una inicialización por la heurística NEHT_RMB, dado que una inicialización aleatoria demostró un comportamiento muy pobre en simulaciones iniciales.

Por último, en la evaluación incluimos los dos algoritmos genéticos que hemos propuesto para el taller SDST, GA_{sd} y HGA_{sd} . Para la realización de las pruebas se ha utilizado el mismo ordenador AMD Athlon 1600+XP (1400 MHz) con 512 Mbytes de memoria RAM que en capítulos anteriores. El criterio de parada inicialmente considerado es la evaluación de 10.000 C_{max} . Hay que tener en cuenta que todos los métodos metaheurísticos menos el *Spirit* (ILS, SAOP, GAReev, RMB_GRASP, GA_{sd} y HGA_{sd}) y la regla RAND son estocásticos y por tanto se han ejecutado cinco veces de manera independiente y se han calculado las medias de los resultados. Recordemos que la variable respuesta es el *IPSOM* total. Los conjuntos de pruebas utilizados para la evaluación son los 480 problemas con tiempos de cambio de partida que se utilizaron en la parametrización de los GAs propuestos. Realizaremos el análisis de los resultados separando cada uno de los cuatro grupos de problemas. Comenzando con el grupo SSD10, los *IPSOM* totales, así como los tiempos de CPU requeridos se muestran en las Tablas 5.4 y 5.5.

Problema	RANDOM	ILS	SAOP	Spirit	GAReev	GA _{sd}	HGA _{sd}
20x5	8,45 (<0,5)	0,76 (0,54)	2,08 (<0,5)	2,03 (<0,5)	1,23 (<0,5)	0,41 (<0,5)	0,09 (2,68)
20x10	10,09 (<0,5)	1,28 (0,93)	2,79 (<0,5)	2,73 (<0,5)	1,90 (<0,5)	0,52 (0,51)	0,18 (2,66)
20x20	8,64 (<0,5)	1,31 (1,21)	2,85 (<0,5)	2,68 (<0,5)	1,65 (<0,5)	0,54 (0,71)	0,17 (3,43)
50x5	9,32 (<0,5)	1,44 (1,09)	2,21 (<0,5)	1,74 (<0,5)	1,38 (<0,5)	0,83 (0,61)	0,23 (3,96)
50x10	13,58 (<0,5)	2,22 (2,31)	3,75 (<0,5)	2,99 (<0,5)	2,28 (0,59)	1,29 (0,94)	0,45 (4,82)
50x20	14,32 (0,58)	2,40 (3,95)	3,27 (0,61)	2,92 (0,58)	2,56 (0,90)	1,29 (1,75)	0,33 (8,71)
100x5	8,89 (<0,5)	1,64 (2,79)	2,21 (<0,5)	2,19 (<0,5)	1,69 (0,97)	1,25 (1,02)	0,29 (6,34)
100x10	12,50 (1,25)	1,87 (4,78)	3,45 (0,63)	2,56 (0,60)	2,04 (1,31)	1,28 (2,00)	0,32 (11,49)
100x20	14,38 (3,95)	2,29 (13,23)	4,26 (1,35)	2,85 (1,32)	2,35 (2,13)	1,58 (5,13)	0,51 (27,21)
200x10	11,19 (4,85)	1,62 (16,74)	3,43 (1,49)	1,94 (1,57)	1,66 (3,87)	1,37 (6,44)	0,50 (28,61)
200x20	13,65 (11,82)	1,67 (39,57)	4,52 (10,68)	2,16 (11,03)	1,90 (13,49)	1,39 (25,35)	0,50 (61,54)
500x20	10,86 (38,75)	0,85 (114,59)	4,21 (31,13)	1,06 (32,62)	0,95 (45,36)	0,77 (141,21)	0,37 (177,93)
Media	11,32 (5,19)	1,61 (16,81)	3,25 (3,92)	2,32 (4,06)	1,80 (5,83)	1,04 (15,51)	0,33 (28,28)

RANDOM=Regla aleatoria, ILS=Búsqueda local iterativa de Stützle (1998) (adaptación), SAOP=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev=GA de Reeves (1995) (adaptación), GA_{sd}=GA propuesto, HGA_{sd}=GA híbrido propuesto

Tabla 5.4 – IPSOM y tiempo de CPU en segundos (entre paréntesis) de los métodos metaheurísticos adaptados y los nuevos algoritmos genéticos propuestos para el taller SDST, grupo de problemas SSD10.

Problema	NEHT_RMB	S_TOTAL	S_SETUP	RMB_HYB	RMB_GRASP	SI_DGK
20x5	3,77 (<0,5)	25,10 (<0,5)	23,00 (<0,5)	15,62 (<0,5)	1,06 (2,68)	22,14 (<0,5)
20x10	4,31 (<0,5)	24,35 (<0,5)	27,89 (<0,5)	18,25 (<0,5)	1,33 (3,83)	21,44 (<0,5)
20x20	3,55 (<0,5)	19,35 (<0,5)	19,35 (<0,5)	14,29 (<0,5)	1,25 (6,49)	17,37 (<0,5)
50x5	2,84 (<0,5)	20,24 (<0,5)	18,22 (<0,5)	14,54 (<0,5)	1,73 (15,49)	21,33 (1,01)
50x10	4,14 (<0,5)	23,85 (<0,5)	23,34 (<0,5)	17,95 (<0,5)	2,22 (24,27)	23,01 (2,02)
50x20	3,83 (<0,5)	22,44 (<0,5)	23,99 (<0,5)	19,31 (<0,5)	2,20 (45,84)	21,37 (4,12)
100x5	2,47 (<0,5)	17,00 (<0,5)	16,92 (<0,5)	12,29 (<0,5)	1,75 (65,64)	19,43 (16,30)
100x10	2,94 (<0,5)	20,57 (<0,5)	20,63 (<0,5)	16,74 (<0,5)	1,81 (119,32)	20,74 (33,02)
100x20	3,13 (<0,5)	21,06 (<0,5)	21,48 (<0,5)	17,96 (<0,5)	2,14 (269,52)	18,11 (68,84)
200x10	2,04 (<0,5)	18,49 (<0,5)	18,19 (<0,5)	13,82 (1,73)	1,50 (706,63)	17,35 (603,74)
200x20	2,28 (<0,5)	18,21 (<0,5)	19,21 (<0,5)	16,52 (2,16)	1,60 (1917,72)	17,38 (2994,50)
500x20	1,09 (1,53)	14,57 (2,49)	14,77 (2,50)	12,77 (41,70)	(*)	(*)
Media	3,03 (<0,5)	20,43 (<0,5)	20,58 (<0,5)	15,84 (3,88)	1,69 (288,86)	19,97 (338,52)

NEHT_RMB=Algoritmo basado en NEH de Ríos-Mercado y Bard (1998b), S_TOTAL=Heurística TOTAL de Simons (1992), S_SETUP=Heurística SETUP de Simons (1992), RMB_HYB=Algoritmo de Ríos-Mercado y Bard (1999a), RMB_GRASP=GRASP de Ríos-Mercado y Bard (1998b), SI_DGK=Algoritmo de Das, Gupta y Khumawala (1995)

Tabla 5.5 – IPSOM y tiempo de CPU en segundos (entre paréntesis) de los métodos heurísticos y metaheurísticos para el taller SDST, grupo de problemas SSD10.

Resulta muy llamativo que los métodos heurísticos propuestos específicamente para el taller SDST tienen un rendimiento muy pobre. Por ejemplo, las heurísticas S_TOTAL y S_SETUP resultan ser muy rápidas, necesitando menos de medio segundo de tiempo de CPU de media y alrededor de dos segundos y medio para los problemas más grandes, pero su incremento con respecto a la mejor solución sobrepasa el 20 %. Este primer resultado es muy pobre, sobre todo teniendo en cuenta que la regla RANDOM, que necesita poco más de cinco segundos de media y algo más de medio minuto para los problemas más grandes, tiene un *IPSOM* de algo más del 11 %. Claramente no parece justificado utilizar estas heurísticas, dado que en unos pocos segundos más podemos encontrar una mejor solución con la regla RANDOM que además es mucho más sencilla de implementar. En una situación peor si cabe están las heurísticas RMB_HYB y SI_DGK. El método RMB_HYB obtiene un *IPSOM* total de casi un 16 % y en un tiempo medio de algo menos de cuatro segundos, mientras que la heurística SI_DGK obtiene un *IPSOM* total de casi un 20 %. Si tenemos en cuenta que el tiempo medio para obtener estos resultados es de casi seis minutos llegamos a la conclusión de que este método no es aconsejable. Además, los tiempos de CPU que esta heurística necesita crecen muy rápido y para los problemas de 200 trabajos y 20 máquinas este método necesitó casi 50 minutos, de hecho, se estimó que el tiempo necesario para resolver los problemas de 500 trabajos y 20 máquinas era superior a las cinco horas, de ahí que no se obtuviesen resultados para este grupo de problemas.

En una mejor situación está la metaheurística RMB_GRASP, donde el *IPSOM* total obtenido es del 1,69 %, lo cual es un buen resultado, pero no tanto si consideramos que los tiempos de CPU pasan de cinco minutos de media y más de media hora para los problemas de 200 trabajos y 20 máquinas. De nuevo, y para este método, no se pudieron resolver los problemas de 500 trabajos y 20 máquinas. El método NEH_RMB obtiene un *IPSOM* de poco más del 3 %, resultado que se calcula en menos de medio segundo de media, lo que lo convierte en el método más rápido de la evaluación.

En cuanto a los métodos metaheurísticos adaptados, podemos comenzar el análisis con el SAOP, que obtiene un *IPSOM* total de 3,25 % en unos tiempos bastante cortos (menos de 4 segundos de media). El método Spirit se ha bene-

ficiado enormemente de la nueva inicialización. Vimos en el Capítulo 3 como este algoritmo resultaba tener un rendimiento muy pobre y atribuimos estos resultados a una inicialización no demasiado buena, también vimos como el algoritmo era de convergencia muy lenta. Con la nueva inicialización, el método consigue un *IPSOM* total de poco más del 2 %. Otro algoritmo que se ha adaptado muy bien al nuevo problema es el GAREEV, con un *IPSOM* total de menos del 2 %, resultados que obtiene en menos de seis segundos de media. La adaptación del método ILS también ha proporcionado muy buenos resultados, menores que la metaheurística RMB_GRASP específicamente diseñada para el taller SDST y en unos tiempos mucho menores (algo menos de 17 segundos de media).

Los resultados obtenidos por los dos GAs propuestos, GA_{sd} y HGA_{sd} indican que son los que presentan mejor comportamiento de la comparativa. El algoritmo GA_{sd} obtiene un *IPSOM* total del 1,04 %, que es un 55 % mejor que el método ILS, al tiempo que los requerimientos computacionales no llegan a los 16 segundos de media. Si comparamos detenidamente ambos métodos, obtenemos los resultados que se muestran en la Tabla 5.6.

Problema	Número de veces solución mejor por ILS	Número de veces solución igual entre ILS y GA_{sd}	Número de veces solución mejor por GA_{sd}
20x5	2	0	8
20x10	0	1	9
20x20	0	0	10
50x5	0	0	10
50x10	0	0	10
50x20	0	0	10
100x5	0	0	10
100x10	0	0	10
100x20	0	0	10
200x10	1	0	9
200x20	1	0	9
500x20	2	0	8
Total	6	1	114

ILS=Búsqueda local iterativa de Stützle (1998) (adaptación), GA_{sd}=GA propuesto

Tabla 5.6 – Comparación entre los métodos metaheurísticos ILS y GA_{sd} para el grupo de problemas SSD10.

Los resultados indican que el método GA_{sd} es bastante mejor que el ILS, dado que de los 120 problemas resueltos ha obtenido una mejor solución en 114 casos. Este es un buen resultado para un algoritmo que no incorpora ningún tipo de búsqueda local.

Como cabía esperar, el algoritmo HGA_{sd} se comporta mejor, con un IPSOM total del 0,33 %, lo que representa una mejora del 488 % con respecto al método ILS. En este caso el tiempo de CPU que se requiere está cercano al medio minuto de media y algo menos de tres minutos para los problemas de 500 trabajos, superior a los métodos ILS o GAReev pero claramente inferior a otros métodos como el RMB_GRASP o SI_DGK.

Los resultados para los grupos de problemas SSD50, SSD100 y SSD125 se muestran de la Tabla 5.7 a la Tabla 5.12.

Problema	RANDOM	ILS	SAOP	Spirit	GAReev	GA _{sd}	HGA _{sd}
20x5	13,96 (<0,5)	2,50 (0,66)	4,55 (<0,5)	4,98 (<0,5)	3,32 (<0,5)	1,00 (<0,5)	0,50 (2,37)
20x10	11,94 (<0,5)	2,17 (0,83)	4,68 (<0,5)	4,67 (<0,5)	2,93 (<0,5)	1,09 (<0,5)	0,32 (2,50)
20x20	9,06 (<0,5)	1,66 (1,28)	3,02 (<0,5)	3,62 (<0,5)	2,06 (<0,5)	0,42 (0,67)	0,18 (3,42)
50x5	20,05 (<0,5)	5,40 (1,23)	5,93 (<0,5)	5,73 (<0,5)	4,56 (<0,5)	3,03 (0,61)	0,92 (3,50)
50x10	17,77 (<0,5)	4,71 (2,05)	5,53 (<0,5)	5,33 (<0,5)	4,18 (0,58)	2,77 (0,92)	1,13 (4,63)
50x20	15,22 (0,60)	4,12 (3,59)	4,38 (0,61)	4,58 (0,57)	3,86 (0,89)	1,97 (1,71)	0,61 (10,08)
100x5	21,39 (<0,5)	5,53 (3,09)	7,16 (<0,5)	5,37 (<0,5)	4,92 (0,97)	3,81 (1,17)	1,28 (6,33)
100x10	18,65 (1,20)	4,81 (5,47)	6,67 (0,62)	4,76 (0,60)	4,30 (1,31)	3,14 (2,32)	1,18 (11,40)
100x20	16,05 (3,74)	3,99 (12,90)	5,75 (1,35)	3,96 (1,31)	3,58 (2,11)	2,47 (5,94)	0,89 (27,53)
200x10	17,89 (4,64)	3,59 (15,84)	7,13 (1,46)	3,58 (1,57)	3,38 (3,79)	2,71 (9,16)	1,29 (27,83)
200x20	15,84 (11,38)	3,18 (37,91)	6,60 (10,11)	3,19 (10,48)	3,00 (12,95)	2,17 (29,49)	1,15 (60,42)
500x20	13,59 (37,65)	1,52 (110,23)	6,50 (29,83)	1,51 (31,39)	1,47 (44,01)	0,96 (167,06)	0,75 (193,20)
Media	15,95 (5,02)	3,60 (16,26)	5,66 (3,76)	4,27 (3,91)	3,46 (5,66)	2,13 (18,33)	0,85 (29,44)

RANDOM=Regla aleatoria, ILS=Búsqueda local iterativa de Stützle (1998) (adaptación), SAOP=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev=GA de Reeves (1995) (adaptación), GA_{sd}=GA propuesto, HGA_{sd}=GA híbrido propuesto

Tabla 5.7 – IPSOM y tiempo de CPU en segundos (entre paréntesis) de los métodos metaheurísticos adaptados y los algoritmos genéticos propuestos para el taller SDST, grupo de problemas SSD50.

Problema	NEHT_RMB	S_TOTAL	S_SETUP	RMB_HYB	RMB_GRASP	SI_DGK
20x5	6,53 (<0,5)	27,68 (<0,5)	25,57 (<0,5)	20,75 (<0,5)	2,81 (2,67)	18,38 (<0,5)
20x10	6,30 (<0,5)	27,00 (<0,5)	24,88 (<0,5)	18,35 (<0,5)	2,31 (3,93)	18,29 (<0,5)
20x20	4,09 (<0,5)	19,09 (<0,5)	19,83 (<0,5)	15,39 (<0,5)	1,45 (6,34)	15,34 (<0,5)
50x5	6,77 (<0,5)	31,47 (<0,5)	32,28 (<0,5)	25,29 (<0,5)	4,61 (15,52)	16,22 (1,01)
50x10	6,14 (<0,5)	27,77 (<0,5)	27,88 (<0,5)	23,74 (<0,5)	4,22 (24,18)	17,74 (2,00)
50x20	4,92 (<0,5)	25,45 (<0,5)	23,19 (<0,5)	19,87 (<0,5)	3,16 (46,06)	17,46 (4,13)
100x5	5,76 (<0,5)	29,39 (<0,5)	30,66 (<0,5)	25,36 (<0,5)	4,63 (66,78)	13,70 (16,21)
100x10	5,13 (<0,5)	25,90 (<0,5)	27,15 (<0,5)	23,02 (<0,5)	3,90 (120,12)	15,63 (32,96)
100x20	4,21 (<0,5)	23,07 (<0,5)	21,81 (<0,5)	19,15 (<0,5)	3,12 (268,32)	13,73 (68,66)
200x10	3,66 (<0,5)	23,76 (<0,5)	24,02 (<0,5)	20,88 (1,73)	3,17 (685,40)	11,00 (602,84)
200x20	3,25 (<0,5)	20,54 (<0,5)	20,29 (<0,5)	17,92 (2,22)	2,52 (1657,36)	11,05 (3018,58)
500x20	1,52 (1,48)	17,01 (2,35)	17,61 (2,34)	15,15 (39,99)	(*)	(*)
Media	4,86 (<0,5)	24,85 (<0,5)	24,60 (<0,5)	20,41 (3,74)	3,26 (263,33)	15,32 (340,60)

NEHT_RMB=Algoritmo basado en NEH de Ríos-Mercado y Bard (1998b), S_TOTAL=Heurística TOTAL de Simons (1992), S_SETUP=Heurística SETUP de Simons (1992), RMB_HYB=Algoritmo de Ríos-Mercado y Bard (1999a), RMB_GRASP=GRASP de Ríos-Mercado y Bard (1998b), SI_DGK=Algoritmo de Das, Gupta y Khumawala (1995)

Tabla 5.8 – IPSOM y tiempo de CPU en segundos (entre paréntesis) de los métodos heurísticos y metaheurísticos para el taller SDST, grupo de problemas SSD50.

Problema	RANDOM	ILS	SAOP	Spirit	GAReev	GA _{sd}	HGA _{sd}
20x5	18,63 (<0,5)	5,00 (<0,5)	7,42 (<0,5)	8,52 (<0,5)	5,78 (<0,5)	2,04 (<0,5)	0,60 (2,23)
20x10	14,26 (<0,5)	3,74 (0,64)	6,17 (<0,5)	5,75 (<0,5)	4,54 (<0,5)	1,66 (<0,5)	0,43 (2,61)
20x20	10,22 (<0,5)	2,47 (1,03)	4,02 (<0,5)	4,28 (<0,5)	3,10 (<0,5)	1,01 (0,68)	0,27 (3,55)
50x5	29,06 (<0,5)	8,89 (1,27)	9,26 (<0,5)	8,57 (<0,5)	7,33 (<0,5)	4,46 (0,61)	1,64 (3,86)
50x10	22,35 (<0,5)	6,65 (1,77)	7,42 (<0,5)	6,29 (<0,5)	5,85 (0,58)	3,63 (0,93)	1,49 (4,68)
50x20	16,73 (0,59)	4,88 (3,45)	5,58 (0,62)	4,88 (0,57)	4,46 (0,90)	2,63 (1,75)	0,77 (8,61)
100x5	32,57 (<0,5)	8,38 (2,86)	11,34 (<0,5)	8,16 (<0,5)	7,71 (0,97)	5,56 (1,17)	1,91 (5,25)
100x10	23,65 (1,20)	6,12 (5,06)	9,06 (0,62)	6,01 (0,60)	5,44 (1,30)	3,96 (2,46)	1,54 (11,37)
100x20	18,56 (3,75)	5,38 (12,61)	7,53 (1,35)	5,30 (1,31)	4,97 (2,11)	3,23 (6,13)	1,43 (26,17)
200x10	24,24 (4,63)	5,12 (16,53)	10,76 (1,50)	5,09 (1,63)	4,90 (3,79)	3,71 (9,65)	2,03 (28,45)
200x20	18,61 (11,41)	3,43 (38,21)	8,48 (10,00)	3,42 (10,48)	3,32 (12,97)	2,56 (30,55)	1,55 (63,26)
500x20	16,83 (37,56)	1,71 (110,05)	8,75 (29,79)	1,70 (31,22)	1,68 (43,85)	1,05 (168,33)	0,91 (200,33)
Media	20,48 (5,01)	5,15 (16,16)	7,98 (3,75)	5,66 (3,90)	4,92 (5,65)	2,96 (18,60)	1,21 (30,03)

RANDOM=Regla aleatoria, ILS=Búsqueda local iterativa de Stützle (1998) (adaptación), SAOP=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev=GA de Reeves (1995) (adaptación), GA_{sd}=GA propuesto, HGA_{sd}=GA híbrido propuesto

Tabla 5.9 – IPSOM y tiempo de CPU en segundos (entre paréntesis) de los métodos metaheurísticos adaptados y los algoritmos genéticos propuestos para el taller SDST, grupo de problemas SSD100.

Problema	NEHT_RMB	S_TOTAL	S_SETUP	RMB_HYB	RMB_GRASP	SI_DGK
20x5	10,18 (<0,5)	36,76 (<0,5)	34,86 (<0,5)	27,97 (<0,5)	4,04 (2,89)	14,21 (<0,5)
20x10	7,23 (<0,5)	28,50 (<0,5)	29,05 (<0,5)	22,34 (<0,5)	3,04 (4,34)	16,28 (<0,5)
20x20	5,32 (<0,5)	21,67 (<0,5)	21,80 (<0,5)	16,20 (<0,5)	2,14 (7,22)	13,97 (<0,5)
50x5	10,00 (<0,5)	43,50 (<0,5)	43,50 (<0,5)	36,40 (<0,5)	6,79 (17,59)	12,60 (1,01)
50x10	7,33 (<0,5)	34,38 (<0,5)	34,04 (<0,5)	28,38 (<0,5)	5,49 (27,09)	14,49 (2,01)
50x20	5,46 (<0,5)	25,45 (<0,5)	25,32 (<0,5)	21,37 (<0,5)	3,77 (52,23)	14,93 (4,13)
100x5	8,56 (<0,5)	43,14 (<0,5)	42,87 (<0,5)	37,41 (<0,5)	6,97 (75,45)	8,02 (16,32)
100x10	6,22 (<0,5)	33,27 (<0,5)	32,51 (<0,5)	27,93 (<0,5)	4,97 (137,38)	11,38 (33,07)
100x20	5,50 (<0,5)	25,65 (<0,5)	24,29 (<0,5)	22,02 (<0,5)	4,03 (316,33)	11,09 (68,72)
200x10	5,13 (<0,5)	31,43 (<0,5)	30,99 (<0,5)	27,57 (1,74)	4,41 (822,37)	6,68 (602,12)
200x20	3,44 (<0,5)	24,23 (<0,5)	24,05 (<0,5)	21,02 (2,19)	3,16 (2064,65)	7,80 (3130,76)
500x20	1,71 (1,47)	20,20 (2,33)	20,59 (2,32)	18,70 (39,59)	(*)	(*)
Media	6,34 (<0,5)	30,68 (<0,5)	30,32 (<0,5)	25,61 (3,71)	4,44 (320,68)	11,95 (350,76)

NEHT_RMB=Algoritmo basado en NEH de Ríos-Mercado y Bard (1998b), S_TOTAL=Heurística TOTAL de Simons (1992), S_SETUP=Heurística SETUP de Simons (1992), RMB_HYB=Algoritmo de Ríos-Mercado y Bard (1999a), RMB_GRASP=GRASP de Ríos-Mercado y Bard (1998b), SI_DGK=Algoritmo de Das, Gupta y Khumawala (1995)

Tabla 5.10 – IPSOM y tiempo de CPU en segundos (entre paréntesis) de los métodos heurísticos y metaheurísticos para el taller SDST, grupo de problemas SSD100.

Problema	RANDOM	ILS	SAOP	Spirit	GAReev	GA _{sd}	HGA _{sd}
20x5	20,87 (<0,5)	5,02 (0,53)	8,39 (<0,5)	9,30 (<0,5)	6,06 (<0,5)	2,20 (<0,5)	0,91 (2,27)
20x10	15,06 (<0,5)	3,77 (0,83)	6,23 (<0,5)	5,54 (<0,5)	4,34 (<0,5)	1,69 (0,52)	0,39 (2,42)
20x20	10,45 (<0,5)	2,34 (1,02)	4,52 (<0,5)	3,79 (<0,5)	3,31 (<0,5)	1,13 (0,71)	0,27 (3,50)
50x5	33,30 (<0,5)	10,12 (1,04)	10,73 (<0,5)	9,93 (<0,5)	8,39 (<0,5)	5,62 (0,64)	2,17 (3,47)
50x10	23,91 (<0,5)	7,52 (1,72)	8,33 (<0,5)	7,64 (<0,5)	6,90 (0,57)	4,11 (0,98)	1,35 (4,66)
50x20	18,03 (0,58)	5,87 (3,26)	6,26 (0,60)	5,60 (0,57)	5,18 (0,89)	3,24 (1,86)	1,15 (8,71)
100x5	36,33 (<0,5)	9,35 (2,51)	12,51 (<0,5)	9,12 (<0,5)	8,52 (0,96)	5,95 (1,25)	2,19 (5,36)
100x10	25,93 (1,21)	6,64 (4,95)	9,98 (0,62)	6,38 (0,59)	5,97 (1,30)	4,30 (2,52)	1,60 (10,14)
100x20	19,44 (3,86)	5,28 (13,07)	7,67 (1,35)	5,08 (1,31)	4,85 (2,11)	3,38 (6,64)	1,36 (27,21)
200x10	26,86 (4,69)	5,48 (16,94)	11,96 (1,44)	5,45 (1,56)	5,26 (3,80)	4,00 (10,39)	2,30 (27,75)
200x20	19,63 (11,25)	4,08 (38,35)	8,96 (9,85)	4,00 (10,32)	3,85 (12,76)	2,71 (30,90)	1,54 (62,28)
500x20	18,09 (37,29)	1,82 (109,81)	9,54 (29,50)	1,81 (31,03)	1,79 (43,59)	1,14 (168,19)	0,95 (200,58)
Media	22,33 (4,99)	5,61 (16,17)	8,76 (3,71)	6,14 (3,87)	5,37 (5,61)	3,29 (18,75)	1,35 (29,86)

RANDOM=Regla aleatoria, ILS=Búsqueda local iterativa de Stützle (1998) (adaptación), SAOP=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev=GA de Reeves (1995) (adaptación), GA_{sd}=GA propuesto, HGA_{sd}=GA híbrido propuesto

Tabla 5.11 – IPSOM y tiempo de CPU en segundos (entre paréntesis) de los métodos metaheurísticos adaptados y los algoritmos genéticos propuestos para el taller SDST, grupo de problemas SSD125.

Problema	NEHT_RMB	S_TOTAL	S_SETUP	RMB_HYB	RMB_GRASP	SI_DGK
20x5	10,61 (<0,5)	40,57 (<0,5)	40,96 (<0,5)	30,30 (<0,5)	4,68 (2,49)	13,32 (<0,5)
20x10	7,31 (<0,5)	29,53 (<0,5)	29,37 (<0,5)	22,88 (<0,5)	3,19 (3,91)	15,10 (<0,5)
20x20	5,23 (<0,5)	21,94 (<0,5)	20,52 (<0,5)	15,81 (<0,5)	2,30 (6,43)	12,72 (<0,5)
50x5	11,31 (<0,5)	48,47 (<0,5)	49,03 (<0,5)	40,46 (<0,5)	8,09 (15,77)	11,65 (1,05)
50x10	8,19 (<0,5)	36,29 (<0,5)	35,81 (<0,5)	29,42 (<0,5)	6,01 (24,35)	13,83 (2,01)
50x20	6,88 (<0,5)	27,00 (<0,5)	26,56 (<0,5)	23,13 (<0,5)	4,60 (46,15)	14,02 (4,12)
100x5	9,54 (<0,5)	48,64 (<0,5)	48,95 (<0,5)	42,19 (<0,5)	7,61 (67,18)	6,80 (16,38)
100x10	6,86 (<0,5)	35,03 (<0,5)	36,05 (<0,5)	30,17 (<0,5)	5,57 (119,95)	10,27 (33,12)
100x20	5,43 (<0,5)	25,74 (<0,5)	26,26 (<0,5)	22,52 (<0,5)	4,09 (266,98)	9,94 (68,81)
200x10	5,49 (<0,5)	34,27 (<0,5)	34,18 (<0,5)	30,03 (1,71)	4,83 (687,60)	5,75 (606,56)
200x20	4,08 (<0,5)	23,99 (<0,5)	24,69 (<0,5)	22,69 (2,16)	3,29 (1701,24)	6,65 (3063,92)
500x20	1,82 (1,47)	21,48 (2,34)	21,75 (2,33)	20,08 (39,86)	(*)	(*)
Media	6,90 (<0,5)	32,75 (<0,5)	32,84 (<0,5)	27,47 (3,73)	4,93 (267,46)	10,91 (345,11)

NEHT_RMB=Algoritmo basado en NEH de Ríos-Mercado y Bard (1998b), S_TOTAL=Heurística TOTAL de Simons (1992), S_SETUP=Heurística SETUP de Simons (1992), RMB_HYB=Algoritmo de Ríos-Mercado y Bard (1999a), RMB_GRASP=GRASP de Ríos-Mercado y Bard (1998b), SI_DGK=Algoritmo de Das, Gupta y Khumawala (1995)

Tabla 5.12 – IPSOM y tiempo de CPU en segundos (entre paréntesis) de los métodos heurísticos y metaheurísticos para el taller SDST, grupo de problemas SSD125.

Como se puede observar, existe una clara tendencia hacia incrementos en los *IPSUM* totales para todos los algoritmos conforme disminuye el ratio entre los tiempos de proceso y los tiempos de cambio. Este resultado coincide con los obtenidos por otros autores (ver por ejemplo Gupta y Darrow, 1986 o Simons, 1992). Este extremo se comentará con detalle más adelante.

Los tiempos de ejecución de los métodos no cambian, como era de esperar, para los distintos grupos de problemas. Tampoco hay grandes variaciones en cuanto al *IPSUM* obtenido por los distintos métodos, aunque algunos acusan más que otros el incremento en los tiempos de cambio, por ejemplo el algoritmo SAOP. De manera general podemos decir que los métodos heurísticos específicos para el taller SDST obtienen unos resultados que son consistentemente peores que los de la regla RANDOM y que los métodos adaptados tienen un comportamiento mucho mejor. En todas las situaciones, los métodos propuestos GA_{sd} y HGA_{sd} obtienen mejores resultados que el resto de métodos evaluados. A modo de resumen, mostramos los resultados obtenidos para los cuatro grupos de problemas en la Tabla 5.13.

Grupo de problemas	Método	Mejor método		Segundo mejor método		Tercer mejor método		
		<i>IPSUM</i> Total (%)	Tpo. CPU (seg.)	<i>IPSUM</i> Método	Total (%)	Tpo. CPU (seg.)	<i>IPSUM</i> Método	Total (%)
SSD10	HGA_{sd}	0,33	28,28	GA_{sd}	1,04	15,51	ILS	1,61
SSD50	HGA_{sd}	0,85	29,44	GA_{sd}	2,13	18,33	RMB_GRASP	3,27
SSD100	HGA_{sd}	1,21	30,03	GA_{sd}	2,96	18,60	RMB_GRASP	4,43
SSD125	HGA_{sd}	1,35	29,86	GA_{sd}	3,29	18,75	RMB_GRASP	4,97

HGA_{sd} =GA híbrido propuesto, GA_{sd} =GA propuesto, ILS =Búsqueda local iterativa de Stützle (1998) (adaptación), RMB_GRASP =GRASP de Ríos-Mercado y Bard (1998b)

Tabla 5.13 – Resultados de los tres mejores métodos para los cuatro grupos de problemas.

Se puede observar que ambos algoritmos genéticos propuestos tienen un mejor comportamiento que el tercer mejor método, que pasa a ser la metaheurística RMB_GRASP por encima de ILS a partir del grupo de problemas SSD50. El método HGA_{sd} es, como se ha dicho, un 488 % mejor que el algoritmo ILS para el grupo de problemas SSD10, un 385 % mejor que el método RMB_GRASP para el

grupo de problemas SSD50, un 366 % mejor para el grupo SSD100 y finalmente, para el grupo SSD125, un 368 % mejor. El algoritmo GA_{sd} resulta ser un 55 % mejor que el método ILS para el grupo SSD10, un 54 % mejor que el algoritmo RMB_GRASP para el grupo SSD50, un 50 % mejor para el grupo SSD100 y un 51 % mejor para el grupo SSD125.

En el capítulo anterior se estableció un criterio de parada alternativo al número de evaluaciones del C_{max} , que es el tiempo transcurrido. Vimos como este criterio resulta ser más adecuado porque sitúa a todos los métodos en igualdad de condiciones y simplifica el análisis de los resultados al no tener que considerar los tiempos de CPU. De los métodos evaluados anteriormente, sólo los métodos metaheurísticos (ILS, SAOP, GAReev, RMB_GRASP, GA_{sd} y HGA_{sd}) y la regla RANDOM pueden interrumpirse una vez ha transcurrido el tiempo máximo permitido que si recordamos se limita a $(n \cdot 60.000)/1.000$ milisegundos. De la Tabla 5.14 a la Tabla 5.17 se muestran los resultados para los cuatro grupos de problemas considerados, considerando este criterio de parada.

Problema	RANDOM	ILS	SAOP	Spirit	GAReev	RMB_GRASP	GA _{sd}	HGA _{sd}
20x5	6,19	0,61	1,89	2,03	1,05	1,30	0,31	0,15
20x10	8,51	1,10	2,35	2,73	1,74	1,81	0,44	0,20
20x20	7,53	1,23	2,32	2,68	1,75	1,57	0,45	0,30
50x5	7,94	1,23	1,84	1,68	1,14	1,96	0,50	0,26
50x10	12,25	2,05	3,07	2,92	2,04	2,65	0,94	0,50
50x20	13,46	2,60	3,11	2,92	2,39	2,77	1,03	0,58
100x5	7,90	1,43	1,40	1,51	1,15	1,89	0,68	0,28
100x10	11,84	1,81	2,20	2,01	1,59	2,18	0,96	0,44
100x20	14,29	2,59	3,02	2,51	1,98	2,66	1,44	1,11
200x10	10,87	1,68	1,95	1,75	1,39	1,82	1,19	0,64
200x20	13,69	1,97	4,51	2,16	1,90	2,04	1,62	1,13
500x20	11,69	0,97	4,22	1,06	0,97	1,15	0,89	0,81
Media	10,51	1,60	2,66	2,16	1,59	1,98	0,87	0,53

RANDOM=Regla aleatoria, ILS=Búsqueda local iterativa de Stützle (1998) (adaptación), SAOP=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev=GA de Reeves (1995) (adaptación), RMB_GRASP=GRASP de Ríos-Mercado y Bard (1998b), GA_{sd}=GA propuesto, HGA_{sd}=GA híbrido propuesto

Tabla 5.14 – IPSOM de los métodos metaheurísticos para el taller SDST, criterio de parada establecido en $(n \cdot 60000)/1000$ milisegundos, grupo de problemas SSD10.

Problema	RANDOM	ILS	SAOP	Spirit	GAReev	RMB_GRASP	GA_{sd}	HGA_{sd}
20x5	11,53	2,12	4,44	4,98	3,11	2,82	1,00	0,67
20x10	10,07	2,03	3,56	4,67	2,86	2,64	1,06	0,56
20x20	7,89	1,71	2,64	3,62	2,25	1,97	0,45	0,28
50x5	18,18	4,84	5,25	5,73	3,77	5,07	1,86	0,81
50x10	16,63	4,52	5,14	5,33	3,74	4,71	2,05	1,26
50x20	14,38	4,33	4,10	4,58	3,44	3,72	1,77	1,28
100x5	20,13	5,27	4,39	4,98	3,55	5,16	2,45	1,18
100x10	18,06	4,75	4,37	4,07	3,47	4,34	2,53	1,63
100x20	15,77	4,07	4,12	3,78	3,16	3,66	2,45	1,90
200x10	17,55	3,59	4,34	3,31	3,00	3,64	2,59	1,83
200x20	15,78	3,23	6,28	3,18	3,03	3,05	2,32	1,93
500x20	14,16	1,52	6,51	1,51	1,50	1,62	1,11	1,13
Media	15,01	3,50	4,60	4,15	3,07	3,53	1,80	1,20

RANDOM=Regla aleatoria, ILS=Búsqueda local iterativa de Stützle (1998) (adaptación), SAOP=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev=GA de Reeves (1995) (adaptación), RMB_GRASP=GRASP de Ríos-Mercado y Bard (1998b), GA_{sd}=GA propuesto, HGA_{sd}=GA híbrido propuesto

Tabla 5.15 – IPSOM de los métodos metaheurísticos para el taller SDST, criterio de parada establecido en $(n \cdot 60000)/1000$ milisegundos, grupo de problemas SSD50.

Problema	RANDOM	ILS	SAOP	Spirit	GAReev	RMB_GRASP	GA _{sd}	HGA _{sd}
20x5	15,59	4,49	7,29	8,52	5,97	4,79	1,73	0,72
20x10	12,15	3,56	5,74	5,75	4,31	3,53	1,44	0,67
20x20	9,15	2,58	3,61	4,28	2,85	2,68	0,76	0,41
50x5	26,84	7,73	7,91	8,33	6,32	7,58	2,98	1,79
50x10	20,77	6,29	6,71	6,29	5,58	6,17	2,72	1,81
50x20	16,11	4,91	5,19	4,88	4,05	4,41	2,22	1,58
100x5	30,81	8,21	7,60	7,84	6,39	7,75	3,98	2,11
100x10	22,88	6,02	6,02	5,82	4,56	5,66	3,39	2,18
100x20	18,39	5,43	5,51	5,06	4,46	4,76	3,32	2,68
200x10	23,83	5,13	6,73	4,93	4,54	5,03	3,57	2,64
200x20	18,56	3,44	8,04	3,41	3,33	3,75	2,71	2,46
500x20	17,44	1,71	8,75	1,70	1,69	2,02	1,26	1,25
Media	19,38	4,96	6,59	5,57	4,50	4,84	2,51	1,69

RANDOM=Regla aleatoria, ILS=Búsqueda local iterativa de Stützle (1998) (adaptación), SAOP=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev=GA de Reeves (1995) (adaptación), RMB_GRASP=GRASP de Ríos-Mercado y Bard (1998b), GA_{sd}=GA propuesto, HGA_{sd}=GA híbrido propuesto

Tabla 5.16 – IPSOM de los métodos metaheurísticos para el taller SDST, criterio de parada establecido en $(n \cdot 60000)/1000$ milisegundos, grupo de problemas SSD100.

Problema	RANDOM	ILS	SAOP	Spirit	GAReev	RMB_GRASP	GA_{sd}	HGA_{sd}
20x5	17,17	4,47	8,06	9,30	5,73	5,31	1,61	0,76
20x10	12,90	3,36	6,45	5,54	4,81	4,02	1,22	0,57
20x20	9,28	2,52	3,62	3,79	2,89	2,78	0,87	0,52
50x5	30,55	9,19	10,12	9,93	7,85	8,85	3,81	1,99
50x10	22,52	7,29	7,51	7,64	6,29	6,60	2,94	1,80
50x20	17,33	5,93	6,11	5,52	4,66	5,22	2,82	2,13
100x5	34,62	9,25	8,48	8,67	6,65	8,35	3,96	2,23
100x10	25,08	6,62	6,77	5,36	5,12	6,04	3,90	2,24
100x20	19,41	5,36	5,68	4,81	4,28	4,88	3,34	2,57
200x10	26,46	5,49	7,34	5,19	4,93	5,58	3,91	3,02
200x20	19,59	4,07	8,49	4,00	3,86	3,89	2,79	2,66
500x20	18,81	1,82	9,52	1,81	1,80	2,21	1,29	1,32
Media	21,14	5,45	7,35	5,96	4,91	5,31	2,71	1,82

RANDOM=Regla aleatoria, ILS=Búsqueda local iterativa de Stützle (1998) (adaptación), SAOP=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev=GA de Reeves (1995) (adaptación), RMB_GRASP=GRASP de Ríos-Mercado y Bard (1998b), GA_{sd}=GA propuesto, HGA_{sd}=GA híbrido propuesto

Tabla 5.17 – IPSOM de los métodos metaheurísticos para el taller SDST, criterio de parada establecido en $(n \cdot 60000)/1000$ milisegundos, grupo de problemas SSD125.

El criterio de parada del tiempo transcurrido nos permite comparar si la hibridación del algoritmo GA_{sd} (HGA_{sd}) resulta en un mejor algoritmo. Podemos ver que el método híbrido es un 64 % mejor en el grupo SSD10, un 50 % en el grupo SSD50, un 49 % en el SSD100 y un 49 % en el SSD125. Luego los resultados indican que la hibridación genera un algoritmo que es mucho más eficaz que la versión estándar.

En cuanto al resto de métodos era de esperar que el algoritmo RMB_GRASP no presente un buen comportamiento al cambiar de criterio de terminación, ya que como vimos en el anterior estudio, se trata de un algoritmo excesivamente lento, y esto hace que con el criterio de parada establecido al tiempo transcurrido, no tenga suficientes iteraciones para converger. El GA GAReev pasa a ser el tercer mejor método en todos los grupos de problemas. Esto es debido a que es un algoritmo muy rápido y con el mismo tiempo permitido puede evaluar más soluciones que otros métodos. A modo de resumen, en la Tabla 5.18 se muestran los resultados de los tres mejores métodos para cada uno de los cuatro grupos de problemas.

Grupo de problemas	Método	Mejor método		Segundo mejor método		Tercer mejor método	
		<i>IPSOM</i> Total (%)	Método	<i>IPSOM</i> Total (%)	Método	<i>IPSOM</i> Total (%)	
SSD10	HGA _{sd}	0,53	GA _{sd}	0,87	GAReev	1,59	
SSD50	HGA _{sd}	1,20	GA _{sd}	1,80	GAReev	3,07	
SSD100	HGA _{sd}	1,69	GA _{sd}	2,51	GAReev	4,50	
SSD125	HGA _{sd}	1,82	GA _{sd}	2,71	GAReev	4,91	

HGA_{sd}=GA híbrido propuesto, GA_{sd}=GA propuesto, GAReev=GA de Reeves (1995) (adaptación)

Tabla 5.18 – Resultados de los tres mejores métodos para los cuatro grupos de problemas, criterio de parada establecido en $(n \cdot 60000)/1000$ milisegundos.

Como se puede observar, de nuevo con este criterio de parada los algoritmos desarrollados en esta Tesis Doctoral presentan un mejor comportamiento que el resto. El tercer mejor algoritmo es el GAReev. El mejor método de la comparativa, el HGA_{sd} resulta ser entre un 256 % y un 300 % mejor según el grupo de problemas considerado. De igual manera, el algoritmo GA_{sd} es entre un 171 % y un 183 % mejor que el método GAReev dependiendo del caso.

Resulta de interés comprobar como se comportan los métodos conforme el ratio entre los tiempos de proceso y los tiempos de cambio disminuye. La Figura 5.14 muestra esta comparativa.

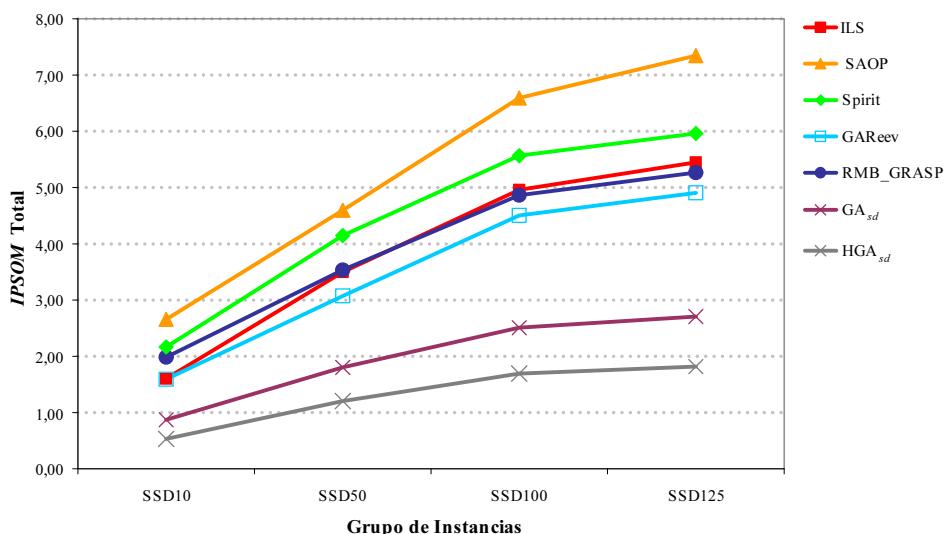


Figura 5.14 – Evolución del *IPSM* total frente a la magnitud de los tiempos de cambio para los mejores métodos metaheurísticos.

Se puede observar cómo conforme aumenta la magnitud de los tiempos de cambio de partida dependientes de la secuencia, parece que los problemas se hacen más “difíciles”. Creemos que el motivo detrás de este comportamiento es que todos los métodos metaheurísticos propuestos se basan en encontrar una permutación de los trabajos, y aunque consideran explícitamente los tiempos de cambio, éstos no dejan de ser una consecuencia de la permutación hallada. Sería interesante diseñar métodos pensando desde el principio en los tiempos de cambio y no en la secuencia. Hemos de tener en cuenta que en el grupo de problemas SSD125, los tiempos de cambio son mayores incluso que los tiempos de proceso.

Como conclusión podemos decir, que los dos algoritmos desarrollados en esta Tesis Doctoral para el taller SDST son muy eficaces y son considerablemente mejores que los métodos publicados hasta la fecha y también que las adaptaciones de los mejores métodos propuestos para el taller de flujo estándar.

5.8. Conclusiones del capítulo

En este capítulo se ha considerado una extensión del problema del taller de flujo más general y realista como es considerar la existencia de tiempos de cambio de partida, con la particularidad de que estos tiempos son separables e independientes de la secuencia. Se ha visto como este problema ($F/S_{sd}, prmu/C_{max}$) es más complejo que el taller de flujo estándar y se ha realizado una completa revisión del estado del arte en esta cuestión.

De esta manera, se han presentado dos algoritmos genéticos (GA_{sd} y HGA_{sd}) que son adaptaciones de los algoritmos propuestos en el Capítulo 4. Al igual que se hiciera en el capítulo anterior, se ha realizado un amplio experimento para calibrar los algoritmos propuestos. En este caso se ha utilizado un conjunto de 480 problemas que se basan en los problemas de Taillard (1993) pero con la adición de los tiempos de cambio de partida dependientes de la secuencia. Concretamente, en estos problemas tenemos cuatro grupos de 120 problemas donde en cada uno los ratios entre los tiempos de proceso y los tiempos de cambio varían.

Estos algoritmos se han comparado con otros 11 métodos entre los que tenemos algoritmos que hemos adaptado al nuevo problema y métodos existentes específicamente diseñados para el mismo. La evaluación comparativa, realizada tanto en términos de eficiencia como de eficacia y con los criterios de terminación del máximo número de C_{max} evaluados y el tiempo transcurrido indica que ambos algoritmos genéticos propuestos son más eficaces que el resto de métodos. Concretamente, el algoritmo genético GA_{sd} es entre un 50 % y un 183 % más eficiente que el resto de métodos y el HGA_{sd} entre un 256 % y un 488 % mejor.



CAPÍTULO

TALLER DE FLUJO HÍBRIDO CON TIEMPOS DE CAMBIO DE PARTIDA: UNA APLICACIÓN AL SECTOR CERÁMICO

En esta Tesis Doctoral se han diseñado y evaluado nuevos algoritmos genéticos para el taller de flujo de permutación, que es, como se ha visto, un problema de programación de la producción más teórico que práctico. Aunque en el Capítulo 5 adaptamos los algoritmos presentados para la consideración de los tiempos de cambio de partida dependientes de la secuencia, esto es, el problema $F/S_{sd}, prmu/C_{max}$, todavía existe un “gap” entre estos problemas y los problemas reales comunes en entornos industriales.

La realidad de los sistemas productivos es mucho más compleja y aunque se pueden hacer simplificaciones que no afecten significativamente a los resultados, existen restricciones y situaciones adicionales que es necesario tener en cuenta en los métodos de programación de la producción, si queremos que realmente lleguen a ser de utilidad práctica. En este sentido, existen una gran cantidad de trabajos donde precisamente se pone de manifiesto la falta de aplicabilidad de los métodos

de programación de la producción que se han venido proponiendo en la literatura. Ya en 1981, Graves, en una completa revisión de los métodos de programación y planificación de la producción, comentó que existe una brecha importante entre la teoría y la práctica de “*scheduling*” y propuso realizar investigaciones en aspectos concretos para aumentar la aplicabilidad de los métodos. La existencia de esta separación se constata en varios estudios que se han realizado en empresas. Por ejemplo en el trabajo de Ledbetter y Cox (1977) (y posterior ampliación por parte de Ford et al., 1987), donde se presentan resultados de la utilización por parte de las empresas de técnicas, métodos y algoritmos para la planificación de la producción. Estos resultados arrojan una baja o nula implantación de sistemas de ayuda a la toma de decisiones o de métodos que permitan resolver los problemas. También en el estudio de McKay, Safayeni y Buzacott (1988) se detallan numerosas situaciones comunes en la práctica de los sistemas de producción y que la literatura existente no contempla. Estudios posteriores (ver Olhager y Rapp, 1995) también coinciden en la valoración de los resultados: los métodos propuestos son difíciles de utilizar y además no contemplan toda la problemática existente en las empresas a la hora de hacer la programación de la producción. Adicionalmente, se han publicado artículos con el propósito de concienciar a la comunidad científica de que es necesario investigar en problemas reales de programación de la producción, claros ejemplos son el trabajo de MacCarthy y Liu (1993) o específicamente para los talleres de flujo el ya citado trabajo de Dudek, Panwalkar y Smith (1992). Muy recientemente, McKay, Pinedo y Webster (2002) han vuelto a poner de manifiesto las deficiencias existentes en la literatura en cuanto a la aplicabilidad de los métodos propuestos.

También es fácil encontrar este tipo de críticas en los estudios de revisión o estado del arte. Junto con el ya citado trabajo de Graves, las revisiones de distintos tipos de problemas de programación de la producción realizadas por Allahverdi, Gupta y Aldowaisan (1999), Yang y Liao (1999), Linn y Zhang (1999), Vignier, Billaut y Proust (1999) y Cheng, Gupta y Wang (2000) también contienen conclusiones donde se hace referencia a la falta de aplicabilidad de los trabajos revisados.

La presente Tesis Doctoral se enmarca en un Proyecto Coordinado en el que participan la Universidad de la Laguna, la Universidad de Alcalá de Henares y la Universidad Politécnica de Valencia, titulado “Programación Flexible de la

Producción en Empresas del Sector Cerámico de la Comunidad Valenciana” y financiado por el Ministerio De Ciencia y Tecnología. El objetivo principal de este proyecto es precisamente el proponer métodos y algoritmos que sean de utilidad en entornos productivos reales, concretamente en los talleres de flujo que se encuentran en las empresas cerámicas.

Para caracterizar la situación real del sector cerámico o de producción de azulejos se ha realizado un extenso estudio de las empresas del sector así como de su proceso productivo. Concretamente, se ha realizado una encuesta de 47 preguntas orientadas hacia el sistema productivo y también hacia la organización y estrategia del sistema de operaciones. Los resultados de esta encuesta se han obtenido mediante entrevistas personales con gerentes o directores de producción de las empresas de la Asociación Española de Fabricantes de Azulejos y Pavimentos Cerámicos (ASCER) (<http://spaintiles.info/esp/index.asp>). Se han realizado encuestas a 81 empresas del sector, lo que representa el 38 % de las empresas a nivel español y el 40 % de las empresas de ASCER.

Este extenso estudio, cuyos resultados se recogen en Vallada et al. (2003a) y en Vallada et al. (2003b), permite conocer en profundidad la problemática concreta del sector y con ello proponer métodos adecuados para resolver los problemas de programación de la producción, objeto del presente capítulo.

Antes de presentar los métodos propuestos, vamos a sintetizar los principales datos del sector, el producto y el proceso de producción del azulejo o baldosa cerámica. Esto nos permitirá caracterizar el problema general de programación de la producción en el sector lo que a su vez nos servirá para proponer nuevos métodos avanzados para resolver los problemas detectados.

6.1. El sector de producción de baldosas cerámicas

El sector de producción de azulejos en España es uno de los más importantes en el mundo. España alcanzó en el año 2001 el 43,1 % de la cuota de producción de la UE y el 11,1 % de la cuota mundial de producción, y siendo con ello el primer país productor en el mundo, junto con Italia, que tiene una cuota también

del 11,1 % (ver ASCER, 2002). Se espera que en los datos referidos al año 2002 España sea el primer productor mundial. En el año 2001, la producción total del sector fue de 638 millones de m² de azulejos, mientras que en año 2000 fue de 621 millones de m² (ver ASCER, 2001), lo que representa un aumento del 2,7 %. Hay dos factores claramente significativos en el sector. El primero de ellos es la alta concentración de las empresas en la provincia de Castellón, en la zona formada por las localidades de Alcora, Borriol, Onda, Nules y Castellón de la Plana. En el año 2001 el 93,6 % de la producción nacional se concentró en la zona citada, donde se ubican el 80 % de las empresas. En el año 2001, el sector contaba con 265 empresas (255 en el 2000), entre las que podemos encontrar desde grandes fabricantes a pequeños talleres.

El segundo factor es el carácter claramente exportador del sector. En el año 2001 el sector exportó a 186 países, mientras que en el 2000 se hicieron exportaciones a 177 países distintos. El valor de estas exportaciones en el 2001 ascendió a 1.987,8 millones de €, lo que representa una cuota del 28,7 % del comercio mundial de azulejos. En la Figura 6.1 podemos ver el fuerte incremento de la capacidad productiva del sector en España y la evolución de la exportación desde el año 1970 hasta el 2001. Los datos se han obtenido a partir de Dalmau Porta y De Miguel Fernández (1991), Gómez López (1999) y de los informes técnicos de ASCER, ASCER (1998), ASCER (1999), ASCER (2000), ASCER (2001) y ASCER (2002).

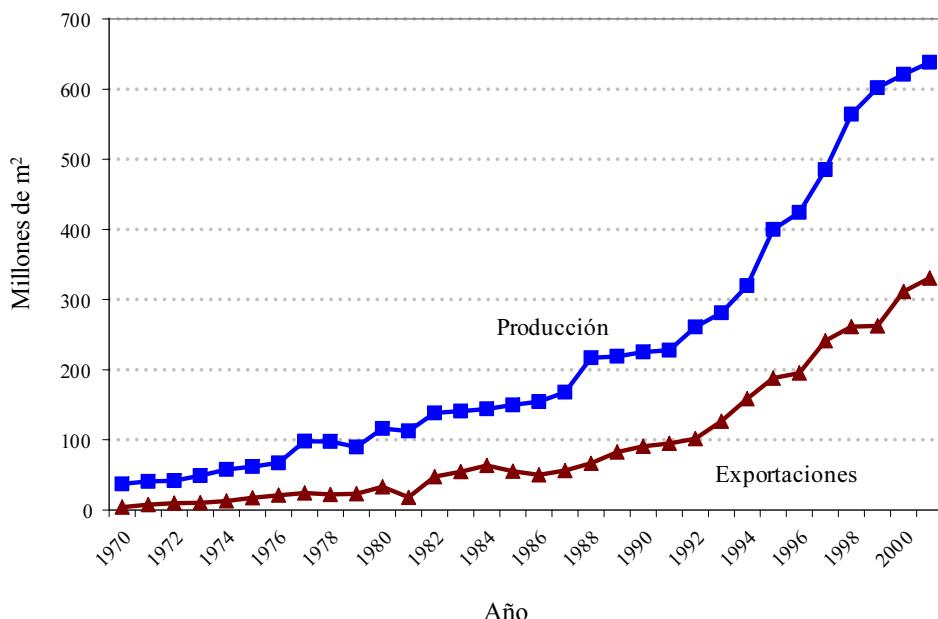


Figura 6.1 – Evolución de la producción y exportación del sector cerámico en España desde el año 1970 hasta el 2001.

A partir de los datos anteriores podemos ver como se trata de un sector en fuerte expansión, aunque se ha visto afectado en los últimos años por el clima de estancamiento económico general. Aún así podemos observar como la capacidad productiva ha aumentado más de un 100 % en los últimos 10 años.

6.2. El azulejo o baldosa cerámica

Básicamente, un azulejo o baldosa cerámica es una pieza plana, de poco espesor, que se usa, por lo general, como pavimento para suelos o revestimiento de paredes. Son piezas cerámicas e impermeables que están constituidas normalmente por un soporte cerámico, de naturaleza arcillosa, y de un recubrimiento vítreo opcional, llamado esmalte cerámico. Aparte de las arcillas, se utilizan fundentes, colorantes y otras materias primas inorgánicas que se someten a procesos de molturación y/o amasado, para luego moldearse y cocerse a temperaturas altas.

Podemos encontrar una amplia gama de productos cerámicos que viene dada por las variedades de las materias primas utilizadas, por los diversos procesos de producción y por la finalidad de los productos, que en la actualidad se utilizan mayoritariamente, en pavimentos y revestimientos. Los pavimentos se usan normalmente para la pavimentación interior de viviendas, terrazas o techadas, también en locales públicos, como escuelas y hospitales o lo que se conocen como pavimentos técnicos, que se utilizan en piscinas e industrias. El revestimiento se utiliza también en interiores (cocinas y baños), pero también para terrazas, decoración, etc...

Las baldosas cerámicas poseen una serie de características que hacen de las mismas un producto necesario y de gran utilidad (ver Escardino Benlloch y González Cudilleiro, 1991), por ejemplo se dice que las baldosas son *versátiles*, dado que se pueden instalar de vertical u horizontalmente, sirven para interiores o exteriores, para pavimento, revestimiento, decoración u otras múltiples aplicaciones. También existe una amplia *variedad*, tanto de colores, como de formas y tamaños. Los azulejos son de fácil *instalación y mantenimiento*, dado que son muy sencillos de colocar y las piezas rotas o defectuosas de pueden cambiar independientemente del resto y también son fáciles de limpiar. Una de las características más notorias de los azulejos es su *intergridad*, dado que resisten al agua, a la humedad, agentes químicos y biológicos. También poseen una alta resistencia mecánica (impacto, flexión, compresión), resisten muy bien los cambios bruscos de temperatura, la luz, etc... No menos importantes son las características de *eficiencia y seguridad*, dado que son excelentes aislantes térmicos, acústicos y eléctricos, además de ser completamente ignífugos. Otras características destacables son la alta durabilidad (el azulejo puede permanecer siglos inalterado) y la resistencia a la abrasión, manchas, cuarteo, ácidos y otros agentes químicos, temperatura, etc, cualidades, que, con el progreso de las tecnologías de fabricación han ido ampliando el espectro de aplicación de los pavimentos y revestimientos cerámicos.

6.3. Tipos de baldosas cerámicas

Una primera clasificación puede realizarse en función de si la baldosa cerámica está o no esmaltada. Esto determina el proceso de fabricación posterior, dado

que en una baldosa no esmaltada tendremos un proceso de cocción única, mientras que las baldosas esmaltadas pueden someterse a cocción única (monococción) o a un proceso de biccocción.

No existe una clasificación de baldosas cerámicas universalmente aceptada, más bien existe una clasificación de uso común, sea por motivos técnicos o por razones comerciales. Existen varios criterios bajo los cuales clasificar los tipos de baldosas cerámicas, algunos de los cuales pueden verse en la Tabla 6.1.

Tipo de baldosa	Soporte	Moldeo	Medidas (cm)	Espesor (mm)	Esmalte
Azulejo	Poroso	Prensado	10 x 10 a 45 x 60	< 10	Si
Pavimento de gres	No poroso	Prensado	10 x 10 a 60 x 60	> 8	Si
Gres porcelánico	No poroso	Prensado	15 x 15 a 60 x 60	> 8	No
Baldosín catalán	Poroso	Extrudido	13 x 13 a 24 x 40	< 8	No
Gres rústico	No poroso	Extrudido	11,5 x 11,5 a 37 x 37	> 10	Si-No
Barro cocido	Poroso	Extrudido	varios	> 10	No

Tabla 6.1 – Tipos de baldosas producidos en España.

A parte de la existencia o no de la etapa de esmaltado, aparecen otros criterios de clasificación igualmente válidos, como es el número de cocciones que soporta la pieza durante el proceso, o una diferenciación según la forma de preparación de la materia prima (vía húmeda o vía seca), el conformado de la pieza (prensado, extrusionado, colado, etc...), el tipo de arcilla base utilizada (pasta roja, pasta blanca, etc...), el uso que se vaya a dar al producto (pavimento, revestimiento de paredes, interior o exterior), la absorción de agua del producto final (gres, semigres, poroso, gres porcelánico, etc.). La gran variedad en el proceso de producción de baldosas cerámicas y la enorme gama de modelos resultante hace que sea difícil la clasificación y la esquematización de la tipología de las baldosas cerámicas. En el Anexo D podemos encontrar una breve descripción de los principales tipos de baldosas cerámicas que se producen en España.

6.4. El proceso de producción de las baldosas cerámicas

Una vez conocemos los productos que fabrica el sector, vamos a estudiar cómo los fabrica. El proceso de producción de la baldosa cerámica es variado y complejo. Existen pocos trabajos donde se detallen de una manera sencilla las etapas productivas y las fases por las que pasa el producto. ATECE (1990) publicó un extenso trabajo muy técnico sobre la tecnología de fabricación. Más información se puede también encontrar en ASCER. En esta sección aunamos éstos y otros trabajos para ofrecer una visión simplificada, pero rigurosa, del proceso de producción del azulejo.

La fabricación de pavimentos y revestimientos cerámicos ha experimentado una evolución considerable y continua en los últimos años. Hoy en día, el proceso de producción está totalmente automatizado, hecho que garantiza la elevada calidad de los productos cerámicos. Como ya vimos en la Sección 6.3, la tipología de las baldosas cerámicas es muy variada y la cantidad de productos resultantes es enorme, lo que hace que los procesos productivos sean igualmente variados y complejos. El criterio más utilizado a la hora de clasificar los procesos de fabricación está basado en el número de cocciones que soporta la pieza durante el proceso, una (monococción), dos (bicocción) o incluso más de dos (tercer fuego). Existen también diferenciaciones a la hora de la preparación de la materia prima, que puede prepararse por vía húmeda o por vía seca, así como en el conformado de la pieza (prensado, extrusionado, colado), la existencia o no de la etapa de esmaltado, el tipo de arcilla utilizada (pasta roja, pasta blanca), etc. De manera general, existen una serie de etapas principales en la producción de las baldosas cerámicas, sea cual sea su proceso productivo:

1. Preparación de materias primas.
2. Conformación de la baldosa cerámica.
3. Tratamientos para conferir al producto conformado las propiedades finales deseadas, incluyendo cocciones y esmaltados.

Como ya hemos comentado, en cada una de estas tres etapas principales aparecen procesos productivos alternativos, de los que cabe destacar, para cada una de ellas:

- Preparación de materias primas
 - Preparación por vía húmeda.
 - Preparación por vía seca.
- Conformación de la pieza
 - Conformación por prensado.
 - Conformación por extrusión.
- Cocciones
 - Monococción rápida.
 - Biccocción rápida.
 - Biccocción mixta.

De esta manera, se dan múltiples combinaciones, como por ejemplo, preparación por vía húmeda, conformación por prensado y luego monococción. A partir del diagrama de la Figura 6.2 se pueden ver los distintos procesos productivos.

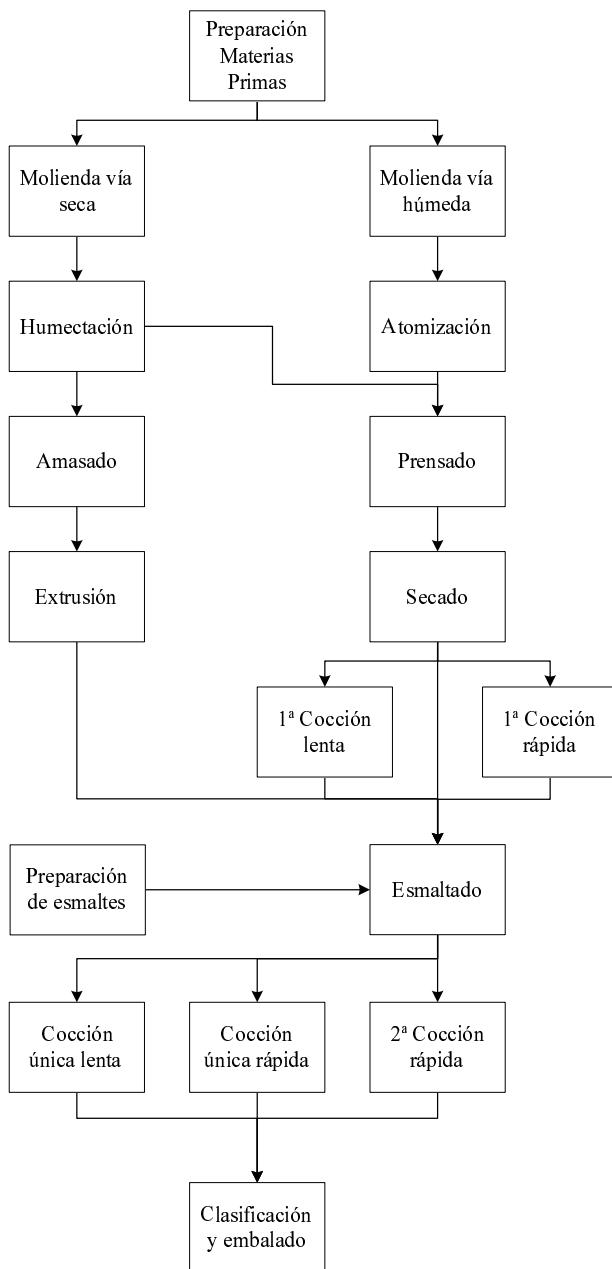


Figura 6.2 – Procesos de fabricación de baldosas cerámicas.

Observando el citado diagrama se pueden desgranar los principales procesos de producción que se dan en la actualidad, aunque, de todos ellos, el predominante es la monococción porosa. A continuación se detallan los distintos tipos de procesos productivos.

6.4.1. Monococción porosa

Hoy por hoy representa el proceso más utilizado, permitiendo unos ciclos de producción muy cortos. En este proceso, la baldosa cerámica una vez prensada y esmaltada, se somete a una única cocción como podemos observar en la Figura 6.3. Cabe destacar que la preparación de las materias primas puede hacerse tanto por vía seca como por vía húmeda, aunque es ésta última la predominante. También es posible que exista una segunda etapa de secado después del esmaltado. Asimismo algunos productos concretos, pasan por etapas adicionales antes de la clasificación, como son pulidos, rectificados, cortados, etc.

6.4.2. Biccocción rápida o doble cocción rápida

El proceso de bicocción es más lento que la monococción. En este caso la baldosa cerámica se cuece antes y después del esmaltado. Concretamente, en la doble cocción rápida, las dos cocciones son de ciclo corto (Figura 6.4).

6.4.3. Biccocción mixta o 2^a biccocción rápida

Se trata de una variante de la bicocción rápida, en la cual la primera cocción es de ciclo lento (Figura 6.5).

6.4.4. Fabricación de baldosas extrudidas

Se trata de un proceso utilizado solo cuando se quieren producir piezas especiales, que son baldosas cerámicas de geometría particular (cenefas, rodapiés, vierteaguas, peldaños, tacos, etc...). Las piezas se moldean por extrusión y luego se someten, normalmente, a un único ciclo de cocción (cocción única) tras el esmaltado (Figura 6.6).

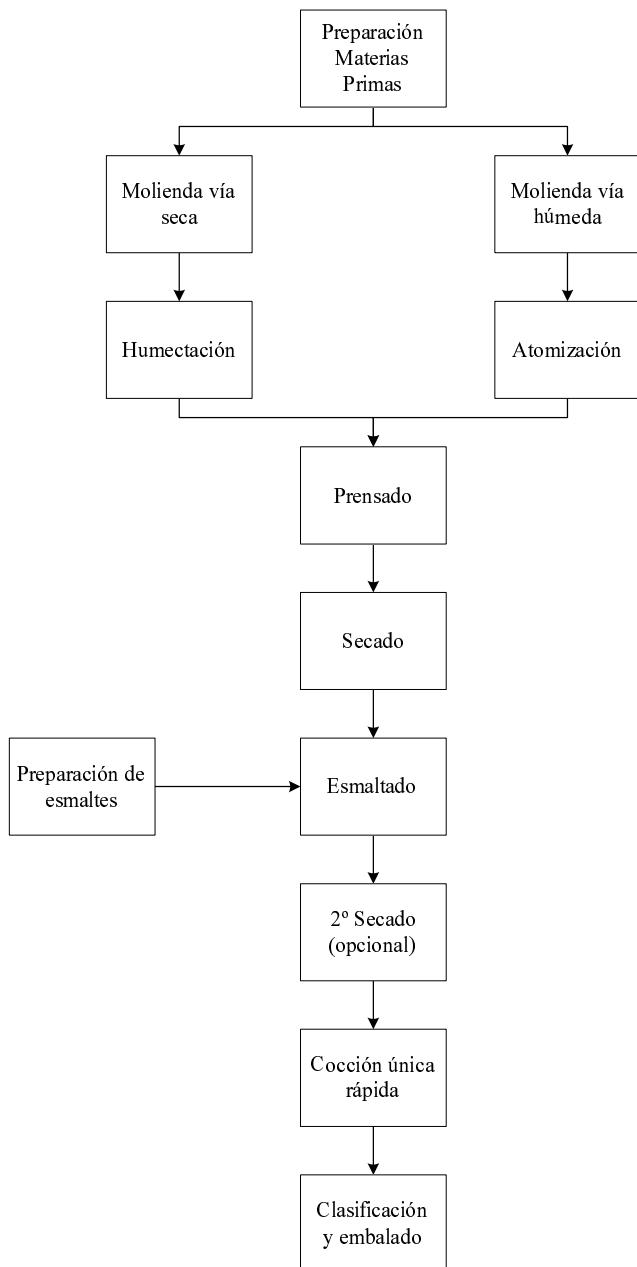


Figura 6.3 – Fabricación de baldosas por monococción.

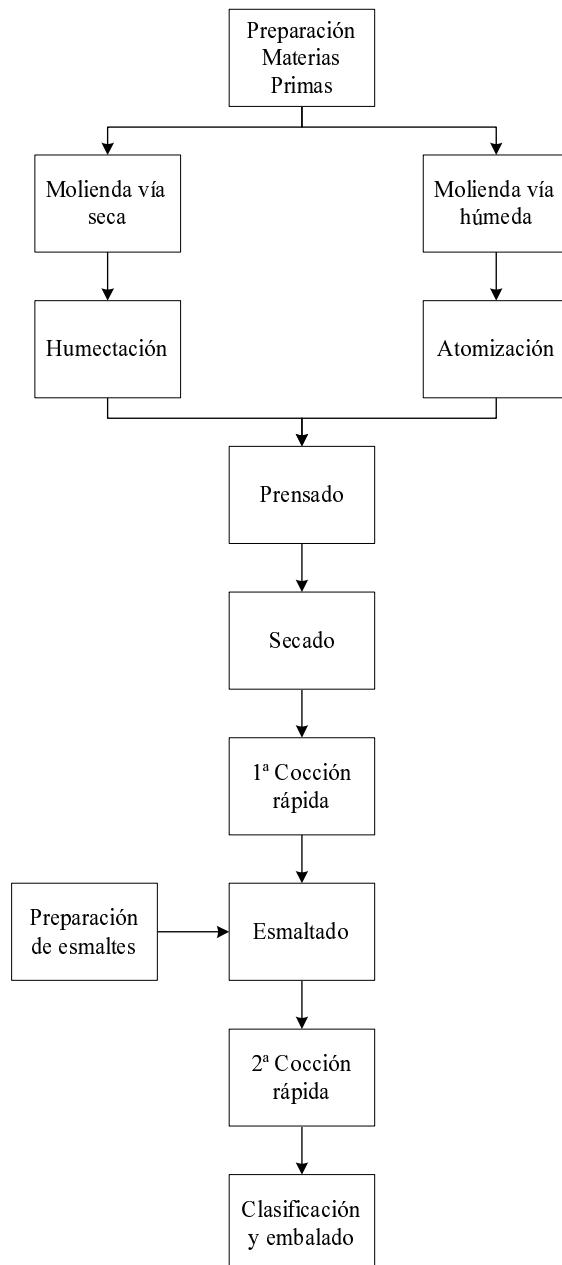


Figura 6.4 – Fabricación de baldosas por bicocción rápida.

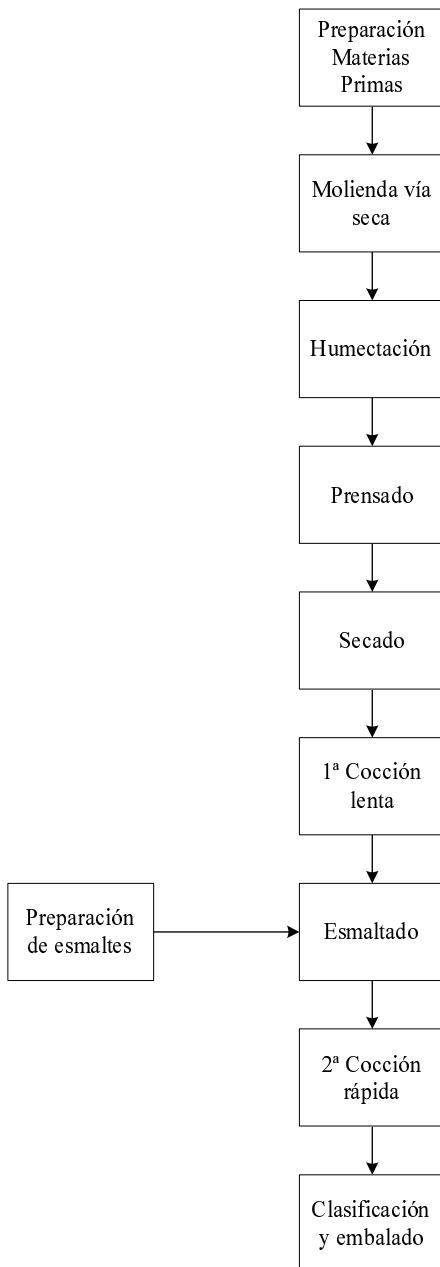


Figura 6.5 – Fabricación de baldosas por bicocción mixta.

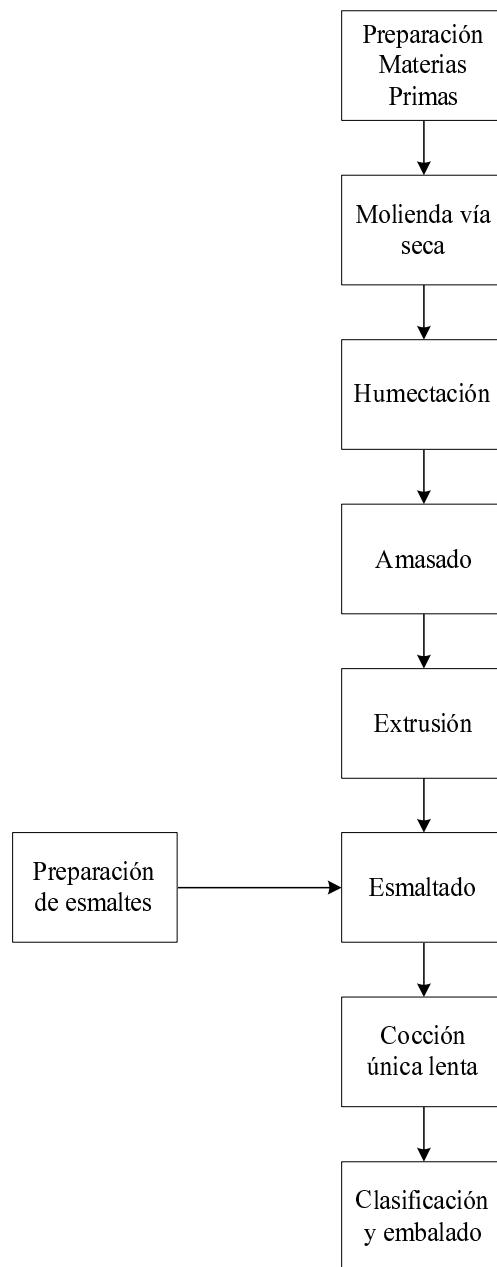


Figura 6.6 – Fabricación de baldosas por extrusión.

Para comprender el proceso de producción de las baldosas cerámicas, se expone a continuación una explicación de las distintas fases por las que pasa la producción del azulejo.

6.4.5. Preparación de las materias primas

El proceso de producción comienza con la recepción de las materias primas, extraídas de las minas o canteras. Existe una gran variedad de arcillas, y normalmente se utilizan de tipo illítico-caoliníticas, con contenidos de cuarzo, feldespatos, chamota y carbonatos. El tratamiento previo a la molienda es sencillo; las arcillas son almacenadas en eras para después ser sometidas a un proceso de homogenización y trituración, pasando posteriormente a ser almacenadas en graneros. Los distintos tipos de arcillas se miden y se pesan para transportarse mediante cintas o tornillos sin fin hacia las tolvas de carga, que alimentan a los molinos.

6.4.6. Molienda o molturación

Por molturación en materias primas sólidas se entiende a las operaciones llevadas a cabo para reducir las dimensiones del material. El objetivo de la molturación es aumentar la superficie específica de la materia prima, para con ello obtener una elevada homogeneidad en la masa y reacciones químicas más completas y breves. Existen dos métodos para molturar las materias primas, en función de la presencia o no de agua en el proceso. La elección de uno u otro método viene determinada por las características que se requieren en el producto en cuanto al grado de finura de las partículas. La molturación por vía seca produce una elevada fragmentación, donde se encuentran partículas de tamaño superior a las 300 micras, mientras que en la molturación por vía húmeda las partículas resultantes son siempre menores a las 200 micras.

En líneas generales, la molturación por vía húmeda reduce rápidamente las dimensiones de los materiales, al tiempo que resulta en una mejor homogenización de la materia prima, mientras que la molturación por vía seca sólo es útil cuando las materias primas son muy puras o cuando se desea producir materiales de calidad intermedia. Actualmente, el proceso predominante es la molturación por

vía húmeda.

6.4.6.1. Molturación por vía húmeda

La molturación por vía húmeda se hace por medio de molinos de bolas para funcionamiento en discontinuo (ver Figura 6.7) o molinos tubulares (funcionamiento en continuo). Las etapas a seguir en la molturación son:

1. Carga de la materia prima, agua y fluidificantes.
2. Molturación.
3. Descarga de la barbotina obtenida.

En esta etapa, existen una serie de factores que influyen de manera muy importante en el resultado y en el rendimiento del proceso, como son la velocidad del molino, los cuerpos molturantes (normalmente bolas de alúmina), la carga de bolas, etc. Una vez terminada la molturación (que dura unas 12 horas aproximadamente) el resultado del proceso, la barbotina (de aspecto parecido al chocolate líquido), se almacena en unas balsas que están en permanente agitación, para luego tamizarse, homogenizarse y ser de nuevo almacenada en balsas agitadas.



Figura 6.7 – Molino de bolas.

6.4.6.2. Secado por atomización

La barbotina tamizada tiene un contenido en humedad muy elevado, en torno a los 0,30-0,55 Kg de agua/Kg de sólido seco. Para que la etapa de prensado se realice en condiciones adecuadas es necesario reducir este contenido de agua. El proceso más extendido es el secado de la barbotina por atomizado. En este proceso, la barbotina se bombea y se pulveriza en finas gotas (nebulización) contra una corriente de aire caliente ascendente. El contacto entre la fina gota y el gas caliente provoca un secado violento de la primera, dando como resultado un gránulo esferoidal y hueco con un bajo contenido en agua (0,05-0,07 Kg de agua/Kg de sólido seco). Las etapas por las que pasa el secado por atomizado se pueden resumir de la siguiente manera:

1. Bombeo y nebulización de la barbotina.
2. Generación y alimentación a la torre de secado de los gases calientes.
3. Secado por contacto entre el gas caliente y la gota de barbotina.
4. Separación del granulado y de los gases calientes.

Este proceso, pese a tener un alto coste energético, se ha impuesto a todos los demás, dado que este tipo de secado aporta numerosas ventajas en el desarrollo de las etapas posteriores de producción. El polvo granulado resultante es homogéneo, esférico y hueco en el interior, por lo que resulta muy fluido, facilitando las operaciones de llenado de los moldes en las prensas y los prensados de piezas de grandes formatos. Además, el proceso de atomizado es relativamente simple, por lo que puede ser automatizado y puede hacerse en continuo, por lo que los costes se abaratan.



Figura 6.8 – Atomizador.

6.4.7. Humectación y amasado

Cuando la molienda de las materias primas se ha hecho en seco y se desea conformar la pieza por extrusión, es necesario que la materia prima se encuentre en estado plástico, por lo que se hace un mezclado íntimo con agua, de esta manera se consigue una pasta plástica fácilmente moldeable por extrusión, a este procedimiento se le conoce como humectación y amasado.

6.4.8. Extrusión

Este procedimiento consiste en hacer pasar una columna de pasta, en estado plástico, a través de una matriz que conforma una pieza de sección constante. Los equipos utilizados para el moldeado por extrusión son:

1. Sistema propulsor (normalmente de hélice).
2. Matriz.
3. Cortadora.

6.4.9. Prensado

Esta etapa es muy importante dentro del proceso de producción. En el prensado se realizan simultáneamente las siguientes operaciones:

1. Formación de la baldosa: se le da a la materia prima una forma geométrica definida y un tamaño concreto.
2. Compactación de la baldosa: se le da a la materia prima una firmeza y dureza necesaria para que resista las operaciones que sobre ella se efectuarán en crudo, antes de la cocción (transporte, soplados, esmaltado, etc...).
3. Compresión: se intentan eliminar los vacíos existentes en la mezcla, para evitar problemas en las posteriores etapas, sobre todo a la hora de la cocción.

Al contrario que el conformado (o moldeado) por extrusión, el prensado en seco, (entre un 5 % y un 7 % de humedad) es el método más utilizado a la hora de

conformar la pieza. El prensado en seco aporta numerosas ventajas al proceso de producción, como son la alta resistencia de la baldosa en crudo (llamada también bizcocho, o bizcocho verde), alta productividad del proceso, facilidad de secado, mínimas deformaciones en las operaciones, reducción de la contracción en la etapa de cocción, etc.

Para el prensado existen varios tipos de prensas; mecánicas, isostáticas e hidráulicas, de las cuales las últimas son las más utilizadas debido a que presentan unas características más deseables como es la elevada fuerza de compactación, alta productividad, facilidad de regulación y constancia en el tiempo del ciclo de prensado. Las fases por las que pasa el prensado se pueden resumir en:

1. Carga del granulado o polvo en el molde.
2. Cierre del molde y primera prensada.
3. Aireación, mediante apertura parcial del molde.
4. Prensadas adicionales.
5. Apertura del molde y extracción del azulejo.

En la figura Figura 6.9 se puede observar una prensa hidráulica típica.



Figura 6.9 – Prensa hidráulica.

Es importante destacar que inmediatamente después del prensado la baldosa se recoge por una máquina, el recogedor de la prensa, que efectúa una serie de tratamientos necesarios, como son el volcado de la pieza (giro de 180°)¹ y el cepillado y desbarbado para la eliminación de impurezas. En la Figura 6.10 se expone un ejemplo de este tipo de máquina.

¹El volcado de las piezas es necesario, ya que la parte de las “costillas”, es decir, la parte inferior de la baldosa, es la que queda arriba después del prensado, no pudiéndose hacer al revés.

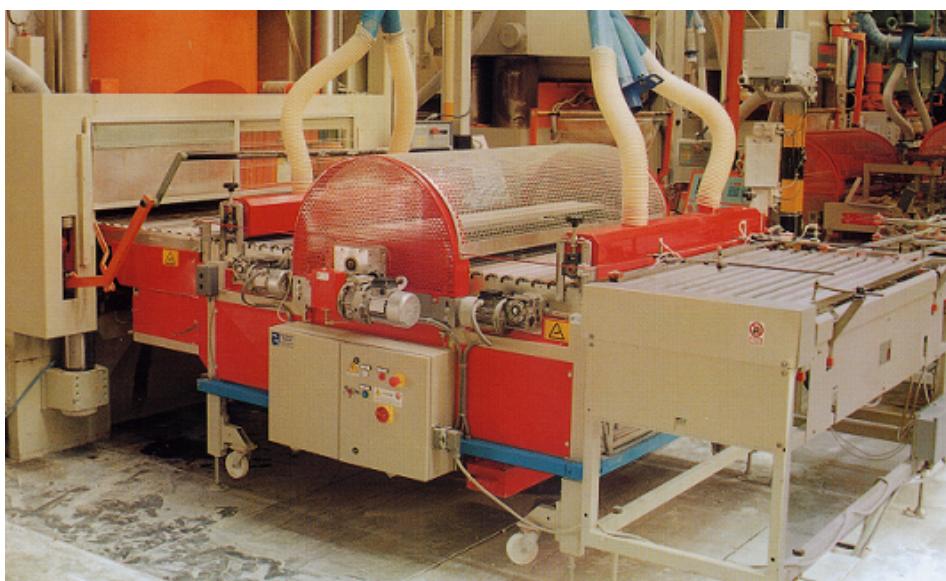


Figura 6.10 – Recogedor de la prensa.

6.4.10. Secado

Tras el conformado de la pieza es necesario reducir todavía más el contenido en agua para que la fase de esmaltado se desarrolle adecuadamente. Concretamente, los niveles de agua deben ser inferiores a 0,005 Kg de agua por Kg de sólido. En el secado, las piezas se exponen a corrientes de aire caliente y seco, para que el líquido de la pieza se elimine por evaporación. Hoy por hoy el secado de las piezas se hace en continuo, existiendo dos tipos de secaderos, los secaderos verticales y los secaderos horizontales.

En los secaderos verticales las piezas se colocan en cestones, que se mueven por el interior del secadero verticalmente. Las temperaturas de este tipo de secaderos suelen rondar los 200°C y los ciclos de secado suelen estar entre los 35 y los 50 minutos. En la Figura 6.11 tenemos un ejemplo de secadero vertical.



Figura 6.11 – Secadero vertical.

El secadero horizontal es muy parecido a un horno monoestrato de rodillos (véase Sección 6.4.13), en donde las piezas se colocan en planos horizontales dentro del secadero y se desplazan por encima de los rodillos. La temperatura de este tipo de hornos es superior, en torno a los 350°C, pero los ciclos de secado son menores, entre 15 y 25 minutos. Pese a funcionar a una temperatura mayor, los secaderos horizontales tienen un consumo de energía menor, debido a su mayor eficiencia. En la Figura 6.12 se observa un ejemplo de secadero horizontal.



Figura 6.12 – Secadero horizontal.

Aunque el secado se hace previo al esmaltado, en ocasiones, también se secan las piezas antes de la etapa de cocción. De nuevo esto se hace para reducir el contenido en humedad de las piezas hasta niveles suficientemente bajos para que la etapa de cocción se desarrolle de manera adecuada.

6.4.11. Esmaltado y serigrafía

En esta fase se aplican una o varias capas de vidriado para cubrir la superficie de la pieza. Esto se hace para dotar a la baldosa de una serie de propiedades, tanto técnicas como estéticas y que se han comentado en la Sección 6.2. En esta etapa existen multitud de variantes, dado que, como hemos visto en secciones anteriores, podemos tener desde una baldosa cerámica no esmaltada hasta baldosas con numerosos esmaltados y decoraciones.

El esmaltado de las baldosas cerámicas se realiza en continuo, donde las piezas van moviéndose por una cinta o línea transportadora, sobre la que se disponen los distintos aparatos necesarios para el esmaltado. Vamos a explicar una línea típica

de esmaltado en el proceso de monococción porosa por vía húmeda y prensado en seco. Nada más salir del secadero, las piezas pasan por un aerosol de agua que sirve para preparar la pieza para el esmaltado. Inicialmente se aplica una capa de esmalte base, llamado también fondo de preparación, que es un esmalte de fuerte color, que sirve para ocultar el color original de la pieza. Tras la aplicación del esmalte base, se aplica lo que se conoce como engobe, que no es más que otro tipo de esmalte, que sirve para tapar defectos superficiales y poros, proporcionar homogeneidad y preparar la pieza para las decoraciones. Tanto el esmalte base como el engobe se suelen aplicar mediante campana, un ejemplo de aplicación puede verse en la Figura 6.13.



Figura 6.13 – Aplicación de esmaltes mediante campana.

Tras esta fase se suele soplar la pieza para eliminar posibles suciedades superficiales. También es necesario eliminar los restos de esmalte de los cuatro bordes de la pieza, por lo que la misma pasa por una secuencia de rascado, giro y rascado. Las máquinas de rascado eliminan la base y el engobe de los lados de la pieza, luego esta pieza se gira 90° para pasar por otra máquina de rascado y raspar los otros dos lados (Figuras 6.14 y 6.15).

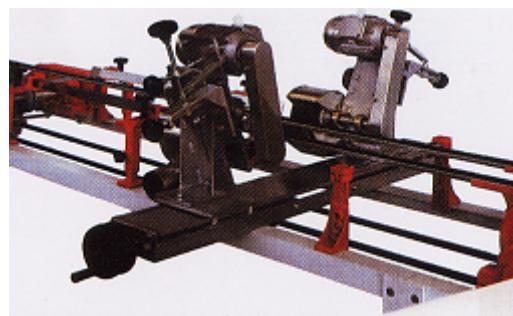


Figura 6.14 – Rascador por vía seca.



Figura 6.15 – Girador de baldosas.

Después de la aplicación de base y engobe y posteriormente al rascado, la pieza se decora. Existen varias máquinas y procedimientos que se utilizan a la hora de decorar la pieza, siendo los siguientes los más utilizados:

- Serigrafiado por pantallas: las piezas pasan por debajo de una pantalla, en la que hay fijada una tinta serigráfica, por encima de la pantalla pasa una espátula que fija la tinta a la pieza.
- Serigrafiado por rodillos: en este caso la tinta se encuentra en rodillos, el proceso es más rápido y permite una mayor flexibilidad.
- Granilladoras y aplicaciones en seco: aplicación de esmalte en polvo, que queda adherido sobre la pieza, con resultados rústicos y vistosos.
- Cepilladoras: dispositivos que cepillan el relieve del esmalte, dejando sin esmaltar las partes superficiales de la pieza, y quedando el fondo intacto.

Permiten decoraciones adicionales sobre la baldosa.

- Discos rotativos: el esmalte se aplica por centrifugación. Este aparato tiene mucha versatilidad y permite conseguir efectos muy variados.
- Fumé y aerógrafo: aplicación de esmaltes por nebulización, con o sin plantilla.
- Tubos goteadores, goteadores a taza, catapultas, etc...: otros tipos de máquinas utilizadas para decorar los azulejos.

En las Figuras 6.16, 6.17 y 6.18 se exponen algunos ejemplos de la maquinaria comentada.

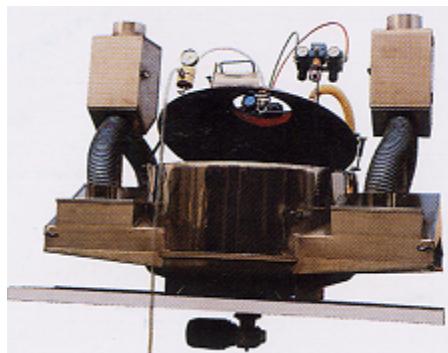


Figura 6.16 – Cabina aerógrafo.



Figura 6.17 – Máquina para cepillar vertical.

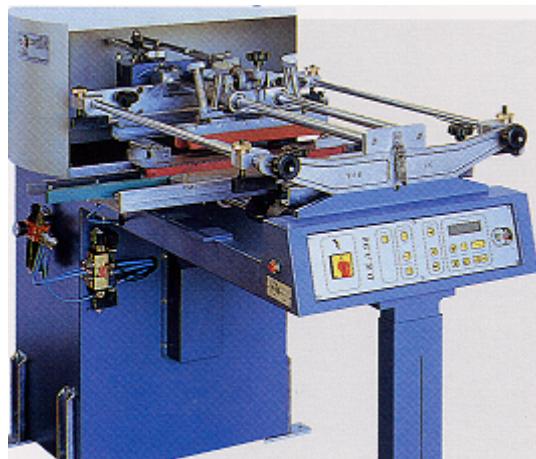


Figura 6.18 – Pantalla serigráfica o máquina de decorar.

6.4.12. Preparación de esmaltes

En la sección anterior vimos cómo se esmaltaban las piezas, pero los esmaltes se deben preparar previamente al comienzo de la producción. En la industria azulejera española, se vienen utilizando materias primas de naturaleza vítreo

(fritas), insolubles en agua, que se preparan a partir de materiales cristalinos sometidos a tratamientos térmicos de alta temperatura. Los esmaltes cerámicos tienen una proporción de fritada, estas fritas normalmente se presentan en sacos, donde la frita viene en pequeños cristales con forma de escama. Con estos sacos se alimentan molinos de bolas a los que se les añaden también colores, aditivos y agua. Tras la molienda, los esmaltes se almacenan en cubas agitadas, quedando preparados para su uso, estas cubas se conocen como “bombadas”, el tamaño de una bombada (o cantidad de metros de baldosa que se pueden esmaltar con la misma) es un factor que determina el tamaño del lote mínimo de producción.

Aparte de los esmaltes cerámicos, también es necesario preparar las tintas serigráficas, que son las que confieren las distintas decoraciones a la baldosa cerámica. Las tintas serigráficas se suelen preparar a partir de polvo base, el cual se dispersa en una mezcla de resinas, dando como resultado una pasta que se utiliza posteriormente en la decoración del azulejo.

6.4.13. Cocción

Sin duda, la cocción es la etapa central dentro del proceso de producción del azulejo y gran parte de las características finales del producto dependen de que ésta sea satisfactoria. Además, el horno es la maquinaria más cara de todo el proceso de producción, por lo que suele ser el cuello de botella en el sistema productivo. En la cocción la pieza sufre una serie de transformaciones físicas y químicas conforme atraviesa las distintas secciones del horno, cada una de ellas a una temperatura prefijada, por lo que tanto el ciclo del horno, como la atmósfera dentro del mismo son determinantes.

Ya se ha comentado que la cocción puede ser única lenta o rápida (monococción) o puede haber dos cocciones (bicocción, una cocción antes y otra después del esmaltado), incluso hay casos en los que el producto se cuece una tercera vez, a menor temperatura, procedimiento llamado de tercer fuego, normalmente necesario cuando se quieren aplicar vidriados o decoraciones adicionales. Actualmente la mayoría de los hornos son de tipo monoestrato de rodillos. En estos hornos las piezas se mueven en sentido horizontal, por encima de unos rodillos de acero o material refractario. El calor necesario dentro del horno se genera a partir de

quemadores de gas natural y de aire. La curva de temperatura dentro del horno se controla desde una consola informatizada, siendo esta curva característica del tipo de azulejo a producir. En este tipo de hornos se distinguen tres etapas diferenciadas:

- Calentamiento o precalentamiento: la humedad procedente del proceso de esmaltado se elimina (Figura 6.19).
- Cocción: se producen las transformaciones físicas y químicas necesarias en la pieza, así como una contracción de la baldosa (Figura 6.20).
- Enfriamiento forzado: se enfriá la pieza de forma acelerada para permitir su manipulación posterior (Figura 6.21).



Figura 6.19 – Horno, zona de precalentamiento.



Figura 6.20 – Horno, zona de cocción.



Figura 6.21 – Horno, zona de enfriamiento.

En los casos en los que la baldosa cerámica pasa por una etapa de secado después del esmaltado, es posible utilizar hornos con un ciclo de cocción más corto, dado que la etapa de precalentamiento ya no es necesaria.

6.4.14. Clasificación y embalaje

El proceso de producción de las baldosas cerámicas no termina cuando estas se cuecen. Todavía son necesarias una serie de acciones encaminadas a dejar el producto listo para su entrega.

Como hemos visto en secciones anteriores, el proceso de producción es largo y complejo, por tanto pueden aparecer múltiples problemas en el transcurso del mismo, problemas que muchas veces no se detectan hasta que el producto está terminado. Debido a esto, todos los azulejos, uno por uno, son inspeccionados para la detección de posibles defectos. Las inspecciones se pueden clasificar en dos tipos:

1. Inspecciones visuales: para detectar roturas, desconchados, falta de uniformidad en el color, pinchazos, etc...
2. Inspecciones automáticas: falta de planaridad, conicidad, etc...

Aparte de los posibles defectos, el proceso de producción, y sobre todo los ciclos de producción cortos, generan una falta de uniformidad en los colores (diversidad de tonos) y en los tamaños (diversidad de calibres). Para controlar este problema, se utilizan máquinas automáticas que clasifican la producción en tonos y calibres. Tras la inspección visual, los productos se clasifican normalmente en dos calidades, una primera, donde el producto está libre de defectos y otra segunda, donde aparecen defectos, pero en poca cantidad o de baja consideración. Si aparecen más defectos, o éstos son importantes, el producto se clasifica como desecho, aunque hay algunas empresas que los venden a precios muy reducidos. Después de la clasificación, el producto terminado se paletiza según calidad, calibre y tono, para el paletizado se utilizan máquinas como la de la Figura 6.22.



Figura 6.22 – Máquina paletizadora.

6.4.15. Otros aspectos de la producción de baldosas cerámicas

Entre las distintas fases del proceso de producción y entre las máquinas, el producto en curso puede viajar a través de líneas de transporte (cintas transportadoras), boxes o vagonetas. En el caso de los boxes o vagonetas, es frecuente el uso de sistemas automáticos para el movimiento de las mismas, como es el caso de los vehículos guiados automáticamente o AGVs, también llamados carros filoguiados. Estos vehículos están controlados mediante un ordenador que se comunica con ellos por radiofrecuencia o por láser (Figura 6.23).



Figura 6.23 – Carros filoguiados.

En el caso de utilizar cintas transportadoras, es común el uso de compensadores, mecanismos que permiten almacenar una pequeña cantidad de producto en curso, en el caso de que si se produce alguna parada en alguna máquina posterior, no sea necesario parar toda la línea de producción. (Figura 6.24).



Figura 6.24 – Compensador en línea.

Como hemos podido ver a lo largo de esta sección, el proceso de producción se encuentra hoy por hoy totalmente automatizado. Queda si cabe, la etapa de clasificación, que se hace por medios visuales, aunque esto está cambiando y se están empezando a instalar al final de las líneas de producción máquinas de clasificación automática que utilizan técnicas de visión artificial y que no necesitan de intervención humana. El primero de estos sistemas se instaló en 1998 en España en una conocida empresa de Castellón, y hoy hay más de 65 sistemas instalados (Figura 6.25).



Figura 6.25 – Sistema de clasificación automática.

Como veremos, la alta automatización del proceso productivo contrasta con la organización manual de la planificación y programación de la producción.

6.5. El problema de la programación de la producción en el sector cerámico

Tras el análisis del proceso de producción en el sector azulejero y los resultados de las encuestas presentados en Vallada et al. (2003a) y en Vallada et al.

(2003b), podemos detallar las siguientes características principales en el proceso de producción del azulejo y en las empresas del sector:

- La mayoría de las empresas utilizan el esquema productivo de monococción, con la preparación de las materias primas por vía húmeda.
- Es común que existan varias líneas de producción, hasta en las empresas más pequeñas. En las empresas grandes podemos encontrar hasta 10 o más líneas.
- Cuando existen varias líneas de producción lo común es que la empresa tenga dedicadas una o más a un tipo de pasta (pasta roja, blanca, gres, etc...).
- Existen muchas etapas productivas, pero es frecuente encontrar las prensas conectadas mediante cintas a los secaderos y éstos a su vez a las líneas de esmaltado.
- Las líneas de producción son polivalentes, es decir, pueden producir distintos tipos y formatos de azulejos, hasta incluso de distintas pastas.
- Los hornos suelen constituir los cuellos de botella del sistema productivo, aunque no siempre.
- Las empresas disponen de parques de boxes con producto pendiente de cocción (bizcocho verde) y producto cocido pendiente de clasificación, es decir, la etapa de cocción funciona de manera independiente a las demás.
- El horno está siempre en continuo funcionamiento.
- Cuando se quiere cambiar de tipo de azulejo en una línea de producción, se incurre en un elevado tiempo de cambio de partida, que además es dependiente de la secuencia.
- Prácticamente todos los productos siguen la misma ruta, pasan por todas las etapas.

- La gran mayoría de las empresas compra la arcilla atomizada a terceros. Las empresas que no la compran la producen en centros de producción especializados y participados con otras empresas.
- Casi ninguna empresa pequeña dispone de software para programar la producción y las que lo utilizan no pasan de simples hojas de cálculo o bases de datos. Sólo algunas de las empresas grandes tienen software a medida o ERP's.
- La gran mayoría de los encuestados opinan que las herramientas informáticas que utilizan no resuelven los problemas de producción de las empresas de manera satisfactoria.

Parece evidente que los resultados que se han comentado al inicio de este capítulo sobre los estudios realizados sobre diversos sectores industriales se repiten en el sector azulejero. Ninguna de las empresas encuestadas utiliza técnicas avanzadas para la programación de la producción. Sólo las empresas grandes usan sistemas integrados de gestión o ERP's tipo SAP o BaaN que emplean sistemas MRP o MRPII en sus módulos de producción. Se podría achacar esta falta de uso de herramientas especializadas a las empresas, pero ¿Realmente existe en el mercado software que incorpore métodos que las empresas puedan utilizar para programar la producción de manera satisfactoria? En la siguiente sección veremos que no se han propuesto métodos que de manera general puedan resolver este problema.

6.5.1. Caracterización del sistema productivo

El flujo de los azulejos en las líneas de producción sugiere un problema de tipo taller de flujo. El número de etapas viene determinado por los procesos que se realizan en la fabricación por monococción y preparación de materia prima por vía húmeda, que es, como se ha podido constatar a través de la encuesta comentada, el proceso productivo más utilizado. De una manera resumida, las etapas se muestran en la Figura 6.26.

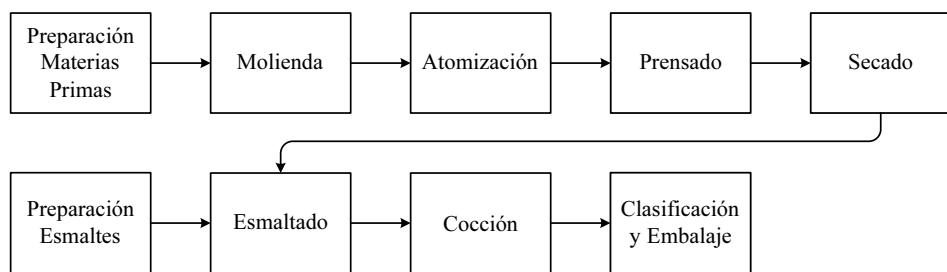


Figura 6.26 – Etapas de producción en el proceso de monococción con preparación de materias primas por vía húmeda.

Hemos comentado que las empresas subcontratan en buena parte de los casos la preparación de las materias primas y compran la arcilla ya atomizada. Además, este proceso es relativamente sencillo y se realiza en continuo, por lo que no representa en realidad un problema de programación de la producción, de manera que podemos realizar una primera simplificación y asumir que la arcilla atomizada se encuentra disponible para empezar el proceso de producción con el prensado. De manera similar, la preparación de los esmaltes es también un proceso relativamente sencillo y que se hace bajo demanda, es decir, se preparan los esmaltes que se necesitan y normalmente en la cantidad requerida para la producción. Luego los esmaltes que hay que preparar vienen determinados por la secuenciación de la producción. En este caso también podemos, sin pérdida de generalidad, separar el problema de la producción de los esmaltes del problema de la producción de los azulejos y asumir que los esmaltes y colores se encuentran disponibles cuando se pretende fabricar un tipo de azulejo.

Las dos simplificaciones anteriores no representan grandes problemas, dado que un tipo de azulejo necesita siempre un único tipo de pasta o de arcilla atomizada. Normalmente las empresas no trabajan con más de tres o cuatro tipos de pasta, por lo que el aprovisionamiento de arcilla atomizada es algo sencillo de llevar a cabo. Los esmaltes son algo más difíciles de controlar pero aún así los esmaltes necesarios para producir un azulejo se pueden producir con un día de antelación y por tanto estar listos a tiempo.

La citada encuesta a empresas cerámicas (ver Vallada et al., 2003a y Vallada et al., 2003b) ha contrastado que por lo general las prensas, secaderos y líneas de esmaltado se encuentran dispuestos en “líneas” de producción y conectadas entre sí mediante cintas transportadoras. De esta manera, el azulejo, una vez prensado, avanza por la línea hasta introducirse en el secadero de la línea para después esmaltarse en la misma línea. Luego existen tres máquinas pero realmente si se para la cinta transportadora ni se prensa ni se seca ni se esmalta. Es decir, las tres máquinas se comportan como una única máquina “virtual” por lo que sin pérdida de generalidad estas tres máquinas se pueden agrupar en una única máquina. De esta forma, el proceso de producción queda simplificado a tres únicas etapas tal y como se muestra en la Figura 6.27.

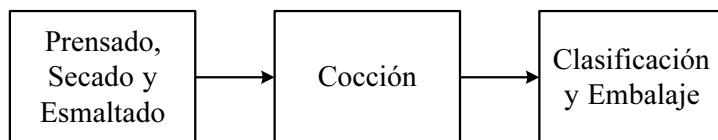


Figura 6.27 – Etapas de producción en el proceso simplificado de monococción con preparación de materias primas por vía húmeda.

El transporte antes y después del horno se hace en la gran mayoría de casos por carros filoguiados y el tiempo de transporte es muy corto en relación con el tiempo de proceso. En principio el proceso de producción del azulejo se podría caracterizar como un taller $F3//X$, es decir, tres únicas máquinas. No obstante, muchas empresas, sobre todo las grandes, disponen de más líneas de esmaltado que prensas o a veces las prensas no están directamente conectadas a los secaderos. También existen empresas pequeñas donde la salida del horno está conectada a la línea de clasificación. Con todas estas posibles situaciones **no** vamos a limitar los métodos propuestos a tres etapas, sino que en general consideraremos m etapas, aunque sepamos, que nunca tendremos menos de dos etapas y rara vez más de cinco.

Cuando una línea de prensado, secado y esmaltado está configurada para un determinado formato de azulejo (por ejemplo 33x33), existe un tiempo de cambio de partida considerable cuando se cambia a un azulejo de otras dimensiones. Este cambio se conoce como *cambio de formato* y puede llegar a ser de varias horas de duración. La máquina que más intervención requiere es la prensa, dado que es necesario desmontarla parcialmente, cambiar moldes, punzones, y demás aspectos y luego volverla a montar. Normalmente también es necesario hacer ajustes en el secadero y línea de esmaltado. Cuando lo que se quiere es producir un azulejo de otras características pero de idénticas dimensiones lo que se ajusta es la línea de esmaltado. Este tiempo de cambio, llamado *cambio de modelo* también puede llegar a ser de horas, pero normalmente es menor que el cambio de formato. Con todo esto tenemos que aparecen, de manera muy importante, tiempos de cambio de partida dependientes de la secuencia, por lo que el problema a resolver se denota entonces como $F/S_{sd}/X$.

De esta manera, parece que el criterio más útil para la programación de la producción es la minimización del tiempo máximo de flujo o C_{max} , criterio que se ha venido utilizando en esta Tesis Doctoral.

Hasta aquí se podrían utilizar los algoritmos presentados en el Capítulo 5 pero desgraciadamente la realidad continúa siendo más compleja. Como hemos comentado, las empresas disponen de varias líneas de producción, de manera que, y por poner un ejemplo, un azulejo se puede prensar, secar y esmaltar en la línea 1, cocerse en el horno 3 y clasificarse en la línea 2, dado que el sistema permite esta flexibilidad. Esta condición establece una nueva dimensión al problema, ya que ahora, en vez de hablar de máquinas hablamos de etapas y un trabajo no se procesa en las m máquinas sino que se procesa en m etapas y dentro de cada una se procesa en alguna de las máquinas disponibles. Estos problemas se denominan talleres de flujo con múltiples procesadores, talleres de flujo híbridos, talleres de flujo flexibles o incluso en algunos casos líneas de flujo flexibles, aunque veremos que no todos estos términos se refieren exactamente al mismo problema. Antes de continuar con la caracterización del problema de producción en las empresas cerámicas vamos a concretar más este tipo de problemas.

6.5.2. El taller de flujo híbrido

En un taller de flujo con múltiples procesadores o FSMP tenemos un conjunto N de trabajos, $N = (1, \dots, n)$, que hay que procesar en un conjunto M de etapas, $M = (1, \dots, m)$ y en cada etapa i , $i = (1, \dots, m)$ existe un número $m_i \geq 1$ de máquinas idénticas que pueden procesar los trabajos. Cada trabajo debe procesarse en todas las etapas y dentro de cada etapa en una única máquina. Como vemos, la “única” diferencia con el taller de flujo estándar es que para cada fase o etapa hay varias máquinas que pueden realizar el trabajo. En realidad un FSMP con $m_i = 1$, $i = (1, \dots, m)$ es equivalente al taller de flujo estándar. Un ejemplo esquemático de este problema puede verse en la Figura 6.28.

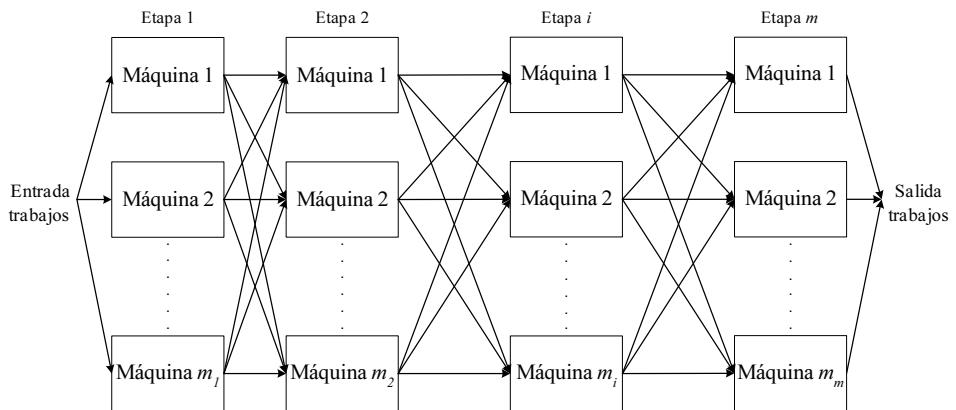


Figura 6.28 – Esquema del problema del taller de flujo con múltiples procesadores o FSMP.

Este problema se estudió por primera vez en Arthanary y Ramaswamy (1971) y en Salvador (1973) y Brah y Hunsucker (1991) demostraron que existen un número muy elevado de posibles rutas para los trabajos, concretamente

$$\prod_{i=1}^m \binom{n-1}{m_i-1} \cdot \frac{n!}{m_i!}$$

Como podemos observar, el número de posibilidades es enorme. Ya sabemos que el taller de flujo de permutación tiene $n!$ posibles soluciones y si tenemos $n = 10$ y $m = 5$ tendremos 3.628.800 posibles secuencias. Un problema similar en un entorno FSMP con $n = 10$ y con tres etapas de 2, 1 y 2 máquinas respectivamente (cinco en total) resultaría exactamente en 9,67641E+20 posibles soluciones, luego es fácil hacerse una idea de que el problema FSMP es mucho más difícil que el ya de por sí complicado taller de flujo. De hecho Gupta (1988) demostró que el problema FSMP con solo dos etapas ($m = 2$) es \mathcal{NP} -Completo incluso en el caso de que alguna de las dos etapas contenga una sola máquina. Incluso proporcionó evidencia de que el problema es \mathcal{NP} -Completo en sentido estricto.

Hasta ahora hemos comentado el caso en el que las m_i máquinas de cada etapa son idénticas y procesan los trabajos a la misma velocidad (máquinas *paralelas*), de ahí que la definición de tiempos de proceso o p_{ij} todavía sirva dado que son los tiempos de proceso del trabajo j en la etapa i . De nuevo estamos ante una simplificación *fuerte* de la realidad. Lo más normal en una empresa es que si se adquiere una máquina adicional para ampliar una etapa de producción que es cuello de botella, ésta nueva máquina sea más moderna y por tanto previsiblemente más rápida, o más específica y por tanto más rápida para unos trabajos y más lenta para otros. De esta manera se conoce en la literatura por taller de flujo híbrido o “*hybrid flow shop*” como el problema FSMP donde las máquinas en cada etapa son no relacionadas y pueden procesar cada trabajo a una velocidad distinta, en esta situación $p_{i,lj}$ indica el tiempo de proceso del trabajo j en la máquina l de la etapa i donde $i = (1, \dots, m)$ y $l = (1, \dots, m_i)$. De todas formas, algunos autores no hacen distinción entre el taller de flujo híbrido y el FSMP, confundiendo ambos términos. Es evidente que el taller de flujo híbrido o HFS es también \mathcal{NP} -Completo puesto que si hacemos que todas las máquinas sean idénticas en cada etapa tendremos un problema de tipo FSMP que ya hemos visto que es \mathcal{NP} -Completo.

El problema HFS no es simplemente una sencilla extensión del problema FSMP, de nuevo, este acercamiento a la realidad conlleva un aumento notable de

la dificultad. Gourgand, Grangeon y Norre (1999) demostraron que el número de posibles soluciones en un problema de tipo HFS es muy elevado, concretamente igual a:

$$n! \left(\prod_{i=1}^m m_i \right)^n$$

Lo que produce muchas más posibles soluciones que el problema FSMP, sobre todo cuando el número de máquinas totales $M = \sum_{i=1}^m m_i$ es grande. Algunos autores llaman al HFS taller de flujo flexible o “*flexible flow shop*”, aunque también a veces este último término se confunde con el problema FSMP, luego como vemos, no existe una nomenclatura única aceptada para referirse a problemas de tipo taller de flujo con etapas y varias máquinas por etapa.

Otros autores definen el problema de líneas de flujo flexibles o “*flexible flow lines*” (FFL). En este tipo de problema algunos trabajos pueden saltarse una etapa de producción y también es posible que pasen más de una vez por alguna de las etapas. El primer caso se puede contemplar haciendo cero los tiempos de proceso para el trabajo en todas las máquinas de la etapa que se pretende saltar, aunque esto dependerá del algoritmo que se utilice. El segundo caso es mucho más difícil de tener en cuenta. El problema FFL se puede considerar una extensión a múltiples máquinas del taller de flujo generalizado o “*general flow shop*” (ver Figura 3.2 del Capítulo 3). El problema FFL no ha sido muy tratado en la literatura, casi todo el trabajo se puede encontrar en Wittrock (1985), Kochhar y Morris (1987) y Wittrock (1988).

La notación expuesta en la Sección 2.1 del Capítulo 2 no contempla la correcta definición de los problemas FSMP o HFS vistos anteriormente, dado que no nos permite especificar cuántas máquinas tenemos por etapa o si éstas son idénticas o no relacionadas. Vignier, Billaut y Proust (1999) propusieron una ampliación de la notación basada en la tripleta $\alpha/\beta/\gamma$. Concretamente, la extensión estriba en el campo α , que se define como: $\alpha = \alpha_1\alpha_2((\alpha_3\alpha_4^{(i)})_{i=1}^{\alpha_2})$, de manera que el campo α_1 indica el tipo de problema, que cuando es igual a FH indica un taller de flujo de tipo híbrido. El campo α_2 indica el número de etapas de producción que tiene el problema. Los campos α_3 y α_4 se repiten, por pares, tantas veces como

etapas tenga el problema (campo α_2), donde α_3 indica cómo son las máquinas en la etapa, paralelas, uniformes o no relacionadas (P, Q, R) y el campo α_4 indica cuántas máquinas hay en la etapa.

De esta forma, el ejemplo de problema de tipo FSMP expuesto anteriormente, y para el que se han calculado el total de posibles secuencias, se denotaría como $FH3, ((P2^{(1)}), (P1^{(2)}), (P3^{(3)})) // C_{max}$ en el caso de que se pretenda minimizar el C_{max} . De manera general, si la configuración de las máquinas es la misma para todas las etapas se podría hacer $FH3, ((P2^{(i)})_{i=1}^{(3)}) // C_{max}$ para indicar un FSMP con 3 etapas y dos máquinas paralelas por etapa.

Con esta notación ya podemos especificar de una manera más concisa el problema de programación de la producción en las empresas cerámicas. Si tenemos en cuenta que prácticamente todas las empresas encuestadas con más de una línea de producción disponen de máquinas no relacionadas en cada etapa y la existencia de los tiempos de cambio de partida comentados anteriormente, el problema se denotaría por $FHm, ((RM^{(i)})_{i=1}^{(m)}) / S_{sd} / C_{max}$.

Adicionalmente, cuando tenemos máquinas no relacionadas en las etapas, es frecuente que algunas de estas máquinas no puedan procesar todos los trabajos, más concretamente, y en el sector cerámico, se dan dos casos extremos con relativa asiduidad: el caso en que un tipo de azulejo sólo puede procesarse en una línea y el caso en que un tipo de azulejo puede procesarse en todas las líneas menos en una, aunque situaciones mixtas también son frecuentes. Esta situación, como ya viéramos en el Capítulo 2, se conoce como restricción de uso de máquinas. Luego el problema completo en el sector cerámico, con muy pocas simplificaciones y nunca fuertes, se denota por $FHm, ((RM^{(i)})_{i=1}^{(m)}) / S_{sd}, M_j / C_{max}$. Notar que en el caso del taller de flujo híbrido no tiene sentido hablar de secuencias de permutación, ya que el orden de entrada de los trabajos en cada etapa no tiene por qué traducirse en el mismo orden de salida de estos mismos trabajos dado que el tercer trabajo en entrar puede ocupar una máquina más lenta que el cuarto trabajo, y éste último salir antes de la etapa, por poner un ejemplo.

Para resolver los problemas híbridos tenemos dos *dimensiones*, la *secuenciación* de los trabajos en el taller, es decir, en qué orden van a llegar éstos a las etapas, y la *asignación*, o cómo se asignan los trabajos a las máquinas dentro de una etapa.

Normalmente, y debido a la complejidad de estos problemas, los autores separan ambas dimensiones, realizando primero la secuenciación y luego la asignación o viceversa, pero casi nunca se consideran las dos dimensiones simultáneamente. En la siguiente sección veremos qué métodos se pueden utilizar para resolver este problema de manera satisfactoria.

6.5.3. Métodos para la programación de la producción en talleres de flujo híbrido

Uno de los primeros trabajos sobre talleres híbridos se debe a Arthanary y Ramaswamy (1971) donde se desarrolla una algoritmo de bifurcación y acotación que proporciona la solución exacta para un sencillo problema que se asemeja a un FSMP con dos etapas y con una única máquina en la segunda etapa. Poco después, Salvador (1973) propuso algoritmos de programación dinámica para el problema FSMP donde los trabajos no pueden esperar entre etapas y donde se considera el criterio de la minimización del C_{max} , por tanto el problema considerado es $FHm, ((PM^{(i)})_{i=1}^{(m)})/nwt/C_{max}$. El autor aplicó los resultados de los algoritmos a la programación de la producción en una empresa de polimerización de nylon con bastante éxito, aunque las técnicas expuestas sólo son útiles para problemas muy pequeños.

Wittrock (1985), desarrolló una serie de algoritmos para el problema FFL donde algunos trabajos pueden saltarse una de las etapas de la producción y donde las máquinas dentro de una etapa son idénticas. En principio considera tres etapas productivas y separa el problema en tres dimensiones: asignación, secuenciación y programación de las operaciones y los resuelve por este orden. Los problemas ejemplo que resuelven son muy específicos y también las técnicas desarrolladas. El autor también incluye la problemática de la lotificación en el estudio. Los criterios de optimización que se utilizan son la maximización de la producción, que en el contexto de este trabajo es equivalente a la minimización del C_{max} .

Kochhar y Morris (1987) resuelven un problema muy concreto de líneas de flujo flexibles donde se consideran tiempos de cambio, bloqueo y rotura aleatoria de las máquinas. Separan el problema en secuenciación y asignación que resuelven en este orden. Los métodos propuestos se basan en búsqueda local descendente

y en heurísticas específicas para el problema. El criterio considerado es el \bar{F} (minimización del tiempo medio de flujo).

Wittrock (1988) extendió el anterior estudio realizado (ver Wittrock, 1985) para el caso donde el almacenamiento entre etapas es limitado. El autor propone una heurística concreta para el problema. El criterio de optimización es doble pero no bicriterio, es decir, se obtienen varias secuencias con un mínimo C_{max} y de éstas se escoge también la que tiene la menor cantidad de producto en curso.

Gupta (1988) desarrolló técnicas heurísticas para un problema FSMP muy simplificado de dos etapas donde solo hay dos máquinas paralelas idénticas en la primera etapa y una única máquina en la segunda. Las técnicas presentadas se basan en la regla de Johnson. De manera parecida, Sriskandarajah y Sethi (1989) resuelven dos problemas similares, uno con una máquina en la primera etapa y un número m de máquinas en la segunda y otro problema de dos etapas con cualquier número de máquinas en cada etapa. Proponen algoritmos de secuenciación por listas y también algoritmos basados en la regla de Johnson.

Sherali, Sarin y Kodialam (1990) trabajaron con un problema real de dos etapas en la industria papelera. Este es el primer trabajo en considerar tiempos de cambio de partida dependientes de la secuencia y en las dos etapas. Los autores dividen el problema en secuenciación y asignación y proponen diversos modelos matemáticos para resolver la problemática. Los autores se centran demasiado en el problema real que tiene 10 máquinas en la primera etapa y 12 en la segunda y los modelos propuestos no son fácilmente extensibles a casos con más etapas. Los autores consideran una función de costes de cambio y de costes de asignación de los trabajos a las máquinas como criterio de optimización.

Otro ejemplo de aplicación industrial es el trabajo de Guinet (1991), donde se desarrollan algoritmos para el problema de la programación de la producción en la producción de telas. El autor divide también el problema general en subproblemas que resuelve mediante la aplicación de heurísticas concretas para el caso considerado. El objetivo de optimización es la minimización del C_{max} y también del \bar{T} (minimización del retraso medio), aunque no se consideran de manera conjunta sino por separado en distintos métodos. De nuevo las heurísticas son demasiado específicas para el problema y el autor se limita a resolver ejemplos de problemas reales sin realizar ningún tipo de evaluación objetiva.

Gupta y Tunc (1991) exponen dos heurísticas basadas en reglas de prioridad para resolver el problema simétrico al expuesto en Gupta (1988), es decir, donde solo hay una máquina en la primera etapa y 2 máquinas idénticas en la segunda etapa. Los autores dividen el problema en secuenciación y asignación.

Brah y Hunsucker (1991) propusieron un algoritmo de tipo bifurcación y acotación para el FSMP general, esto es, con m etapas y cualquier número de máquinas idénticas por etapa. Los autores también proporcionaron una serie de cotas inferiores y superiores para el problema. Como cualquier aproximación exacta, el algoritmo propuesto sólo es útil para problemas muy pequeños y aún así el tiempo de CPU necesario es elevado debido a la complejidad algorítmica.

Algunos autores han estudiado el comportamiento de las reglas de prioridad aplicados a distintos problemas, por ejemplo Hunsucker y Shah (1992) evaluaron seis sencillas reglas de prioridad en un caso especial del FSMP, el FSMP “restringido” donde se limitan el número de trabajos que pueden estar simultáneamente en el taller. A partir del estudio, la regla LPT parece ser mejor con el criterio de optimización \bar{T} y para el criterio N_T (minimización del número de trabajos retrasados) no domina ninguna regla en especial.

Chandrasekharan y Chaudhuri (1992a) desarrollaron también un algoritmo de bifurcación y acotación para el FSMP pero en este caso el criterio de optimización considerado es el F_{max} . Ante la imposibilidad de resolver problemas de unos pocos trabajos y máquinas en un tiempo razonable, los autores también propusieron una sencilla heurística. En un trabajo similar (ver Chandrasekharan y Chaudhuri, 1992b), los autores aplicaron el estudio al mismo problema pero con el criterio de la minimización del C_{max} . Lo interesante de este trabajo es que se aplica un algoritmo de bifurcación y acotación para resolver el problema de la secuenciación, es decir, se resuelve un taller de flujo estándar, para luego realizar la asignación de los trabajos en cada etapa a la primera máquina que quede disponible. Esta solución consigue reducir los tiempos de CPU del algoritmo, pero evidentemente a costa de la optimalidad.

Adler et al. (1993) desarrollaron un sistema llamado BPSS para la programación de la producción en una empresa de fabricación de bolsas y sacos de papel. El sistema se basa en la identificación de cuellos de botella en la producción y en la aplicación de reglas de prioridad aplicadas al caso. El sistema es muy específico

para la empresa y considera tiempos de cambio de partida dependientes de la secuencia y máquinas no relacionadas, aunque no en todas las etapas.

Hunsucker y Shah (1994) vuelven a evaluar el FSMP restringido en un estudio muy similar al citado anteriormente (ver Hunsucker y Shah, 1992) pero esta vez añadiendo los criterios de C_{max} , F_{max} y \bar{F} . Los resultados indican que la regla SPT proporciona mejores resultados que las otras 5 reglas evaluadas en los criterios de C_{max} y \bar{F} mientras que las reglas SPT y FIFO resultan ser las mejores con el criterio F_{max} .

Otra aplicación real puede encontrarse en el trabajo de Aghezzaf et al. (1995). En este caso se proponen una serie de métodos para dar solución a un taller de flujo híbrido de tres etapas con máquinas no relacionadas y tiempos de cambio de partida dependientes de la secuencia. El problema modeliza la fabricación de alfombras en una empresa real. Los autores construyen un modelo de programación matemática para la secuenciación y asignación para luego descomponer estos dos problemas ya que el modelo resultante es intratable. Se desarrolla una heurística que asigna los trabajos a máquinas en primer lugar y luego se aplican heurísticas de secuenciación en problemas de una sola etapa y máquinas paralelas, de manera que cada una de las tres etapas del problema completo se resuelve independientemente de las otras. El problema que se resuelve en este trabajo es muy parecido (conceptualmente) al problema de producción de azulejos, el único problema es que los métodos desarrollados son muy específicos para el problema de producción de alfombras y además, no se dan apenas detalles de los métodos desarrollados así como tampoco ninguna evaluación ni tablas de resultados.

En el trabajo de Santos, Hunsucker y Deal (1995) no se proponen algoritmos completos, sino una serie de cotas inferiores para el problema FSMP con máquinas idénticas en cada etapa. Los autores simplemente hacen esta aportación para que se puedan utilizar estas cotas como soluciones de referencia en comparativas y desarrollos, dado que las soluciones óptimas para estos problemas, sean cuales sean los problemas de ejemplo, son muy difíciles de obtener.

Guinet y Solomon (1996) publicaron un trabajo muy interesante para resolver el problema FSMP de máquinas idénticas. Los autores dividen el problema en secuenciación y asignación y los criterios que se consideran son el C_{max} y el T_{max} (por separado). El interés estriba en que para resolver el problema de

secuenciación aplican reglas estándares como la NEH o CDS y luego hacen la asignación de los trabajos en las etapas conforme a simples reglas de prioridad. El método resultante es muy sencillo y fácil de implementar. Las simulaciones indican que la heurística NEH proporciona una buena secuenciación y es superior a la CDS.

Un trabajo muy similar se debe a Santos, Hunsucker y Deal (1996) donde la solución que se propone es exactamente la misma, utilizar una heurística común para la secuenciación de los trabajos en el taller y luego realizar la asignación a las máquinas dentro de cada etapa siguiendo la regla de prioridad FIFO. En este caso los autores consideran las heurísticas RA, Palmer, CDS, y la de Gupta (ver Capítulo 3), resultando las heurísticas CDS y RA las mejores de la comparativa. En la línea de estos trabajos, Guinet y Solomon (1996) propusieron una solución similar para el problema FSMP con dos etapas únicamente. Aparte de proponer un modelo de programación entera con variables binarias y algunas cotas inferiores para el problema, los autores dividen también el problema en secuenciación y asignación. En este caso, la regla de Johnson parece ser la que mejor funciona para la etapa de asignación de este problema con dos etapas.

Gupta, Hariri y Potts (1997) tratan de nuevo el problema FSMP con dos etapas donde en la segunda etapa sólo hay una máquina. Los autores proponen un algoritmo nuevo de bifurcación y acotación junto con nuevas cotas superiores e inferiores. También se proponen 4 algoritmos heurísticos, uno de ellos basado en búsqueda local descendente. En un trabajo similar, Gupta y Tunc (1998) evalúan el caso simétrico, donde la primera etapa tiene una máquina y donde el criterio a minimizar es N_T . En este artículo se proponen seis métodos heurísticos y se hace una extensa comparativa entre ellos.

Una aproximación diferente es la que siguen Nowicki y Smutnicki (1998) que proponen una ampliación del algoritmo de búsqueda tabú propuesto para el problema del taller de flujo estándar (ver Nowicki y Smutnicki, 1996). Los autores hacen uso de las propiedades del camino crítico en una secuencia para el FSMP donde se busca minimizar el C_{max} . El algoritmo resultante es intrincado y muy difícil de codificar, de igual manera, extensiones para la consideración de tiempos de cambio de partida no son nada triviales, tampoco el resolver problemas con máquinas no relacionadas. Un trabajo muy similar de los mismos autores apareció

publicado poco después (ver Nowicki y Smutnicki, 1999).

En Portmann et al. (1998) se propone un algoritmo de bifurcación y acotación parecido al de Brah y Hunsucker pero con algunas mejoras muy importantes. Se utiliza un GA hibridizado en la fase de ramificación del algoritmo para mejorar las estimaciones de las cotas inferiores y superiores, de esta manera se consigue acotar más el árbol de búsqueda. El algoritmo híbrido resultante, aunque bastante complejo, es superior al de Brah y Hunsucker.

Otro algoritmo de bifurcación y acotación, esta vez estándar, se muestra en Riane, Artiba y Elmaghraby (1998). En este caso se resuelve un problema muy concreto de una industria, un FSMP con tres etapas, donde tenemos una, dos y una máquina en cada etapa respectivamente. Los autores también proponen una heurística basada en programación dinámica.

Gourgand, Grangeon y Norre (1999) proponen algoritmos heurísticos basados en simulated annealing para resolver el problema del taller de flujo híbrido con máquinas en cada etapa que no tienen por qué ser idénticas. Los autores diseñan un interesante tipo de vecindario que une los problemas de secuenciación y de asignación, por lo que los algoritmos propuestos resuelven ambos problemas simultáneamente. Los resultados obtenidos utilizando algunos sencillos problemas son buenos y los autores aplican el algoritmo propuesto en un entorno industrial real.

Brah y Loo (1999) utilizan la técnica ya comentada de secuenciar los trabajos de acuerdo a una heurística común y luego hacer la asignación de los trabajos dentro de las etapas con una regla de prioridad. La diferencia estriba en que en este caso el criterio de optimización utilizado es el \bar{F} y por tanto las heurísticas que se utilizan para el problema de la secuenciación incluyen métodos específicos para el problema $F//\bar{F}$ (ver Capítulo 3).

Más recientemente, Botta-Genoulaz (2000) propuso seis nuevas heurísticas para un tipo de problema FSMP muy específico que incluye lapsos en las máquinas, restricciones de precedencia en los trabajos, fechas de finalización, tiempos de limpieza y muchas otras situaciones. El autor también evalúa los métodos en el ámbito de un taller de flujo estándar. El criterio de optimización utilizado es la minimización de la tardanza máxima o L_{max} .

Finalmente, Soewnadi y Elmaghraby (2001) han propuesto heurísticas sencillas

y rápidas para obtener soluciones de mínimo C_{max} en el FSMP con tres etapas y cualquier número de máquinas idénticas por etapa. Las heurísticas propuestas se comparan con unas nuevas cotas inferiores que se desarrollan también en este trabajo.

Un trabajo importante es el de Andrés (2001), que se centra en el taller de flujo híbrido con tiempos de cambio dependientes de la secuencia y además propone métodos de resolución para un problema de una empresa azulejera. El autor hace varias propuestas de modelos de programación matemática, reglas de prioridad y métodos metaheurísticos, como colonias de hormigas y simulated annealing. Los resultados de implantación en la empresa en cuestión son buenos, aunque los métodos propuestos se centran en tres etapas, no contemplan la elegibilidad de máquinas y supone que todas las máquinas en cada etapa son idénticas.

En la Tabla 6.2 se recogen todos los trabajos citados ordenados cronológicamente y se comentan sus principales características.

Año	Autor/es	Problema	Comentarios
1971	Arthanary y Ramaswamy	$FH2, ((PM^{(1)}), (P1^{(2)})) / / C_{max}$	bifurcación y acotación, sólo para problemas pequeños
1973	Salvador	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / nwt / C_{max}$	programación dinámica, sólo para problemas pequeños
1985	Wittrock	$FH3, ((PM^{(i)})_{i=1}^{(3)}) / / C_{max}$	problema específico
1987	Kochhar y Morris	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / block, brkdwn, S_{nsd} / \bar{F}$	problema específico, búsqueda local
1988	Wittrock	$FH3, ((PM^{(i)})_{i=1}^{(3)}) / block / C_{max}$	problema específico
	Gupta	$FH2, ((P2^{(1)}), (P1^{(2)})) / / C_{max}$	heurísticas basadas en la regla de Johnson
1989	Sriskandarajah y Sethi	$FH2, (((PM^{(i)})_{i=1}^{(2)}) / / C_{max}$	heurísticas basadas en listas de secuenciación
1990	Sherali, Sarin y Kodialam	$FH2, (((PM^{(i)})_{i=1}^{(2)}) / S_{sd} / Costes$	problema específico, modelos de programación entera y lineal
1991	Guinet	$FHm, (((PM^{(i)})_{i=1}^{(m)}) / S_{sd} / C_{max}, \bar{T}$	problema específico, heurísticas ad-hoc

Año	Autor/es	Problema	Comentarios
	Gupta y Tunc	$FH2, ((P1^{(1)}, (P2^{(2)})) / / C_{max})$	dos heurísticas basadas en reglas de prioridad
	Brah y Hunsucker	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / / C_{max}$	bifurcación y acotación, sólo para problemas pequeños
1992	Hunsucker y Shah	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / / \bar{T}, N_T$	FSMP restringido, reglas de prioridad
	Chandrasekharan y Chaudhuri	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / / F_{max}$	bifurcación y acotación, sólo para problemas pequeños
	Chandrasekharan y Chaudhuri	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / / C_{max}$	bifurcación y acotación, basado en taller de flujo estándar
1993	Adler et al.	$FHm, ((RM^{(i)})_{i=1}^{(m)}) / / S_{sd} / \sum_{j=1}^n w_j T_j$	problema específico, reglas de prioridad
1994	Hunsucker y Shah	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / / C_{max}, F_{max}$ y \bar{F}	FSMP restringido, reglas de prioridad
1995	Aghezzaf et al.	$FHm, ((RM^{(i)})_{i=1}^{(m)}) / / S_{sd} / C_{max}, F_{max}$ y \bar{F}	problema específico, heurísticas ad-hoc
1996	Guinet y Solomon	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / / C_{max}, T_{max}$	heurísticas y reglas de prioridad
	Santos, Hunsucker y Deal	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / / C_{max}$	heurísticas y reglas de prioridad
	Guinet y Solomon	$FH2, ((PM^{(i)})_{i=1}^{(2)}) / / C_{max}$	heurísticas y reglas de prioridad
1997	Gupta, Hariri y Potts	$FH2, ((PM^{(1)}, (P1^{(2)})) / / C_{max})$	heurísticas y algoritmos de bifurcación y acotación
1998	Gupta y Tunc	$FH2, ((P1^{(1)}, (PM^{(2)})) / / N_T)$	heurísticas simples
	Nowicki y Smutnicki	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / / C_{max}$	TS basado en bloques de trabajos y camino crítico
	Portmann et al.	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / / C_{max}$	bifurcación y acotación mezclada con GAs
	Riane, Artiba y Elmaghhraby	$FH2, ((P1^{(1)}, (P2^{(2)}), (P3^{(1)})) / / C_{max})$	bifurcación y acotación, programación dinámica
1999	Gourgand, Grangeon y Norre	$FHm, ((RM^{(i)})_{i=1}^{(m)}) / / C_{max}$	heurísticas basadas en SA

Año	Autor/es	Problema	Comentarios
	Brah y Loo	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / / \bar{F}$	heurísticas y reglas de prioridad
2000	Botta-Genoulaz	$FHm, ((PM^{(i)})_{i=1}^{(m)}) / S_{nsd}, lags, prec, R_{nsd} / C_{max}$	problema específico, heurísticas
2001	Soewnadi y Elmaghrary	$FH3, ((PM^{(i)})_{i=1}^{(3)}) / / C_{max}$	heurísticas
	Andrés	$FH3, ((P4^{(1)}), (P2^{(2)}), (P3^{(3)})) / S_{sd} / C_{max}$	heurísticas y metaheurísticas

Tabla 6.2 – Métodos para el problema del taller de flujo híbrido.

Como hemos podido comprobar, gran parte de la literatura se centra en casos muy simplificados del FSMP donde se consideran solamente dos etapas. Cuando se contemplan m etapas, las máquinas dentro de cada una se asumen idénticas. Sin embargo, la realidad del sector cerámico indica lo contrario. Únicamente conocemos tres trabajos (Adler et al., 1993, Aghezzaf et al., 1995 y Gourgand, Grangeon y Norre, 1999) donde se consideran máquinas no relacionadas en cada etapa y sólo dos de estos casos consideran tiempos de cambio de partida dependientes de la secuencia. Desgraciadamente, en Adler et al. (1993) se resuelve un problema específico de una empresa concreta y los métodos propuestos están adaptados al caso y en Aghezzaf et al. (1995) se dan pocos detalles del método implementado y también se puede decir que es muy concreto para el problema de producción de alfombras. En el trabajo de Gourgand, Grangeon y Norre (1999) no se consideran los tiempos de cambio de partida dependientes de la secuencia. No conocemos de ningún trabajo donde se considere la elegibilidad de máquinas dentro de cada etapa, y menos si añadimos los tiempos de cambio dependientes de la secuencia y máquinas no relacionadas en cada etapa. Por tanto, hemos desarrollado un algoritmo general, que es una adaptación al problema del sector cerámico de los algoritmos genéticos propuestos en capítulos anteriores.

6.5.4. Algoritmos genéticos y el problema de la programación de la producción en el sector cerámico

Hemos comentado en secciones anteriores como el problema del sector cerámico, denotado por $FHm, ((RM^{(i)})_{i=1}^{(m)})/S_{sd}, M_j/C_{max}$ no se ha tratado con anterioridad y además es muy difícil de resolver por las dos dimensiones que tiene; *secuenciación* y *asignación*. En un problema de naturaleza combinatoria con un espacio de soluciones posibles tan grande es necesario realizar algunas simplificaciones. La simplificación que más hemos observado en la literatura es separar ambas dimensiones y resolverlas por separado. Nosotros escogemos un camino diferente, vamos a realizar una secuenciación teniendo en cuenta una forma de realizar la asignación concreta. Es decir, no vamos a separar completamente la secuenciación de la asignación.

Nuestro objetivo es intentar que el algoritmo resultante sea “sencillo” y “general”. Es decir, un método que resuelva el problema $FHm, ((RM^{(i)})_{i=1}^{(m)})/S_{sd}, M_j/C_{max}$, pero que no solucione sólo los problemas del sector cerámico (tres etapas y con una estructura determinada de tiempos de proceso y tiempos de cambio) sino que pueda ser utilizado en otros sectores. Por ejemplo en el sector donde el problema de fabricación de telas puede modelizarse también como un $FHm, ((RM^{(i)})_{i=1}^{(m)})/S_{sd}, M_j/C_{max}$ dado que se dan las mismas situaciones que en el sector azulejero en cuanto a la disposición de las líneas productivas (algunos tipos de telas no pueden tejerse en un telar, existen tiempos de cambio dependientes de la secuencia y normalmente los distintos telares y urdidoras en una etapa procesan las telas a diferentes velocidades).

Hemos diseñado un algoritmo genético para resolver el problema anteriormente citado. Se trata de una adaptación del GA expuesto en el Capítulo 4 y del algoritmo GA_{sd} del Capítulo 5, y lo llamaremos GA_H .

6.5.4.1. Representación genética y función de evaluación

Inicialmente, consideraremos el problema de la secuenciación, por lo que seguiremos utilizando la representación ordinal, esto es, una simple permutación de todos los trabajos que se quieren procesar (ver Secciones 4.2.1 y 4.3.1).

La función de evaluación es la que va a sufrir importantes cambios y es donde vamos a incorporar las decisiones de asignación. Normalmente los trabajos citados realizan una secuenciación resolviendo el problema como si fuese un taller de flujo estándar y cuando se ha obtenido una solución, entonces se lleva a cabo la asignación. Funcionando de esta manera provocamos una separación total de ambos problemas. Nosotros proponemos incorporar las decisiones de asignación en la función de evaluación del algoritmo genético, por lo que de alguna manera el GA estará contemplando ambos problemas simultáneamente, aunque no de manera explícita. La forma más explícita de resolver ambos problemas sería incorporar en la representación genética la asignación de los trabajos a las máquinas dentro de cada etapa (como se hace en Gourgand, Grangeon y Norre, 1999). Esta representación alternativa supone añadir un vector de tamaño m para cada uno de los n trabajos donde los genes indican la máquina a la que se asigna el trabajo dentro de cada una de las m etapas. Experimentos iniciales con este tipo de representación dieron muy malos resultados, lo que indica que son necesarios nuevos operadores de cruce y mutación, así como una completa revisión del diseño del GA. Alternativamente, podemos hacer algunas adaptaciones al GA y cambiar la función de evaluación para que sea ésta la que realize la asignación. Si la asignación resulta en un C_{max} alto, el individuo será poco interesante y terminará por ser sustituido en la población.

La asignación se puede realizar de varias maneras distintas una vez se tiene la secuenciación de los trabajos y se quiere determinar qué trabajo se asigna y a qué máquina dentro de una etapa. Lo normal es realizar la asignación completa etapa por etapa, de esta manera, empezando por la primera podemos:

1. Asignar los trabajos a la primera máquina de la etapa de acuerdo a una regla de prioridad adecuada (FCFS, SPT, LPT, etc...).
2. Asignar los trabajos a la primera máquina que quede libre de acuerdo a una regla de prioridad.
3. Asignar cada trabajo a la primera máquina que quede libre siguiendo el orden de la secuenciación obtenido para los trabajos.

4. Asignar cada trabajo a la máquina que antes pueda terminar los trabajos, escogidos éstos de acuerdo al orden de la secuenciación obtenido.

De acuerdo con la revisión de la literatura realizada en la sección anterior, lo más frecuente ha sido la utilización del segundo método de asignación. El tercer y cuarto métodos de asignación son equivalentes si tenemos que todas las máquinas en la etapa son idénticas, pero el cuarto método de asignación proporcionará resultados diferentes en el caso de que las máquinas sean no relacionadas. Por ejemplo, podemos tener un trabajo j que se tiene que procesar en la etapa i , si las máquinas son idénticas, la primera que quede libre será al mismo tiempo la que primero pueda terminar el trabajo. Sin embargo, si las máquinas no son idénticas, se puede dar el siguiente caso: supongamos que la primera máquina que queda libre es la máquina l y tenemos que $p_{i,l,j}=100$, por poner un ejemplo. Puede existir otra máquina l' que queda libre después que l y puede ocurrir que $p_{i,l',j}=20$, es decir, la máquina l' es cinco veces más rápida procesando el trabajo j que la máquina l . Asignando el trabajo a la máquina l' podemos *terminar antes* el trabajo j incluso habiendo empezando más tarde. Recordemos que también tenemos tiempos de cambio de partida, y puede ocurrir que exista una máquina que quede libre mucho antes que las demás, pero que ésta haya terminado de procesar un trabajo concreto y se requiera un gran tiempo de cambio para procesar el siguiente trabajo que se pretende asignar. En este caso lo más sensato sería buscar una máquina que quedase libre después pero necesitase un menor tiempo de cambio. Este modo de asignación provocará que los trabajos sufran más esperas (al no asignarse a la primera máquina que quede libre) pero evidentemente resultarán en soluciones con un menor C_{max} .

Esta manera de realizar la asignación no ha sido considerada anteriormente. El motivo es, como hemos visto anteriormente, que la literatura se centra casi exclusivamente en los problemas con máquinas idénticas por lo que la asignación es más sencilla. Simulaciones iniciales demostraron que este modo de asignación es el que presenta mejores resultados que los otros tres modos considerados (considerando las reglas de prioridad FCFS, SPT y LPT) por lo que lo utilizaremos en el GA que se propone para el problema $FHm, ((RM^{(i)})_{i=1}^{(m)})/S_{sd}, M_j/C_{max}$.

Este modo de asignación se recoge en las funciones de cálculo del C_{max} . Que en el caso de este problema se complican un poco y quedan como se explica a continuación.

Tenemos una permutación o secuencia de n trabajos, que llamamos π . El trabajo que ocupa la posición k de la secuencia se denota por $\pi_{(k)}$. Existen m etapas. Al número total de máquinas en el problema lo llamamos M y es igual a $\sum_{i=1}^m m_i$. Llamamos L_{i_l} al último trabajo que se asignó a la máquina l dentro de la etapa i , $l \in \{1, \dots, m_i\}$. La manera de operar es la siguiente: recorremos ordenadamente la permutación π y para cada trabajo recorremos las m etapas buscando, en cada caso la máquina que antes pueda terminar el trabajo teniendo en cuenta las diferencias en los tiempos de proceso y los tiempos de cambio de partida.

$$\begin{aligned}
 Tpo_{min} &= \min_{\substack{l=1 \\ p_{i_l, \pi_{(j)}} \neq -1}} \left\{ \max \left\{ O_{i, L_{i_l}} f + S_{i_l, L_{i_l}, \pi_{(j)}}; O_{i-1, \pi_{(j)}} f \right\} + p_{i_l, \pi_{(j)}} \right\} \\
 asig &= arg(Tpo_{min}) \\
 O_{i, \pi_{(j)}} f &= Tpo_{min} \\
 O_{i, \pi_{(j)}} s &= Tpo_{min} - p_{i_{asig}, \pi_{(j)}} \\
 L_{i_{asig}} &= \pi_{(j)} \\
 i &= (1, \dots, m), j = (1, \dots, n)
 \end{aligned}$$

Donde $O_{0j}f = 0, j = (1, \dots, n)$ y $O_{i0}f = 0, i = (1, \dots, m)$. Como vemos, Tpo_{min} representa lo antes que puede terminar el trabajo, para lo cual se recorren todas las máquinas dentro de la etapa, con una importante salvedad, si nos encontramos que en una máquina de la etapa el $p_{i_l, \pi_{(j)}}$ es igual a -1 significará que esa máquina no puede procesar el trabajo $\pi_{(j)}$, por lo que tendrá que procesarse en otra. Esta es la manera en la que contemplaremos la restricción de uso de máquinas. Para cada máquina se busca el máximo de los dos datos siguientes: cuando queda liberada la máquina (instante de finalización del último trabajo que se asignó a esta máquina) más el tiempo de cambio si se quiere procesar el trabajo $\pi_{(j)}$ y cuando termina el trabajo $\pi_{(j)}$ en la etapa anterior. Este cálculo nos dará el instante de comienzo más temprano para el trabajo $\pi_{(j)}$, al que luego le sumamos el tiempo de proceso en la máquina l de la etapa, con lo que tenemos el tiempo

de finalización. *asig* es simplemente la máquina que proporciona este tiempo mínimo de finalización del trabajo $\pi_{(j)}$, es decir, la máquina a la que se asigna el trabajo en la etapa. A partir de estos cálculos anteriores ya podemos calcular el instante de comienzo y de finalización para el trabajo $\pi_{(j)}$. De igual manera actualizamos el último trabajo que se asigna a la máquina *asig* con el trabajo $\pi_{(j)}$. Es importante tener en cuenta que en un taller de flujo híbrido con máquinas no relacionadas es necesario definir las matrices de tiempos de proceso y de tiempos de cambio por máquinas, y no por etapas como se hace en el problema FSMP. De ahí que tengamos M matrices de tiempos de cambio de partida y no m , de igual manera la matriz de tiempos de proceso es de dimensiones $M \times n$ en vez de $m \times n$. Una vez calculados los instantes de comienzo y finalización para todas las tareas calculamos los tiempos de finalización para los trabajos con la siguiente expresión:

$$C_{\pi_{(j)}} = O_{m, \pi_{(j)}} f, \quad j = (1, \dots, n)$$

Es decir, el tiempo de finalización para cada trabajo equivale al instante de finalización de la tarea de ese mismo trabajo en la última etapa. El C_{max} se calculará como:

$$C_{max} = \max \left\{ C_{\pi_{(1)}}, C_{\pi_{(2)}}, \dots, C_{\pi_{(n)}} \right\}$$

Hay que tener en cuenta que en el caso del problema que nos ocupa, esta última expresión no se puede simplificar a $C_{max} = O_{m, \pi_{(n)}} f$, dado que la última tarea (en la etapa m) del último trabajo de la secuencia no tiene por qué ocupar la máquina que termina más tarde, puede ser que cualquier otro trabajo en la última etapa termine después que el último de la secuencia.

El funcionamiento del resto de la función de evaluación es idéntico a la de los algoritmos propuestos en capítulos anteriores, simplemente se mapea el valor del C_{max} calculado y se realizan los escalados cuando sea necesario dependiendo del tipo de selección.

6.5.4.2. Inicialización de la población

El capítulos anteriores presentamos una nueva inicialización de la población utilizando la heurística NEH, una modificación de esta heurística (NEH_m) e individuos generados aleatoriamente. Este tipo de inicialización funcionó muy bien en el caso del taller de flujo estándar y en la versión con tiempos de cambio dependientes de la secuencia que hacía uso de la heurística NEHT_RMB de Ríos-Mercado y Bard (1998b). Desgraciadamente, la heurística NEH sólo contempla el problema de la secuenciación y no funciona en el caso de máquinas paralelas y mucho menos con las mejoras de Taillard. Si recordamos, estas mejoras evitaban tener que evaluar el C_{max} de una secuencia parcial cuando se realizan las inserciones de los trabajos, lo cual acelera notablemente el proceso y reduce la complejidad computacional de la heurística.

Una propuesta para incorporar el problema de la asignación a la heurística NEH es utilizar la versión estándar de este método (sin las mejoras de Taillard) y cada vez que se evalúen todas las posibles posiciones de inserción en una iteración dada, utilizar el procedimiento de asignación visto en la sección anterior. De esta manera tenemos una nueva heurística que considera simultáneamente los problemas de secuenciación y de asignación, proporcionando como resultado una buena secuencia con la que inicializar la población. Llamaremos NEH_H a esta modificación de la heurística NEH que además contemplará la restricción de uso de máquinas y los tiempos de cambio de partida dependientes de la secuencia.

Un problema adicional es que, como ya vimos en capítulos anteriores, la heurística NEH original es muy costosa en términos computacionales, y más en este caso donde evaluar el C_{max} requiere el paso adicional de asignación. Por esto, realmente no es factible utilizar la versión modificada de la heurística (NEH_m) en la inicialización, ya que ésta costaría mucho de llevar a cabo.

La inicialización que proponemos se basa en generar un único individuo con la heurística NEH_H y el resto aleatorios. Para los individuos obtenidos al azar realizamos una secuenciación aleatoria y luego una asignación siguiendo el esquema visto en la sección anterior. Este tipo de inicialización, aunque inferior en prestaciones a las vistas en los algoritmos propuestos en capítulos anteriores, demostró ser notablemente superior a las inicializaciones totalmente aleatorias.

La inicialización se muestra en el Listado 6.1.

```
function Inicializacion(var P: Population);
begin
    //El primer individuo se genera por la heurística NEHH, una extensión
    //de la NEH original para tiempos de cambio de partida dependientes
    //de la secuencia, máquinas paralelas no relacionadas y restricción
    //de uso de máquinas
    Population[1]:=NEHH;
    //El resto de individuos hasta llenar la población se generan mediante
    //permutaciones obtenidas al azar y la asignación a las máquinas que
    //antes puedan terminar el trabajo en cada etapa
    for m := 2 to P.size do
        Population[m]:=RANDOM;
    end;
```

Listado 6.1 – Proceso de inicialización para el GA_H.

6.5.4.3. Otros parámetros y operadores del algoritmo genético

El operador de selección, cruce, mutación no necesitan modificaciones, dado que trabajan con la representación genética ordinal. El esquema generacional, como se ha comentado en el capítulo anterior, es aplicable a cualquier problema y sólo depende de la función de evaluación que ya se ha modificado convenientemente en la sección anterior.

El operador o procedimiento de reinicialización sí necesita cambios. Recordemos que este operador hace uso de la heurística NEH_m que se podría modificar para el problema que nos ocupa, pero a un alto coste computacional. Proponemos una simplificación de este operador visto en la Sección 4.3.4 del Capítulo 4 donde no se utiliza esta heurística. Concretamente, el operador queda como sigue:

1. En cada generación g , almacenar el mínimo C_{max} utilizando las funciones para el cálculo vistas en la Sección 6.5.4.1: C_{max_g} .
2. Si $C_{max_g} = C_{max_{g-1}}$ entonces incrementamos el contador de generaciones sin mejora en el C_{max} : $countmak = countmak + 1$. En cambio, si $C_{max_g} < C_{max_{g-1}}$, hacemos $countmak = 0$.
3. Si $countmak > G_r$ entonces aplicamos el procedimiento de reinicialización simplificado siguiente:

- Ordenar la población ascendente por el C_{max} de cada individuo.
- Saltar el primer 20 % de los individuos de la lista ordenada (los mejores individuos).
- Del 80 % de individuos restantes, el primer 50 % se reemplazan por simples mutaciones de desplazamiento (mutación SHIFT) de individuos extraídos del primer 20 % (mutaciones de los mejores) y el 50 % restante se reemplaza con individuos generados totalmente al azar.
- $countmak = 0$.

Como vemos, hemos eliminado la introducción de “buenos” individuos en la reinicialización, aunque mediante las mutaciones de los mejores individuos supliremos un poco esta carencia en el operador.

6.5.4.4. Calibración de los parámetros y operadores del algoritmo genético

En esta ocasión, debido a los cambios importantes que se han realizado en la función de evaluación, en la inicialización de la población y en el operador de reinicialización, es necesario calibrar el algoritmo genético GA_H propuesto. El conjunto de operadores y parámetros, así como los correspondientes niveles o variantes que se van a considerar en la parametrización se muestran a continuación:

- Tipo de operador de selección, 2 variantes: Torneo y Ranking.
- Operador de cruce, 9 variantes: OP, OX, PMX, SB2OX, SBOX, SJ2OX, SJOX, TP y UOB.
- Probabilidad de cruce (P_c), 6 niveles: 0, 0,1, 0,2, 0,3, 0,4 y 0,5.
- Probabilidad de mutación (P_m), 5 niveles: 0, 0,005, 0,01, 0,015 y 0,02.
- Tamaño de la población (P_{size}), 4 niveles: 20, 30, 40 y 50.
- Generaciones antes de reinicialización (G_r), 3 niveles: 25, 50 y 75.

Todos estos operadores y parámetros dan un total de $2 \cdot 9 \cdot 6 \cdot 5 \cdot 4 \cdot 3 = 6.480$ combinaciones posibles, o diferentes algoritmos genéticos. En este caso hemos

eliminado el operador GPX de la evaluación porque en experimentos iniciales proporcionaba unos resultados muy pobres. Hemos de tener en cuenta que este problema es costoso de resolver por lo complicado de la función de evaluación.

De nuevo, para poder evaluar los algoritmos necesitamos un conjunto de pruebas que contenga todos los aspectos que se resuelven en este problema: máquinas no relacionadas en cada etapa y tiempos de cambio dependientes de la secuencia. En Vallada, Ruiz y Maroto (2003) se propusieron 12 conjuntos de 110 problemas cada uno. Para estos problemas los tiempos de proceso y los tiempos de cambio se calculan de manera similar a los visto en los Capítulos 3 y 5. En el primer grupo de problemas tenemos talleres de flujo híbridos con un número de máquinas por etapa entre 1 y 3 ($U[1, 3]$) y a su vez existen 4 subgrupos con diferentes ratios entre los tiempos de proceso y los tiempos de cambio de partida. En el segundo grupo de problemas el número de máquinas por etapa está fijado a 2 y en el tercer grupo tenemos un total de 3 máquinas por etapa. De esta manera, el conjunto de datos SSD10_P13 se refiere a los 110 problemas donde tenemos talleres de flujo híbridos con entre 1 y 3 máquinas por etapa y los tiempos de cambio son el 10 % de los tiempos de proceso. De igual manera el conjunto SSD125_P3 indica los 110 problemas con 3 máquinas por etapa y donde los tiempos de cambio son el 125 % de los de proceso. Uniendo los 12 grupos de problemas tenemos un total de 1320 problemas diferentes. El número de etapas y el número de trabajos dentro de cada uno de los 12 grupos es idéntico a los problemas de Taillard, es decir, hay 11 tipos de problemas de entre 20 trabajos y 5 etapas hasta 200 trabajos y 20 etapas. Notar que hemos eliminado los problemas de 500 trabajos y 20 etapas por ser problemas de un tamaño demasiado grande. Para hacernos una idea, un ejemplo de 500 trabajos y 20 etapas, con 3 máquinas por etapa tendría una matriz de 500×60 con los tiempos de proceso y 60 matrices de 500×500 con los tiempos de cambio dependientes de la secuencia lo que en total son 15.030.000 datos. Los problemas de este tipo simplemente eran demasiado grandes y los ficheros de datos resultaban difíciles de manejar. En esta fase inicial también hemos preferido no incluir restricciones de uso de máquinas, esto realmente no supone un problema dado que realmente al quitar estas restricciones estamos complicando más el problema dado que existen más posibilidades de asignación de los trabajos dentro

de las máquinas. Más adelante estudiaremos un grupo de ejemplos reales con este tipo de restricciones.

El formato de los ficheros de entrada cambia ligeramente para poder considerar todas las restricciones, en la primera línea aparecen tres números, el primero representa el número de trabajos (n), el segundo el número de máquinas totales o M y el tercero el número de etapas o m . En la segunda línea tenemos un total de m datos, que representan los distintos m_i , es decir, el número de máquinas no relacionadas que tenemos por etapa. Tras estas dos líneas aparece una matriz con los tiempos de proceso que tiene tamaño nxM , es decir, se especifican todos los tiempos de proceso para todas las máquinas, no por etapas. Al terminar esta matriz aparecen M matrices de tamaño nxn , que son las matrices de los tiempos de cambio de partida para cada una de las M máquinas. Un ejemplo puede ser el problema ta007_SSD10_P13, de 20 trabajos y 5 etapas, que se muestra en el Listado 6.2.

20 8 5
1 2 2 2 1
0 91 1 81 2 82 3 78 4 98 5 40 6 2 7 12
0 27 1 77 2 98 3 3 4 39 5 74 6 87 7 40
0 34 1 69 2 97 3 69 4 75 5 94 6 68 7 95
0 42 1 52 2 12 3 99 4 33 5 9 6 28 7 80
0 3 1 28 2 35 3 41 4 8 5 62 6 84 7 6
0 11 1 28 2 84 3 73 4 86 5 86 6 30 7 92
0 54 1 77 2 70 3 28 4 41 5 13 6 35 7 14
0 27 1 42 2 27 3 99 4 41 5 78 6 42 7 83
0 30 1 53 2 37 3 13 4 22 5 46 6 39 7 49
0 9 1 46 2 59 3 59 4 43 5 83 6 85 7 36
0 15 1 49 2 42 3 47 4 34 5 52 6 34 7 38
0 88 1 15 2 57 3 8 4 80 5 13 6 36 7 43
0 55 1 43 2 16 3 92 4 16 5 70 6 9 7 89
0 50 1 65 2 11 3 87 4 37 5 40 6 96 7 94
0 57 1 41 2 34 3 62 4 94 5 60 6 84 7 33
0 28 1 77 2 1 3 45 4 34 5 5 6 86 7 28
0 4 1 36 2 59 3 73 4 20 5 48 6 35 7 39
0 43 1 57 2 95 3 59 4 57 5 80 6 5 7 55
0 93 1 15 2 49 3 63 4 47 5 43 6 93 7 21
0 1 1 81 2 90 3 54 4 62 5 34 6 74 7 25
SSD
M0
0 2 6 6 5 1 9 3 4 4 1 1 1 5 3 3 1 2 8 5 5
6 0 3 1 4 9 1 8 7 5 3 7 4 5 5 2 4
6 4 0 7 4 6 2 8 8 9 9 2 4 8 7 3 4 6 4 6
2 1 4 0 1 6 9 5 9 8 1 7 9 4 7 3 9 2 6 5
1 9 8 6 0 5 7 6 6 5 9 4 7 3 1 8 4 4 2 4
4 8 2 4 9 0 4 2 6 5 6 4 7 9 1 6 8 2 8 5
8 5 2 7 4 6 0 1 5 8 4 4 1 8 7 3 8 3 4 4
4 8 4 4 1 9 8 0 8 4 1 9 7 2 4 9 8 1 9 2
8 5 4 4 4 9 9 3 0 3 4 5 2 3 8 6 4 9 2 3
1 6 9 2 7 3 8 8 9 0 1 9 5 5 7 2 6 8 5 9
1 5 4 4 6 9 8 4 6 2 0 4 8 7 5 3 6 9 7 8
2 8 5 8 9 7 7 6 2 8 3 0 6 8 1 4 6 9 2 1
7 6 8 6 6 9 9 3 8 3 5 9 0 6 7 3 5 1 6 4
6 6 9 4 4 1 1 7 8 9 6 9 2 0 1 3 2 2 4 1
8 5 4 5 1 7 1 2 5 8 8 9 3 6 0 8 9 9 9 6
1 8 6 9 4 5 7 9 6 5 8 5 1 9 2 0 9 4 1 5
5 5 7 5 3 6 2 4 5 7 1 1 8 2 3 3 0 1 2 4
4 8 6 3 7 5 6 2 1 6 7 5 7 2 2 8 2 0 4 3
3 3 8 2 8 9 7 1 8 8 1 5 7 7 5 1 2 3 0 1
9 8 4 1 1 5 1 4 4 4 6 3 1 8 4 8 9 9 3 0

M1	0	6	9	5	2	2	1	3	1	1	6	3	1	1	5	3	9	3	1	6	
0	8	4	2	7	1	7	1	1	4	4	4	3	8	7	8	4	5	8	5		
2	6	0	4	3	8	8	7	6	9	2	5	4	3	5	6	8	4	9	4		
2	2	7	0	2	8	2	7	1	6	1	1	3	4	4	7	8	7	2	2		
2	5	1	2	0	5	5	7	2	5	7	4	4	8	7	7	7	1	2	7		
5	1	9	4	5	0	7	6	6	3	1	3	9	4	2	8	2	3	3	2		
4	8	3	4	3	5	0	9	8	4	9	2	7	8	4	5	7	9	2	2		
1	7	1	4	6	5	8	0	3	6	8	8	4	5	2	2	1	9	8	3		
1	3	6	7	2	4	1	5	0	2	2	9	6	2	3	4	2	2	5	7		
3	2	8	5	5	6	3	6	5	0	6	2	7	2	1	7	7	8	5	7		
2	6	5	7	5	7	4	5	3	3	0	7	6	8	5	7	8	9	6	2		
4	1	2	5	9	1	6	1	8	4	3	0	2	5	4	7	3	6	2	1		
2	2	2	9	6	7	2	9	9	5	5	5	0	6	3	3	5	4	9	9		
4	3	4	3	3	2	7	1	8	3	9	7	6	0	7	1	4	5	8	4		
6	6	3	5	8	8	5	4	5	6	7	2	3	7	0	7	2	9	7	3		
8	5	2	1	8	9	1	9	2	9	3	2	6	6	8	0	4	1	4	9		
2	1	3	4	3	9	9	8	4	9	6	8	3	3	6	3	0	4	4	4		
1	9	7	2	2	1	8	7	6	4	8	2	3	9	2	6	4	0	7	8		
9	3	2	3	3	7	8	9	9	3	4	3	1	8	3	6	6	5	0	3		
7	5	4	5	7	2	5	1	1	4	8	1	1	2	5	1	6	1	7	0		
M2	0	3	3	1	3	1	5	5	6	3	8	9	1	2	1	4	3	6	2	1	
1	0	8	7	8	2	1	6	1	4	2	7	8	8	5	1	9	6	7	3		
3	3	0	4	9	5	3	4	2	1	8	5	6	2	4	4	1	6	2	7		
6	4	3	0	1	5	7	9	2	9	7	8	1	6	3	5	5	3	7	8		
5	8	5	4	0	5	1	5	1	3	4	3	1	6	7	4	8	1	7	8		
1	2	1	3	4	0	2	4	6	3	3	8	9	2	1	1	9	2	9	2		
2	9	5	6	3	7	0	9	4	8	7	5	7	3	4	2	6	5	4	7		
1	8	1	4	3	6	2	0	8	8	9	6	9	8	8	1	9	2	7	5		
8	4	8	5	4	4	4	7	2	0	7	5	9	6	6	4	8	7	9	9		
7	6	5	2	7	9	2	9	2	0	6	2	3	1	6	6	1	3	5	5		
1	5	9	4	3	9	8	7	1	3	0	6	8	7	3	4	4	1	6	6		
4	8	4	4	1	7	1	6	5	4	5	0	1	4	4	5	9	6	7	4		
5	7	8	9	4	3	4	1	5	6	5	5	0	4	1	8	6	9	8	8		
3	6	9	1	2	9	4	9	5	9	7	9	4	0	7	1	2	2	5	1		
2	4	1	1	3	6	6	4	9	9	6	2	1	2	0	3	4	3	3	5		
6	2	7	8	3	5	1	8	6	8	6	1	1	7	9	0	2	5	1	2		
6	6	2	2	1	6	3	5	8	7	3	5	7	8	2	4	0	9	3	3		
4	2	7	5	1	6	1	4	2	3	5	4	1	3	6	2	2	7	0	7		
3	3	7	2	1	9	6	9	5	1	9	8	7	6	4	7	7	4	0	3		
5	5	7	8	8	9	6	3	8	6	9	7	2	7	1	3	6	5	3	0		
M3	0	1	8	5	5	7	3	9	3	4	3	4	6	4	9	2	3	8	5	5	
2	0	5	1	6	9	5	5	1	9	3	2	4	4	7	6	3	1	5	9		
4	3	0	9	9	9	2	7	3	6	9	2	5	8	3	4	2	3	4	7		
3	7	4	0	8	1	1	6	6	8	4	2	1	6	4	1	9	3	9	9		
8	6	5	2	0	7	4	3	1	2	3	6	6	8	4	1	8	9	2			
2	9	5	7	7	0	8	7	9	7	7	6	9	5	6	6	3	4	7	3		
5	6	8	3	7	7	0	6	1	8	8	7	2	8	2	1	5	3	9			
4	4	8	3	8	9	9	0	5	9	1	2	7	3	6	4	6	9	8	2		
3	2	7	9	7	3	9	9	0	1	4	1	7	6	9	8	1	9	6	1		
3	6	1	1	5	6	7	2	9	0	8	5	1	9	1	4	5	5	1	2		
2	4	4	3	1	9	1	9	6	7	0	3	2	9	5	9	7	7	5	4		
8	3	9	6	1	5	2	4	3	7	1	0	3	9	5	3	1	2	7	6		
3	1	8	2	8	3	4	5	9	7	4	9	0	2	5	9	1	5	8	2		
8	8	4	8	4	5	8	3	7	6	3	2	3	0	4	4	1	1	6	7		
3	7	7	1	8	6	9	5	1	8	8	5	0	6	2	9	1	7				
7	2	3	9	6	4	2	7	6	1	2	7	1	4	5	0	6	9	9	5		
4	4	8	1	6	4	7	6	1	2	7	2	4	8	7	9	6	0	7	1	4	
2	3	8	3	6	5	9	7	3	1	7	5	8	8	7	7	5	0	9	9		
6	8	2	8	5	8	3	1	4	9	6	1	4	2	8	9	1	9	0	5		
5	8	5	9	1	4	3	8	7	6	5	9	6	2	5	8	2	1	2	0		
M4	0	9	1	8	4	1	7	2	7	2	7	8	3	5	2	8	1	3	4	6	
6	0	1	1	4	4	3	7	8	6	3	9	6	8	5	3	7	9	3	9		
9	6	0	6	1	2	4	5	4	8	3	8	1	1	1	6	9	5	7	6		
9	1	2	0	1	9	4	5	6	7	2	6	7	6	9	7	1	4	4	3		
9	4	3	8	0	1	1	6	8	3	7	6	6	6	4	5	7	9	4	8		
5	6	1	6	4	0	5	8	8	1	3	4	3	3	9	1	1	5	1	7		
2	3	4	1	8	7	0	7	3	8	3	9	7	1	3	4	9	5	3	2		
6	7	5	5	5	2	2	0	6	9	1	1	3	6	4	4	6	5	6	6		
6	1	1	2	6	4	5	5	0	8	3	1	2	7	4	7	7	2	8	7		
4	2	4	9	7	3	8	9	3	0	4	2	4	5	9	2	9	4	7	4		
8	5	4	7	9	9	1	9	4	7	0	7	9	4	3	8	1	6	5	8		
5	1	5	4	9	4	8	3	6	8	3	0	4	1	6	1	9	1	4	6		
3	9	8	9	7	7	4	4	4	7	6	0	7	3	8	4	9	8	9	9		
3	1	8	6	4	4	7	9	6	1	4	5	1	0	4	6	3	3	3	1		
7	3	2	9	3	1	9	1	6	9	4	6	2	1	0	8	8	4	4	8		
7	3	2	9	8	6	7	1	3	7	1	7	7	8	3	0	5	1	9	7		
9	5	1	4	8	6	6	1	9	1	3	4	9	5	9	5	0	1	1	1		
5	7	6	8	2	8	6	2	3	9	3	9	5	8	8	3	7	0	5	7		
3	6	9	5	1	4	7	2	1	9	9	4	7	7	3	3	5	9	0	8		
3	9	9	6	5	1	4	4	5	9	3	4	8	5	6	7	4	6	1	0		

M5	0	3	9	7	9	2	4	4	6	6	2	1	4	9	8	3	2	8	2	4
	6	0	8	3	6	1	2	5	1	8	2	4	6	3	2	8	8	4	4	9
	1	9	0	2	9	6	5	3	4	3	3	5	7	3	5	4	6	4	5	3
	3	7	5	0	9	2	6	8	9	9	5	8	1	9	3	3	3	2	2	3
	4	8	6	1	0	4	8	6	1	1	9	6	8	1	4	6	5	9	4	1
	8	5	2	4	3	0	2	7	2	8	4	7	2	8	4	3	7	2	9	9
	6	8	6	1	6	0	4	1	2	2	6	7	5	6	8	7	7	7	2	
	1	1	9	6	2	9	6	0	8	3	4	5	7	3	3	7	2	2	2	6
	1	6	7	7	6	8	9	7	0	1	2	7	2	3	9	2	7	5	1	8
	6	9	6	9	6	1	2	6	5	0	1	7	7	2	1	4	4	6	2	3
	7	4	9	1	7	3	2	6	3	6	0	8	5	7	8	6	4	1	6	1
	5	4	3	4	4	9	1	9	5	7	8	0	7	6	1	6	1	2	6	4
	9	8	6	7	5	9	4	5	2	7	3	4	0	1	1	7	7	5	3	4
	1	1	1	2	7	6	3	7	4	9	8	2	3	0	1	3	4	2	4	9
	3	7	6	2	9	3	7	4	2	7	9	1	1	5	0	0	3	1	5	8
	6	7	3	8	8	8	4	2	4	7	6	1	7	1	0	0	9	3	1	5
	3	2	3	5	9	4	5	5	1	3	3	2	7	3	4	8	0	7	3	2
	1	2	2	6	1	9	9	6	3	1	9	1	9	1	3	2	9	0	0	8
	7	2	2	5	7	4	5	6	3	6	5	8	3	6	2	7	1	0	3	
	8	5	3	8	9	6	9	7	7	6	5	2	5	8	3	9	5	1	2	0
M6	0	1	6	3	5	4	3	8	4	8	4	1	3	3	5	6	9	6	4	6
	8	0	5	7	5	6	1	7	5	7	2	7	9	6	6	2	2	8	4	2
	4	5	0	3	7	4	5	4	3	6	6	8	2	2	7	3	4	9	5	8
	3	8	4	0	9	3	9	7	1	1	8	4	2	3	7	2	1	4	5	8
	5	6	3	1	0	1	7	3	7	7	1	1	5	6	8	2	1	1	5	9
	9	1	2	3	7	0	9	2	7	6	9	8	6	4	8	4	6	8	8	8
	7	3	4	4	4	4	0	1	6	6	6	7	6	2	9	3	3	5	9	5
	4	6	8	5	8	8	5	0	7	2	8	2	6	1	8	8	1	2	7	5
	2	1	4	8	7	8	9	5	0	1	5	1	3	2	2	2	1	2	3	1
	1	3	7	1	7	2	6	8	9	0	6	9	5	7	2	6	4	1	8	8
	9	2	7	2	2	8	1	8	9	5	0	6	8	9	9	1	3	5	5	5
	3	6	3	1	2	5	3	3	1	9	9	0	2	3	5	6	5	8	7	9
	4	6	6	8	8	2	9	6	2	1	7	8	0	6	3	9	8	8	1	8
	8	2	2	3	5	7	5	9	2	1	8	2	6	0	3	1	3	8	5	5
	6	8	5	7	2	6	7	7	3	6	6	7	9	9	0	4	9	1	6	7
	3	3	7	8	5	8	4	3	6	7	1	2	3	2	6	0	4	6	3	8
	6	1	5	8	9	6	9	3	7	8	2	6	6	7	9	2	0	7	5	1
	4	9	7	9	7	8	5	2	2	2	2	9	2	1	3	2	2	9	0	4
	1	2	2	1	1	7	3	1	5	4	9	5	8	2	4	4	3	7	0	3
	5	5	4	4	3	8	6	6	6	2	3	5	5	2	2	2	6	9	9	0
M7	0	4	3	2	7	6	8	1	2	6	1	2	9	7	3	5	3	7	8	5
	7	0	7	2	5	8	3	8	7	9	6	6	5	7	3	9	4	7	6	
	6	3	0	4	6	8	8	2	2	6	9	8	6	1	5	1	8	7	8	4
	5	1	5	0	8	3	1	7	3	5	5	3	5	5	9	2	8	5	3	1
	8	1	8	4	0	4	1	2	2	5	4	1	6	4	2	6	2	6	9	5
	9	3	2	6	7	0	3	1	3	9	1	6	6	3	9	2	1	1	9	1
	3	2	3	7	4	3	0	1	5	6	8	2	4	5	8	3	8	9	8	8
	1	6	5	6	8	4	3	0	3	1	2	1	1	5	4	7	7	2	2	9
	2	3	3	3	3	5	2	9	0	2	8	6	7	3	6	1	4	4	1	3
	1	9	5	8	8	8	5	1	0	9	1	2	6	4	5	3	6	7	4	
	2	6	7	6	4	5	1	7	2	7	0	5	5	5	4	8	6	5	4	6
	3	8	6	1	2	5	1	1	2	6	3	0	9	8	7	6	6	6	3	8
	9	4	5	8	2	3	5	5	1	6	8	6	0	8	1	2	4	8	7	6
	7	6	1	2	2	9	9	5	4	9	8	3	6	0	4	5	7	5	4	4
	9	1	1	6	8	2	9	7	5	2	9	5	9	2	0	3	2	6	2	9
	2	6	4	7	4	3	2	3	3	3	8	5	3	6	2	0	4	8	9	3
	4	9	4	8	8	4	5	7	5	7	1	4	8	8	2	6	0	5	4	7
	9	3	5	7	9	2	2	8	6	7	4	3	6	9	6	1	1	0	5	1
	7	2	5	5	4	6	7	3	8	5	1	2	1	9	9	8	1	5	0	7
	9	5	7	7	7	6	3	3	9	8	5	5	1	4	5	7	3	5	0	0

Listado 6.2 – Problema ta007_SSD10_P13.

El resto de problemas de prueba se puede descargar de <http://www.upv.es/gio>. Para estas pruebas tampoco conocemos el óptimo. Para poder realizar la calibración del algoritmo GA_H necesitamos unos valores de referencia, que se han obtenido para los 1320 problemas de ejemplo tras ejecutar el algoritmo GA_H con el criterio de parada fijado a la evaluación de 500.000 C_{max} y con los parámetros

y operadores obtenidos en el experimento del capítulo anterior, que si recordamos resultaron ser prácticamente idénticos a los obtenidos en el experimento realizado en el Capítulo 4. Las mejores soluciones obtenidas se detallan en la Sección B.2 del Anexo B.

Realizaremos un total de 12 experimentos distintos, uno para cada grupo de problemas. Dada la gran cantidad de experimentos a realizar y el coste computacional de evaluar 6.480 algoritmos diferentes en cada uno, vamos a limitar el número de ejemplos que se resuelven en cada experimento. Primero, de los 110 problemas de ejemplo vamos a considerar solo los problemas de máximo 100 trabajos y 20 etapas (hasta el problema 90) y de estos 90 problemas solo los pares, es decir, desde el ta002 hasta el ta090, lo que nos proporciona un total de 45 ejemplos a resolver para cada algoritmo en cada experimento. La variable respuesta es el *IPSOM* total sobre estos 45 ejemplos dentro de cada grupo de problemas. Por criterios de eficiencia, el criterio de parada se fija en 8.000 evaluaciones del C_{max} . Con estos datos, para cada experimento se han resuelto $6.480 \cdot 45 = 291.600$ problemas, y $219.600 \cdot 8.000 = 2.332.800.000$ secuencias. En general, uniendo los cuatro experimentos se han resuelto $219.600 \cdot 12 = 3.499.200$ problemas y $27.993.600.000$ secuencias.

De nuevo, la metodología para la obtención de los resultados es idéntica a la utilizada en los Capítulos 4 y 5, donde se utilizó un cluster de cuatro ordenadores para realizar las ejecuciones. Asimismo, se mantiene la codificación de los factores.

Vamos a analizar el experimento realizado con el primer grupo de problemas SSD10_P13. De nuevo, con 6.480 datos para cada experimento, donde cada dato es el *IPSOM* total obtenido a través del promedio de los resultados de 45 problemas diferentes, se cumplen las hipótesis de normalidad, homocedasticidad e independencia del residuo en los 12 experimentos realizados. Por motivos de espacio no se incluyen las gráficas que verifican estas hipótesis (para realizar la verificación se necesitan nueve gráficas, que para los 12 experimentos darían un total de 108 gráficas). A continuación se muestra la tabla del ANOVA que se muestra en la Tabla 6.3.

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	6,1659	5	1,23318	329,50	0,0000
B:Cross_Type	10,1335	8	1,26668	338,46	0,0000
C:Mut_Prob	212,74	4	53,1849	14210,93	0,0000
D:Pop_Size	14,4977	3	4,83255	1291,25	0,0000
E:Restart	0,17486	2	0,0874301	23,36	0,0000
F:Select_Type	87,5079	1	87,5079	23381,97	0,0000
INTERACTIONS					
AB	2,62436	40	0,0656091	17,53	0,0000
AC	6,29856	20	0,314928	84,15	0,0000
AD	0,124187	15	0,00827915	2,21	0,0045
AE	0,668155	10	0,0668155	17,85	0,0000
AF	0,429334	5	0,0858667	22,94	0,0000
BC	4,71308	32	0,147284	39,35	0,0000
BD	0,176023	24	0,00733428	1,96	0,0034
BE	0,0948603	16	0,00592877	1,58	0,0643
BF	0,970691	8	0,121336	32,42	0,0000
CD	0,625437	12	0,0521197	13,93	0,0000
CE	6,31868	8	0,789835	211,04	0,0000
CF	8,29387	4	2,07347	554,03	0,0000
DE	0,254867	6	0,0424778	11,35	0,0000
DF	7,99629	3	2,66543	712,20	0,0000
EF	0,240764	2	0,120382	32,17	0,0000
RESIDUAL	23,3946	6251	0,00374254		
TOTAL (CORRECTED)	394,443	6479			

All F-ratios are based on the residual mean square error.

Tabla 6.3 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD10_P13.

Seguiremos el procedimiento usual para analizar los resultados utilizando los valores del “F-Ratio”. Según la tabla, el factor más importante es el tipo de selección (Select_Type), en la Figura 6.29 se muestra el correspondiente gráfico de medias.

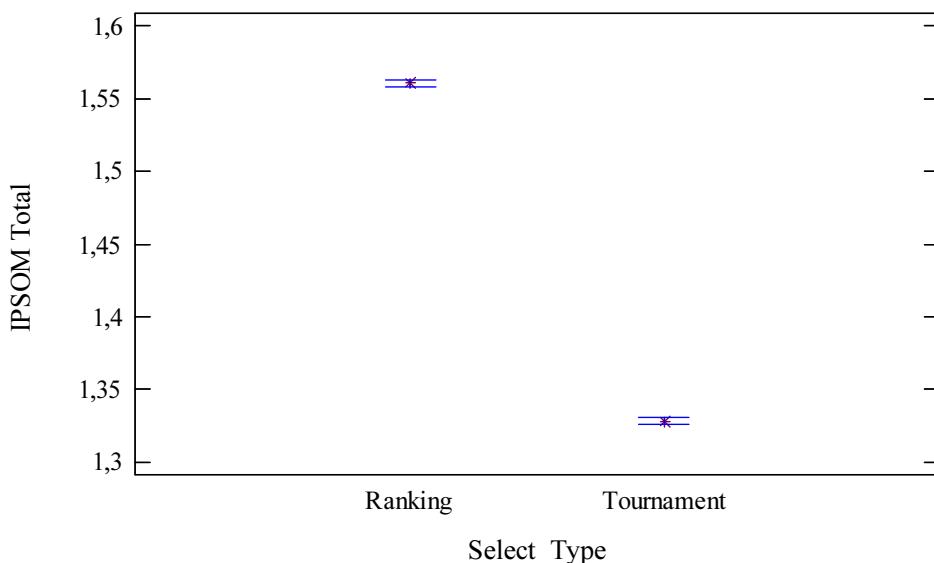


Figura 6.29 – Gráfico de medias e intervalos LSD al 95 % para el factor tipo de selección (Select_Type), experimento SSD10_P13.

Observamos como se repite el patrón de todos los experimentos realizados hasta el momento: la selección por torneo binario es superior a la selección por ranking. El siguiente factor por orden de “*F-Ratio*” es la probabilidad de mutación (Mut_Prob), que se muestra en la Figura 6.30.

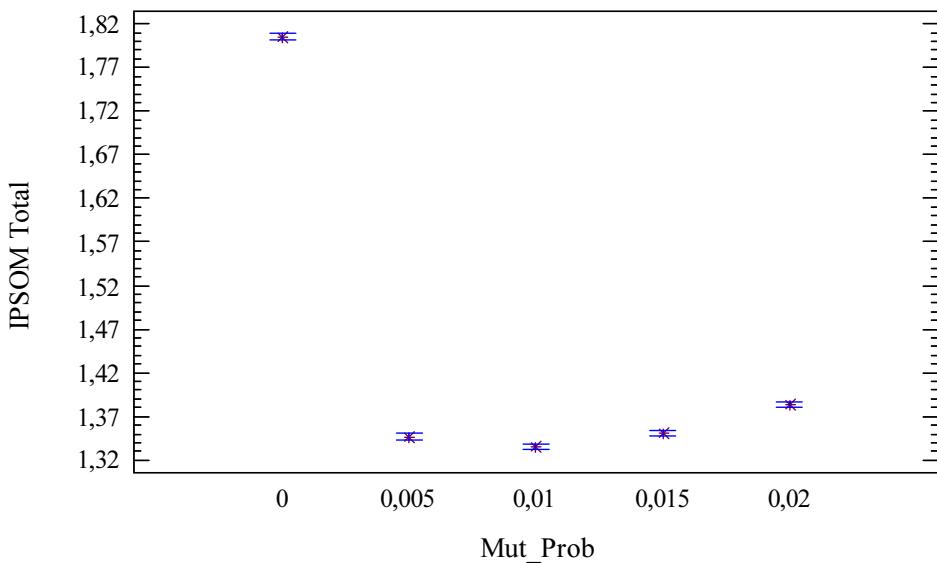


Figura 6.30 – Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de mutación (Mut_Prob), experimento SSD10_P13.

En este caso podemos observar como una ausencia de mutación perjudica seriamente al algoritmo. Según la gráfica, una mutación de 0,01 parece ser lo más adecuado. El siguiente factor es el tamaño de la población (Pop_Size).

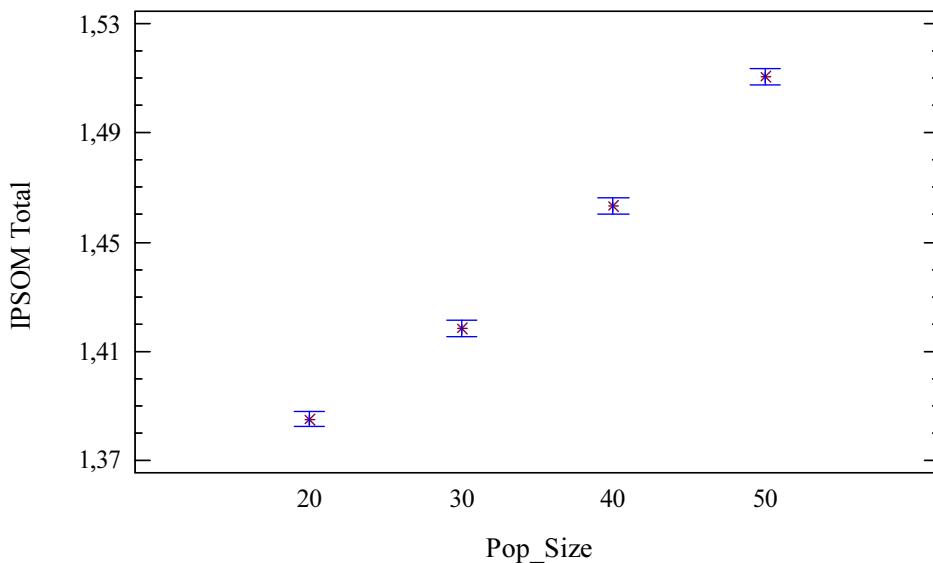


Figura 6.31 – Gráfico de medias e intervalos LSD al 95 % para el factor tamaño de la población (Pop_Size), experimento SSD10_P13.

En este caso una población de 20 parece funcionar mejor. Tamaños de población menores serían probablemente mejores a juzgar por el perfil de la gráfica, pero esto tiene el efecto de aumentar el número de generaciones y por tanto el “overhead” (trabajo realizado por el algoritmo no directamente encaminado al objetivo, como ordenaciones, reestructuración de datos, etc...) del algoritmo, resultando en una menor eficiencia. Debido a esto preferimos elegir el tamaño de la población o P_{size} de 20. La interacción entre los factores Pop_Size y Select_Type es el siguiente elemento con mayor “F-Ratio” de la tabla. El gráfico de medias se muestra en la Figura 6.32.

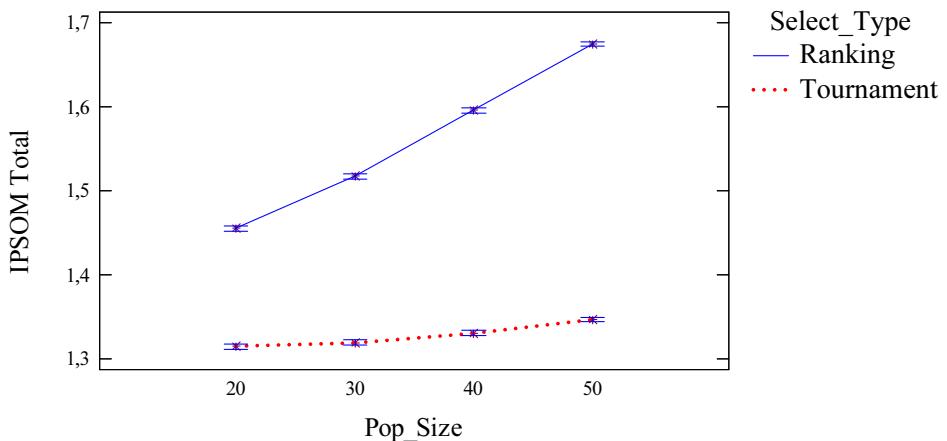


Figura 6.32 – Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores tamaño de la población (Pop_Size) y tipo de selección (Select_Type), experimento SSD10_P13.

Vemos la existencia de una fuerte interacción, mientras que el efecto de cambiar la población no resulta muy importante en la selección por torneo, en la selección por ranking sí resulta determinante. El gráfico nos confirma que la selección por torneo resulta muy interesante. Seguidamente se muestra la interacción entre los factores probabilidad de mutación (Mut_Prob) y tipo de selección (Select_Type).

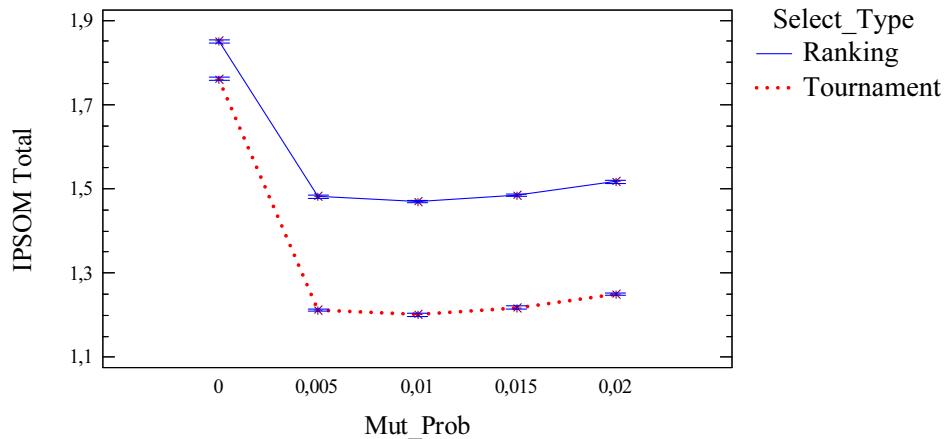


Figura 6.33 – Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores probabilidad de mutación (Mut_Prob) y tipo de selección (Select_Type), experimento SSD10_P13.

Esta gráfica vuelve a confirmar que la selección por torneo es muy determinante. Otro factor de importancia es Cross_Type, que se muestra en la Figura 6.34.

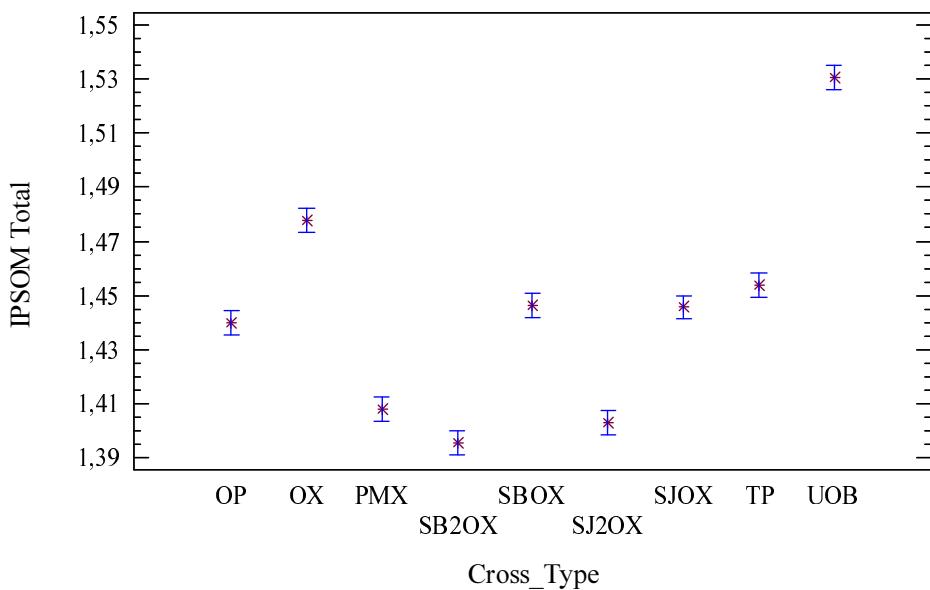


Figura 6.34 – Gráfico de medias e intervalos LSD al 95 % para el factor tipo de cruce (*Cross_Type*), experimento SSD10_P13.

El cruce SB2OX sigue presentando un mejor comportamiento frente a las otras ocho alternativas, aunque en este caso el cruce PMX sí ha resultado ser también bastante bueno, aunque no tanto como el SB2OX. El siguiente factor a considerar es la probabilidad de cruce (*Cross_Prob*).

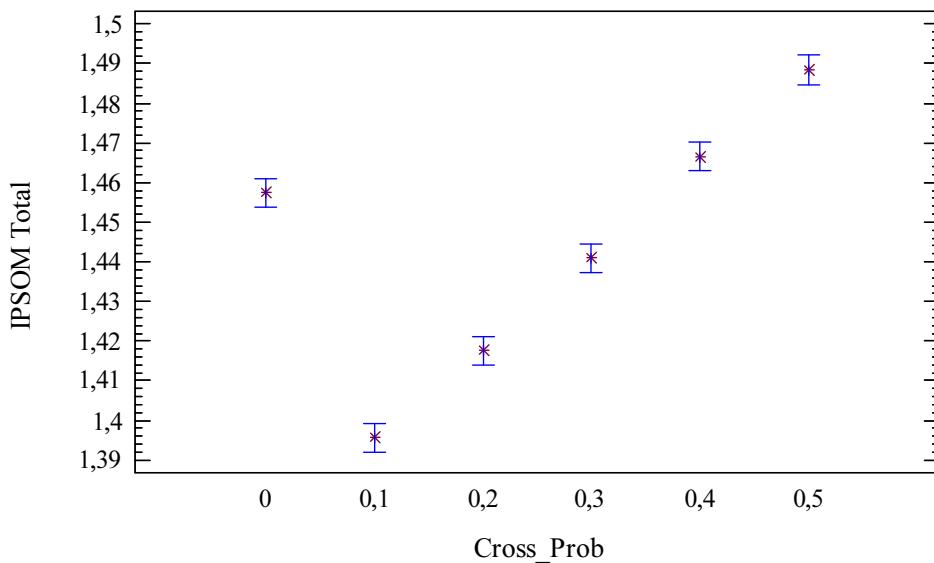


Figura 6.35 – Gráfico de medias e intervalos LSD al 95 % para el factor probabilidad de cruce (Cross_Prob), experimento SSD10_P13.

Como ya ocurriera en todos los experimentos anteriores, una probabilidad de cruce baja de 0,1 parece ser lo más adecuado. También podemos observar como una probabilidad nula empeora el algoritmo. El último factor que queda por fijar es el `Restart`, el primer elemento de la tabla del ANOVA por orden de importancia que contiene este factor es la interacción entre `Mut_Prob` y `Restart` que se muestra en la Figura 6.36.

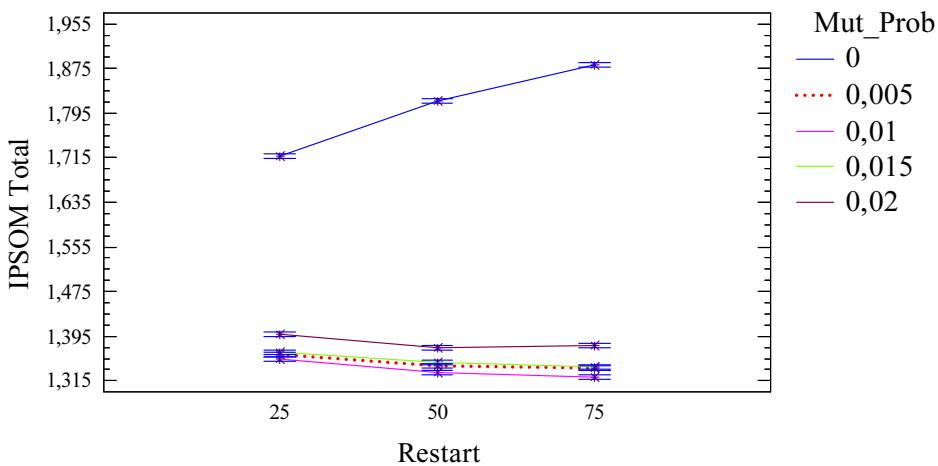


Figura 6.36 – Gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores probabilidad de mutación (Mut_Prob) y operador de reinicialización (Restart), experimento SSD10_P13.

Como vemos, se ve cómo una probabilidad de mutación de 0 es muy poco recomendable. Si excluimos esta probabilidad de mutación de la gráfica podremos observar con más detalle el comportamiento de la interacción (Figura 6.37).

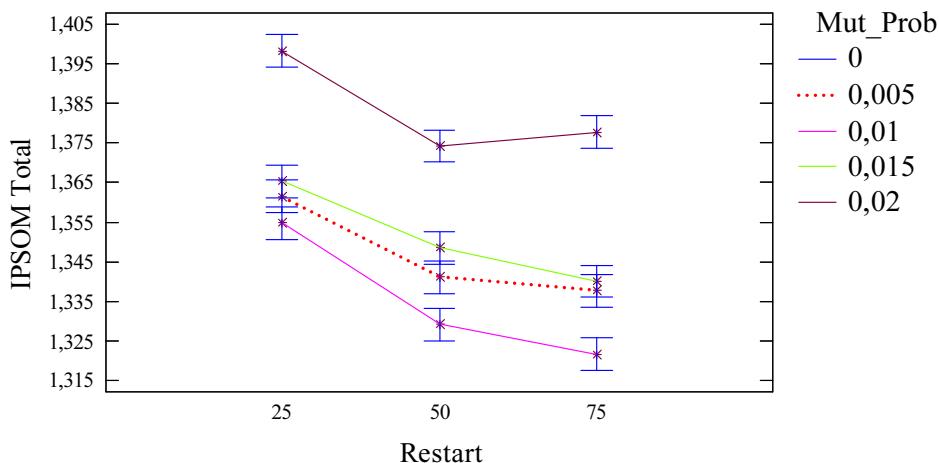


Figura 6.37 – Detalle del gráfico de medias e intervalos LSD al 95 % para la interacción entre los factores probabilidad de mutación (Mut_Prob) y operador de reinicialización (Restart), experimento SSD10_P13.

Ahora podemos observar mejor como la probabilidad de mutación de 0,01 resulta ser la más favorable para el algoritmo, algo que ya sabíamos de los anteriores gráficos. A partir de esta gráfica no hay diferencias estadísticamente significativas entre un valor de *Restart* de 50 y 75. Análisis de otras interacciones de menor importancia resuelven este “empate” a favor de un valor de *Restart* de 50.

Los resultados de la calibración para los otros 11 experimentos realizados se encuentran en la Sección A.3 del Anexo A. El resultado de los experimentos se resume en la Tabla 6.4 donde se recoge la parametrización del mejor algoritmo para cada experimento.

Experimento	Operadores y parámetros					
	Select	Restart	Mut_Prob	Cross_Type	Cross_Prob	Pop_Size
SSD10_P13	Tournament	50	0,01	SB2OX	0,1	20
SSD50_P13	Tournament	50	0,01	SB2OX	0,1	20
SSD100_P13	Tournament	75	0,01	SB2OX	0,1	20
SSD125_P13	Tournament	50	0,01	SB2OX	0,1	20
SSD10_P2	Tournament	50	0,01	SB2OX	0,1	50
SSD50_P2	Tournament	75	0,01	SB2OX	0,1	50
SSD100_P2	Tournament	50	0,01	SB2OX	0,2	50
SSD125_P2	Tournament	50	0,01	SB2OX	0,2	50
SSD10_P3	Tournament	25	0,01	SB2OX	0,1	50
SSD50_P3	Tournament	50	0,01	SB2OX	0,1	50
SSD100_P3	Tournament	50	0,01	SB2OX	0,2	50
SSD125_P3	Tournament	50	0,01	SB2OX	0,2	50

Select=Tipo de selección, Restart=Operador de reinicialización, Mut_Prob=Probabilidad de mutación, Cross_Type=Tipo de cruce, Cross_Prob=Probabilidad de cruce, Pop_Size=Tamaño de la población, Tournament=Selección por torneo binario, SB2OX=Operador de cruce “Similar Block 2-Point Order Crossover”

Tabla 6.4 – Resultado de los 12 experimentos y de la calibración del algoritmo GA_H propuesto.

Los algoritmos resultantes de los 12 experimentos se parecen mucho entre sí, concretamente, el operador de selección, cruce y la probabilidad de mutación permanecen constantes, y los dos primeros coinciden con lo obtenido en los experimentos de capítulos anteriores. El operador de reinicialización (Restart) resultó ser poco significativo en todos los experimentos realizados, de ahí que se hayan dado sus tres niveles, aunque el que más se repite es el 50. La probabilidad de cruce también varía en los resultados, aunque el valor de 0,1 es el que más se da, que de nuevo coincide con los algoritmos genéticos propuestos y estudiados anteriormente. Por último, el tamaño de la población aparece con el valor 20 consistentemente en los cuatro primeros experimentos y con el valor 50 en los ocho restantes. Este patrón tiene una clara explicación. En muchos de los problemas de los grupos SSD10_P13, SSD50_P13, SSD100_P13 y SSD125_P13 podemos encontrar una o más etapas con una sola máquina mientras que el resto de etapas tienen dos o tres máquinas. Esta configuración es muy dada a producir cuellos de botella importantes que serán precisamente las etapas con una sola máquina.

En esta situación, no tiene demasiada importancia la asignación de los trabajos en las etapas que no son cuellos de botella y realmente no existe el problema de asignación en las etapas con una sola máquina. El algoritmo GA_H no necesita un tamaño de población grande al preferir trabajar con menos soluciones y poder realizar mutaciones y búsquedas dentro de un población reducida, ya que con menos individuos se podrán realizar más generaciones y por tanto hacer una búsqueda más intensiva.

Los mejores resultados los obtendríamos utilizando 12 algoritmos diferentes, cada uno adaptado a los 12 conjuntos de pruebas. Nosotros preferimos una aproximación distinta. Aún habiendo parametrizado cada algoritmo, vamos a utilizar un único conjunto de operadores y parámetros, dado que consideramos que los algoritmos deben ser robustos frente a los datos de entrada y no tendría sentido utilizar unos u otros operadores dependiendo de los casos. A partir de la tabla anterior utilizaremos aquellos operadores y parámetros que más se repiten, es decir, la moda de los niveles y/o variantes para los 12 experimentos, por lo que la parametrización final queda como sigue:

- Tipo de operador de selección: Torneo.
- Operador de cruce: SB2OX.
- Probabilidad de cruce (P_c): 0,1.
- Probabilidad de mutación (P_m): 0,01.
- Tamaño de la población (P_{size}): 50.
- Generaciones antes de reinicialización (G_r): 50.

6.6. Evaluación de técnicas para la programación de la producción en el sector cerámico

Como vimos en la Sección 6.5.3, realmente no se han propuesto métodos que de manera general nos permitan resolver los 12 grupos de problemas que hemos generado. No obstante, es posible hacer adaptaciones de los métodos que

se han propuesto en la literatura y que se han utilizado en capítulos anteriores. Proponemos la adaptación de varios métodos heurísticos y metaheurísticos para el problema $FHm, ((RM^{(i)})_{i=1}^{(m)})/S_{sd}/C_{max}$.

La adaptación de los métodos la realizaremos sustituyendo el cálculo del C_{max} estándar por el que se ha propuesto en este capítulo. El primer método que se puede modificar es la heurística original NEH de Nawaz, Enscore y Ham (1983). El cambio estriba en realizar la asignación y el cálculo del C_{max} para el problema ampliado en vez de calcular el C_{max} como en el taller de flujo estándar. Llamaremos NEH_H a la adaptación de la heurística NEH para el problema que nos ocupa. El método simulated annealing de Osman y Potts (1989) también es fácil de cambiar puesto que tan solo hay que sustituir el cálculo del C_{max} , a este método lo llamaremos $SAOP_H$. El algoritmo genético de Reeves (1995) también es fácilmente adaptable, puesto que podemos modificar la función de evaluación tal y como se ha realizado para el algoritmo GA_H , así como la inicialización, que se realizará por la heurística NEH_H en vez de utilizar la heurística NEH estándar. Nos referiremos a este método modificado como $GAReev_H$. El algoritmo basado en búsqueda tabú de Widmer y Hertz (1989) también se ha adaptado. En este caso se ha modificado la inicialización, que se realiza por la heurística NEH adaptada a este problema (NEH_H), de igual manera, la evaluación de la solución en cada paso del algoritmo se hace con las funciones de cálculo del C_{max} ampliadas. Llamaremos a la adaptación de este algoritmo $Spirit_H$. De manera similar se han modificado los algoritmos genéticos de Chen, Vempati y Aljaber (1995), Murata, Ishibuchi y Tanaka (1996a) y Ponnambalam, Aravindan y Chandrasekaran (2001) a los que nos referiremos como $GACHen_H$, $GAMIT_H$ y $GAPAC_H$ respectivamente, para los que simplemente se han cambiado las funciones de evaluación de los individuos.

También hemos modificado el método RANDOM de manera que para cada individuo generado al azar se realiza la asignación a máquinas siguiendo el mismo esquema que los métodos anteriores. En total tenemos nueve algoritmos distintos que proponemos para resolver el problema $FHm, ((RM^{(i)})_{i=1}^{(m)})/S_{sd}/C_{max}$, ocho de ellos son adaptaciones de métodos anteriormente comentados y el noveno es el algoritmo propuesto, GA_H .

Vamos a realizar una comparativa de estos nueve algoritmos para ver cual es que proporciona mejores resultados. Para las pruebas se ha utilizado el mismo ordenador AMD Athlon 1600+XP (1400 MHz) con 512 Mbytes de memoria RAM que en capítulos anteriores. Fijamos el criterio de parada a la evaluación de 5.000 C_{max} . Hemos de tener en cuenta que este es el problema más difícil de los que se han considerado hasta el momento, de ahí que se permitan un número tan relativamente bajo de evaluaciones del C_{max} . Todos los métodos propuestos, a excepción de la heurística NEH_H y la búsqueda tabú $Spirit_H$ son estocásticos y se han realizado cinco ejecuciones independientes, para después calcular la media de los resultados. La variable respuesta que queremos minimizar es el *IPSUM* total. Utilizaremos los 12 conjuntos de 110 pruebas que se han presentado en este capítulo. Realizaremos el análisis de los resultados por separado para cada conjunto de pruebas.

Los resultados para el conjunto SSD10_P13 se muestran en la Tabla 6.5.

Problema	RANDOM	GA_H	$SAOP_H$	$Spirit_H$	$GAReev_H$	NEH_H	$GACHen_H$	$GAPAC_H$	$GAMIT_H$
20x5	3,86	0,06	0,70	1,60	0,63	1,78	2,14	5,45	1,94
20x10	6,07	0,16	1,16	3,19	0,66	2,36	3,05	8,99	3,07
20x20	7,53	0,22	1,42	3,77	0,80	3,63	3,90	10,45	3,12
50x5	5,67	0,11	0,71	1,35	0,43	1,71	2,66	7,08	2,74
50x10	9,21	0,22	2,12	3,11	0,80	2,80	3,74	11,11	5,63
50x20	10,96	0,33	1,99	3,93	0,96	3,39	4,97	12,99	6,78
100x5	8,36	0,50	2,29	5,55	0,68	1,97	4,85	10,36	8,57
100x10	8,42	0,36	2,08	5,91	0,62	1,76	4,83	10,25	10,39
100x20	9,54	0,37	1,18	6,18	0,53	2,25	4,85	11,01	11,30
200x10	7,08	0,90	0,82	8,43	0,69	1,39	4,79	7,97	4,21
200x20	8,72	1,04	1,30	10,35	0,84	1,78	5,05	9,98	7,03
Media	7,77	0,39	1,43	4,85	0,70	2,26	4,07	9,60	5,89

RANDOM=Regla aleatoria, GA_H =GA propuesto, $SAOP_H$ =Simulated annealing de Osman y Potts (1989) (adaptación), $Spirit_H$ =Tabu search de Widmer y Hertz (1989) (adaptación), $GAReev_H$ =GA de Reeves (1995) (adaptación), NEH_H =Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), $GACHen_H$ =GA de Chen, Vempati y Aljaber (1995) (adaptación), $GAPAC_H$ =GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), $GAMIT_H$ =GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.5 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P13.

Lo primero que se puede observar es que, como ya pudimos observar en el Capítulo 3, el método $GAPAC_H$ resulta ser peor que la simple regla RANDOM para los 11 grupos de pruebas. Veremos además como este resultado se repite para todos los conjuntos de problemas. En una mejor situación están los otros dos algoritmos genéticos adaptados, los métodos $GAMIT_H$ y $GAChen_H$, aunque para el primero el $IPSON$ pasa del 10 % en algunos tipos de pruebas. El método $Spirit_H$ se ha adaptado bastante bien al nuevo problema, aunque como veremos, es sensible al número de máquinas por etapa. La heurística NEH_H también se adapta de forma favorable al nuevo problema de programación de la producción, con algo más del 2 % de $IPSON$ total. Los tres mejores métodos para este conjunto de pruebas resultan ser el GA_H , $GAReev_H$ y $SAOP_H$, por este orden. Concretamente, el método GA_H es un 79,49 % mejor que el $GAReev_H$, aunque para los grupos de problemas más grandes parece que el $GAReev_H$ domina, llegando a ser un 30,43 % mejor para los problemas de 200 trabajos y 10 etapas. Este comportamiento se reproduce para los cuatro grupos de problemas de tipo P13, es decir, donde es fácil que se produzcan cuellos de botella importantes en las secuencias de producción. Vamos a analizar el resultado de los métodos conforme aumentamos la magnitud de los tiempos de cambio de partida (grupos de problemas SSD50_P13, SSD100_P13 y SSD125_P13). Los resultados se muestran de la Tabla 6.6 a la Tabla 6.8.

Problema	RANDOM	GA_H	$SAOP_H$	$Spirit_H$	$GAReev_H$	NEH_H	$GAChen_H$	$GAPAC_H$	$GAMIT_H$
20x5	14,54	0,99	4,22	7,09	2,64	5,77	8,37	20,52	7,81
20x10	12,40	0,16	3,42	6,39	2,32	6,13	6,25	16,69	6,41
20x20	10,90	0,63	3,43	6,05	2,62	6,17	5,53	14,48	6,79
50x5	20,82	0,71	3,32	6,04	2,09	4,80	10,49	24,22	12,02
50x10	18,20	1,01	3,92	6,50	2,67	5,05	8,59	21,21	10,89
50x20	16,01	0,66	3,66	5,79	2,12	4,50	7,65	18,84	8,67
100x5	23,24	1,48	4,47	16,75	1,54	3,78	11,86	26,56	23,83
100x10	20,85	1,27	3,90	14,98	1,86	3,44	10,69	23,69	21,09
100x20	19,26	1,68	3,16	13,74	2,16	4,07	10,27	21,66	19,41
200x10	24,42	2,55	3,70	27,46	2,16	3,03	14,98	26,02	15,28
200x20	18,61	2,18	3,30	20,88	1,91	2,65	10,45	20,36	12,97
Media	18,11	1,21	3,68	11,97	2,19	4,49	9,56	21,29	13,20

RANDOM=Regla aleatoria, GA_H =GA propuesto, $SAOP_H$ =Simulated annealing de Osman y Potts (1989) (adaptación), $Spirit_H$ =Tabu search de Widmer y Hertz (1989) (adaptación), $GAReev_H$ =GA de Reeves (1995) (adaptación), NEH_H =Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), $GAChen_H$ =GA de Chen, Vempati y Aljaber (1995) (adaptación), $GAPAC_H$ =GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), $GAMIT_H$ =GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.6 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD50_P13.

Problema	RANDOM	GA_H	SAOP_H	Spirit_H	GAReev_H	NEH_H	GACHen_H	GAPAC_H	GAMIT_H
20x5	22,58	1,25	6,59	11,56	5,44	9,86	14,58	34,46	11,72
20x10	17,07	1,13	7,04	10,21	4,40	8,54	10,35	24,46	10,06
20x20	13,88	1,19	4,88	8,10	3,83	8,28	8,26	18,31	7,94
50x5	34,83	1,35	6,15	9,47	4,12	7,58	18,87	41,12	18,26
50x10	25,00	0,55	5,40	7,75	3,00	7,23	12,45	30,12	12,84
50x20	19,87	0,57	4,58	6,85	2,79	5,81	10,40	24,39	14,16
100x5	37,88	1,87	6,19	27,21	2,26	4,67	21,51	43,34	31,75
100x10	31,25	2,01	5,35	23,03	2,50	4,70	17,54	35,51	26,20
100x20	26,50	2,07	4,19	19,19	2,47	4,83	15,23	29,86	26,15
200x10	40,47	3,22	6,55	46,17	2,74	3,67	26,61	43,74	28,32
200x20	27,76	2,97	5,64	30,77	2,55	3,40	17,10	30,03	17,99
Media	27,01	1,65	5,69	18,21	3,28	6,23	15,72	32,30	18,67

RANDOM=Regla aleatoria, GA_H=GA propuesto, SAOP_H=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit_H=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev_H=GA de Reeves (1995) (adaptación), NEH_H=Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), GACHen_H=GA de Chen, Vempati y Aljaber (1995) (adaptación), GAPAC_H=GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), GAMIT_H=GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.7 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD100_P13.

Problema	RANDOM	GA_H	$SAOP_H$	$Spirit_H$	$GARev_H$	NEH_H	$GACHen_H$	$GAPAC_H$	$GAMIT_H$
20x5	27,08	1,80	8,17	13,93	6,10	11,52	15,95	37,79	14,55
20x10	19,37	1,56	7,86	10,82	4,85	9,80	11,77	27,70	11,39
20x20	14,36	0,64	6,25	8,38	3,61	7,51	8,63	20,29	9,32
50x5	39,15	1,36	6,86	10,78	3,75	9,15	21,02	46,97	18,20
50x10	27,66	0,14	5,28	8,50	3,61	7,57	14,94	33,38	15,47
50x20	22,38	0,64	5,49	8,25	3,44	6,91	11,97	26,34	14,09
100x5	43,65	2,01	7,22	31,51	2,62	5,45	24,57	49,77	32,29
100x10	34,97	2,14	5,54	25,36	2,68	5,23	20,48	40,04	33,44
100x20	29,16	1,91	4,20	21,36	2,54	4,78	16,56	32,77	25,75
200x10	46,67	3,32	7,42	52,02	2,81	3,81	30,14	50,15	26,47
200x20	31,73	2,90	6,89	34,57	2,57	3,46	20,12	34,66	22,52
Media	30,56	1,68	6,47	20,50	3,51	6,83	17,83	36,35	20,32

RANDOM=Regla aleatoria, GA_H =GA propuesto, $SAOP_H$ =Simulated annealing de Osman y Potts (1989) (adaptación), $Spirit_H$ =Tabu search de Widmer y Hertz (1989) (adaptación), $GARev_H$ =GA de Reeves (1995) (adaptación), NEH_H =Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), $GACHen_H$ =GA de Chen, Vempati y Aljaber (1995) (adaptación), $GAPAC_H$ =GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), $GAMIT_H$ =GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.8 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD125_P13.

El comportamiento de los algoritmos se ve afectado por la magnitud de los tiempos de cambio, tal y como viéramos en el Capítulo 5. Concretamente el método $SAOP_H$ empeora más del 450 % pasando del grupo de problemas SSD10_P13 al grupo SSD125_P13, el resto de métodos se ven afectados de una forma similar. Podemos ver que en los cuatro grupos de problemas el método $GARev_H$ domina al GA_H en los ejemplos de mayor tamaño, pero éste último aumenta su ventaja sobre el primero conforme aumentan los tiempos de cambio. Concretamente, el método GA_H es un 80,99 % mejor que el método $GARev_H$ en el grupo SSD50_P13 y un 98,79 % y 107,14 % mejor para los grupos SSD100_P13 y SSD125_P13 respectivamente. La explicación de este comportamiento es que es posible que el método GA_H no sea tan capaz de escapar de óptimos locales como el método $GARev_H$, ya que no dispone del esquema de mutación adaptativo. Recordemos que los óptimos locales son muy fuertes cuando existen etapas que representan cuellos de botella. Vamos a analizar ahora los grupos de problemas de dos máquinas no relacionadas por cada etapa, los grupos del SSD10_P2, al SSD125_P2, que se muestran de la Tabla 6.9 a la Tabla 6.12.

Problema	RANDOM	GA_H	SAOP_H	Spirit_H	GAReev_H	NEH_H	GAChen_H	GAPAC_H	GAMIT_H
20x5	13,46	2,64	8,64	12,54	3,77	9,94	7,55	19,38	10,10
20x10	11,83	1,76	5,99	9,52	4,12	7,67	6,57	16,32	9,50
20x20	9,22	1,23	4,51	6,73	3,12	5,74	5,29	12,49	6,98
50x5	13,37	1,43	8,63	7,86	1,90	4,61	4,94	18,20	10,61
50x10	13,97	1,35	8,46	9,01	2,58	4,61	5,42	17,85	11,87
50x20	12,40	0,35	7,24	8,03	1,85	3,25	4,22	15,64	9,74
100x5	11,60	0,95	7,73	7,65	1,35	2,64	4,54	14,71	13,56
100x10	11,64	0,94	7,79	8,45	1,55	2,62	2,84	15,32	13,41
100x20	11,37	0,79	7,48	8,62	1,52	2,61	3,01	13,95	13,23
200x10	8,19	0,65	6,75	10,56	0,63	0,99	2,15	10,59	7,64
200x20	8,61	0,59	6,41	10,88	0,61	0,84	2,15	10,80	7,83
Media	11,42	1,15	7,24	9,08	2,09	4,14	4,43	15,02	10,41

RANDOM=Regla aleatoria, GA_H=GA propuesto, SAOP_H=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit_H=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev_H=GA de Reeves (1995) (adaptación), NEH_H=Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), GAChen_H=GA de Chen, Vempati y Aljaber (1995) (adaptación), GAPAC_H=GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), GAMIT_H=GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.9 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P2.

Problema	RANDOM	GA_H	$SAOP_H$	$Spirit_H$	$GAReev_H$	NEH_H	$GAChen_H$	$GAPAC_H$	$GAMIT_H$
20x5	14,35	3,10	11,70	12,79	6,33	13,15	9,08	21,03	12,52
20x10	11,45	1,38	8,00	10,78	4,55	9,30	6,57	17,43	9,38
20x20	8,84	1,13	5,70	7,07	2,73	5,91	4,99	12,41	7,08
50x5	14,08	1,74	10,35	9,66	3,64	6,33	4,77	19,41	11,06
50x10	12,35	1,76	8,52	8,94	3,04	5,46	4,47	16,92	11,30
50x20	9,95	1,15	6,50	7,08	2,77	4,40	3,11	12,99	8,01
100x5	12,07	0,82	8,02	8,05	1,52	2,54	5,43	15,82	13,62
100x10	10,17	0,98	7,01	7,73	1,52	2,46	2,99	13,40	12,74
100x20	8,62	0,53	5,90	6,54	1,37	2,14	1,82	11,07	10,12
200x10	7,62	0,77	5,90	9,14	0,81	1,09	3,63	9,96	7,28
200x20	6,19	0,57	4,79	7,20	0,63	0,94	2,17	8,24	5,71
Media	10,52	1,27	7,49	8,63	2,63	4,88	4,46	14,43	9,89

RANDOM=Regla aleatoria, GA_H =GA propuesto, $SAOP_H$ =Simulated annealing de Osman y Potts (1989) (adaptación), $Spirit_H$ =Tabu search de Widmer y Hertz (1989) (adaptación), $GAReev_H$ =GA de Reeves (1995) (adaptación), NEH_H =Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), $GAChen_H$ =GA de Chen, Vempati y Aljaber (1995) (adaptación), $GAPAC_H$ =GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), $GAMIT_H$ =GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.10 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD50_P2.

Problema	RANDOM	GA_H	SAOP_H	Spirit_H	GAReev_H	NEH_H	GAChen_H	GAPAC_H	GAMIT_H
20x5	15,58	4,09	13,85	15,31	7,59	14,72	11,12	25,52	15,86
20x10	11,71	2,40	9,81	11,25	5,66	9,42	6,93	18,60	10,02
20x20	9,35	1,64	7,27	8,07	4,09	7,48	5,30	12,85	7,84
50x5	14,96	2,15	10,49	9,62	4,16	7,82	5,94	21,67	13,07
50x10	11,94	1,35	8,44	9,53	3,98	5,77	4,65	17,21	10,25
50x20	9,76	1,10	6,70	7,05	3,34	5,29	3,53	12,84	7,99
100x5	12,67	1,14	8,35	8,97	1,91	3,06	5,51	17,40	14,94
100x10	9,58	0,76	6,29	7,60	1,99	3,18	4,01	13,13	10,19
100x20	8,32	0,74	5,70	6,62	1,85	2,61	2,95	10,86	9,59
200x10	8,19	1,01	6,38	9,00	1,25	1,56	5,76	11,01	8,34
200x20	6,22	0,97	4,98	6,67	1,18	1,49	4,07	8,53	6,13
Media	10,75	1,58	8,02	9,06	3,36	5,67	5,43	15,42	10,38

RANDOM=Regla aleatoria, GA_H=GA propuesto, SAOP_H=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit_H=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev_H=GA de Reeves (1995) (adaptación), NEH_H=Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), GAChen_H=GA de Chen, Vempati y Aljaber (1995) (adaptación), GAPAC_H=GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), GAMIT_H=GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.11 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD100_P2.

Problema	RANDOM	GA _H	SAOP _H	Spirit _H	GAReev _H	NEH _H	GACHen _H	GAPAC _H	GAMIT _H
20x5	16,60	3,96	15,09	16,54	9,07	17,45	12,24	26,12	16,24
20x10	12,28	2,07	11,23	13,07	5,88	10,03	8,33	18,58	11,21
20x20	8,85	1,46	6,61	8,55	4,18	6,88	5,58	12,98	7,54
50x5	15,39	3,13	10,75	10,96	5,38	7,84	6,72	22,48	12,93
50x10	11,98	1,32	9,40	9,25	4,78	7,07	4,85	17,49	10,65
50x20	9,35	1,16	6,63	7,03	3,14	5,64	3,93	13,04	8,35
100x5	13,58	1,06	8,99	9,67	2,11	3,42	6,29	18,56	14,59
100x10	10,04	0,58	6,51	7,64	1,61	2,43	4,44	13,98	10,71
100x20	7,24	0,67	5,04	5,72	1,86	2,58	2,73	10,14	9,04
200x10	8,59	0,97	6,53	9,42	1,22	1,61	6,42	11,63	7,96
200x20	6,14	0,84	4,85	6,95	1,10	1,45	4,33	8,56	5,81
Media	10,91	1,56	8,33	9,53	3,67	6,04	5,99	15,78	10,46

RANDOM=Regla aleatoria, GA_H=GA propuesto, SAOP_H=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit_H=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev_H=GA de Reeves (1995) (adaptación), NEH_H=Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), GACHen_H=GA de Chen, Vempati y Aljaber (1995) (adaptación), GAPAC_H=GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), GAMIT_H=GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.12 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD125_P2.

Tal y como habíamos adelantado, en este grupo de problemas el método GA_H domina, ahora ya sí en todos los tamaños de problema, al método GAReev_H. La ventaja del primer método sobre el segundo es del 81,74 %, 107,87 %, 112,66 % y del 135,26 % para los cuatro grupos de pruebas. Luego no solo el método GA_H es mejor, sino que aumenta su ventaja conforme aumentan los tiempos de cambio. De forma general, el disponer de dos máquinas no relacionadas por etapa complica el problema, ya que todos los métodos obtienen un IPSOM total superior a los obtenidos en los grupos de problemas donde hay entre una y tres máquinas por etapa. Otro dato a tener en cuenta es que el método SAOP_H, que en el anterior grupo de problemas había resultado ser el tercer mejor algoritmo, ahora está incluso por debajo de la heurística NEH_H y del GA GACHen_H. Otro método que ha empeorado mucho con este tipo de problemas es el Spirit_H, ya que, como podemos observar, en el grupo de problemas SSD125_P3 obtiene un IPSOM total del 9,53 %, no mucho mejor que el método RANDOM. Los resultados para el tercer grupo de problemas con tres máquinas no relacionadas por etapa; los grupos del SSD10_P3 al SSD125_P3 se muestran de la Tabla 6.13 a la Tabla 6.16.

Problema	RANDOM	GA_H	$SAOP_H$	$Spirit_H$	$GAReev_H$	NEH_H	$GAChen_H$	$GAPAC_H$	$GAMIT_H$
20x5	13,01	3,06	9,53	13,33	4,01	11,28	8,40	22,11	12,31
20x10	11,74	2,34	6,39	11,79	3,25	6,81	7,11	16,75	10,09
20x20	8,82	1,03	4,51	7,71	2,33	4,84	5,56	12,48	6,84
50x5	14,03	2,33	10,42	10,89	3,05	7,16	4,21	19,70	12,22
50x10	14,22	1,96	12,22	11,67	2,62	5,23	4,81	18,50	11,83
50x20	13,14	0,87	8,82	11,04	2,43	3,85	5,17	16,68	10,76
100x5	11,48	1,48	10,18	10,19	2,06	3,23	4,59	15,84	14,97
100x10	11,30	0,54	9,46	10,14	0,97	2,32	2,54	15,17	15,55
100x20	10,93	0,41	9,15	10,16	0,97	1,97	2,72	14,09	13,18
200x10	9,65	0,60	9,38	12,51	0,62	0,99	4,27	12,21	10,09
200x20	7,37	0,32	6,86	10,05	0,57	0,97	0,95	9,55	7,89
Media	11,43	1,36	8,81	10,86	2,08	4,42	4,58	15,74	11,43

RANDOM=Regla aleatoria, GA_H =GA propuesto, $SAOP_H$ =Simulated annealing de Osman y Potts (1989) (adaptación), $Spirit_H$ =Tabu search de Widmer y Hertz (1989) (adaptación), $GAReev_H$ =GA de Reeves (1995) (adaptación), NEH_H =Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), $GAChen_H$ =GA de Chen, Vempati y Aljaber (1995) (adaptación), $GAPAC_H$ =GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), $GAMIT_H$ =GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.13 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P3.

Problema	RANDOM	GA_H	$SAOP_H$	$Spirit_H$	$GAReev_H$	NEH_H	$GAChen_H$	$GAPAC_H$	$GAMIT_H$
20x5	13,89	4,22	11,35	15,49	6,73	15,16	10,27	23,81	13,55
20x10	12,42	2,86	9,90	12,70	5,18	10,67	8,26	18,28	11,53
20x20	9,63	1,13	7,51	8,87	3,39	5,27	6,21	13,45	7,61
50x5	14,06	2,50	11,33	10,94	3,35	7,34	5,56	20,16	13,83
50x10	12,00	2,09	11,07	10,30	3,72	6,48	4,23	17,01	11,05
50x20	10,23	1,11	8,41	9,27	2,72	4,93	3,36	13,85	8,73
100x5	10,51	1,17	8,40	7,96	1,91	3,42	6,84	15,20	13,73
100x10	8,59	0,88	7,06	7,34	1,87	2,86	3,27	12,19	10,13
100x20	7,69	0,45	6,57	7,24	1,06	1,70	1,95	10,58	8,83
200x10	6,72	0,63	5,76	8,22	0,69	1,06	4,27	9,56	6,32
200x20	5,48	0,55	4,96	6,82	0,95	1,32	2,61	7,64	5,80
Media	10,11	1,60	8,39	9,56	2,87	5,47	5,16	14,70	10,10

RANDOM=Regla aleatoria, GA_H =GA propuesto, $SAOP_H$ =Simulated annealing de Osman y Potts (1989) (adaptación), $Spirit_H$ =Tabu search de Widmer y Hertz (1989) (adaptación), $GAReev_H$ =GA de Reeves (1995) (adaptación), NEH_H =Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), $GAChen_H$ =GA de Chen, Vempati y Aljaber (1995) (adaptación), $GAPAC_H$ =GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), $GAMIT_H$ =GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.14 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD50_P3.

Problema	RANDOM	GA_H	SAOP_H	Spirit_H	GAReev_H	NEH_H	GAChen_H	GAPAC_H	GAMIT_H
20x5	15,62	5,89	15,63	17,11	8,60	17,39	11,33	26,27	15,57
20x10	12,05	2,20	10,54	12,46	5,38	12,29	8,34	18,92	12,03
20x20	10,09	1,81	9,48	9,39	4,62	8,34	6,44	15,21	8,62
50x5	13,56	3,15	10,83	11,41	4,55	8,91	6,41	21,22	14,79
50x10	12,14	2,91	11,85	11,14	4,52	6,29	5,47	17,30	11,35
50x20	9,15	1,32	7,66	8,13	3,24	6,08	2,92	12,77	8,58
100x5	11,17	1,02	9,05	8,78	2,05	3,10	10,68	16,09	13,60
100x10	7,88	0,73	6,01	6,04	1,44	2,52	5,55	11,48	10,05
100x20	6,18	0,76	4,86	5,66	1,44	2,68	3,24	9,38	6,83
200x10	6,61	0,58	5,34	7,44	0,68	1,02	6,60	9,67	6,92
200x20	4,71	0,51	4,21	5,45	1,04	1,48	3,98	7,21	4,67
Media	9,92	1,90	8,68	9,36	3,42	6,37	6,45	15,05	10,27

RANDOM=Regla aleatoria, GA_H=GA propuesto, SAOP_H=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit_H=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev_H=GA de Reeves (1995) (adaptación), NEH_H=Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), GAChen_H=GA de Chen, Vempati y Aljaber (1995) (adaptación), GAPAC_H=GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), GAMIT_H=GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.15 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD100_P3.

Problema	RANDOM	GA _H	SAOP _H	Spirit _H	GARev _H	NEH _H	GAChen _H	GAPAC _H	GAMIT _H
20x5	16,58	5,69	16,40	18,86	8,97	19,47	12,01	27,88	17,12
20x10	12,30	3,67	12,01	13,70	7,99	15,72	8,53	20,72	13,93
20x20	9,31	1,65	8,53	9,77	4,60	7,60	6,34	14,49	9,11
50x5	15,34	3,66	12,51	12,86	5,25	9,07	8,75	22,60	15,80
50x10	12,36	3,66	11,30	11,12	5,53	9,03	5,80	18,15	11,62
50x20	8,73	1,42	7,29	8,34	3,34	5,35	2,89	12,62	8,55
100x5	11,34	0,92	8,90	8,62	2,02	3,59	11,21	17,02	13,81
100x10	8,55	0,73	6,89	6,70	1,96	2,98	7,40	12,69	10,46
100x20	5,77	0,30	4,37	5,27	1,52	2,42	3,84	8,86	6,23
200x10	5,49	0,60	4,31	6,33	0,79	1,39	6,19	8,83	5,85
200x20	4,92	0,73	4,08	5,57	1,30	1,71	4,92	7,31	5,14
Media	10,06	2,09	8,78	9,74	3,93	7,12	7,08	15,56	10,69

RANDOM=Regla aleatoria, GA_H=GA propuesto, SAOP_H=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit_H=Tabu search de Widmer y Hertz (1989) (adaptación), GARev_H=GA de Reeves (1995) (adaptación), NEH_H=Algoritmo de Nawaz, Enscore y Ham (1983) (adaptación), GAChen_H=GA de Chen, Vempati y Aljaber (1995) (adaptación), GAPAC_H=GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), GAMIT_H=GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.16 – IPSOM de los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD125_P3.

De nuevo, el algoritmo GA_H desarrollado supera a todos los demás. Más precisamente, la diferencia con respecto al segundo mejor método (GARev_H) es del 52,84 %, 79,36 %, 80 % y 88,04 % respectivamente para los cuatro conjuntos de pruebas con distintas magnitudes de los tiempos de proceso. Un último apunte puede hacerse con respecto a los demás métodos. Claramente, en el grupo de pruebas más “difícil”, que es el grupo SSD125_P3, ningún método a excepción de los dos anteriormente comentados, ha sido capaz de obtener IPSOM totales por debajo del 7 %. Esto quiere decir que el método GA_H es un 338,76 % mejor que el tercer mejor algoritmo (GAChen_H). La explicación puede ser que el resto de métodos se basan en características del problema del taller de flujo o similares y no son fácilmente adaptables al problema de producción en las empresas cerámicas. También resulta interesante comentar que, para los conjuntos de pruebas parece que las instancias más “grandes” sean más fáciles, dado que por ejemplo, en el caso del método GA_H, el IPSOM para los ejemplos de 200x20 es de 0,73 % mientras que para los ejemplos de 20x5 es de 5,69 %. No es que los ejemplos grandes sean más fáciles, sino que la solución de referencia de la que dispone-

mos es de menor calidad para estos ejemplos. Recordemos que hemos utilizado 100.000 evaluaciones del C_{max} del algoritmo GA_H para obtener estas soluciones.

Vamos ahora a comentar sucintamente los tiempos de CPU que cada uno de los métodos evaluados anteriormente necesita para resolver los problemas considerados. Ya vimos en el Capítulo 5 como la magnitud de los tiempos de cambio de partida apenas afecta a los tiempos de CPU, por lo que no es necesario mostrar los resultados para los 12 grupos de pruebas. Sí que mostramos, no obstante, los resultados para los grupos SSD10_P13, SSD10_P2, y SSD10_P3, que aparecen en las Tablas de la 6.17 a la 6.19.

Problema	RANDOM	GA_H	$SAOP_H$	$Spirit_H$	$GAReev_H$	NEH_H	$GAChen_H$	$GAPAC_H$	$GAMIT_H$
20x5	0,90	2,19	0,94	0,81	2,10	<0,5	1,14	1,05	0,80
20x10	1,43	2,83	1,58	1,46	2,75	<0,5	1,72	1,70	1,44
20x20	2,95	4,37	3,13	2,99	4,27	<0,5	3,23	3,22	2,95
50x5	1,82	3,13	1,94	1,79	3,85	<0,5	2,48	2,27	1,79
50x10	4,16	5,45	4,40	4,17	6,22	<0,5	4,84	4,63	4,13
50x20	11,17	10,14	9,12	8,69	10,73	<0,5	9,79	11,17	8,82
100x5	5,63	5,88	4,58	4,33	8,42	<0,5	6,79	6,18	4,46
100x10	17,61	11,74	9,86	9,72	13,86	0,63	16,51	17,49	10,11
100x20	47,32	47,61	45,75	46,66	49,70	2,32	49,84	49,04	46,42
200x10	51,34	43,22	48,88	50,39	49,91	10,11	57,40	53,50	49,45
200x20	114,89	102,71	114,58	117,09	109,92	30,15	122,10	117,91	115,73
Media	23,56	21,75	22,25	22,55	23,79	3,98	25,08	24,38	22,37

RANDOM=Regla aleatoria, GA_H =GA propuesto, $SAOP_H$ =Simulated annealing de Osman y Potts (1989) (adaptación), $Spirit_H$ =Tabu search de Widmer y Hertz (1989) (adaptación), $GAReev_H$ =GA de Reeves (1995) (adaptación), NEH_H =Algoritmo de Nawaz, Encore y Ham (1983) (adaptación), $GAChen_H$ =GA de Chen, Vempati y Aljaber (1995) (adaptación), $GAPAC_H$ =GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), $GAMIT_H$ =GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.17 – Tiempos de proceso (en segundos) utilizados por los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P13.

Problema	RANDOM	GA_H	SAOP_H	Spirit_H	GAReev_H	NEH_H	GACHen_H	GAPAC_H	GAMIT_H
20x5	0,79	2,20	0,93	0,79	2,08	<0,5	1,09	1,14	0,96
20x10	1,41	2,81	1,56	1,45	2,73	<0,5	1,71	1,70	1,42
20x20	2,81	4,22	2,99	2,82	4,13	<0,5	3,40	3,08	2,80
50x5	1,92	3,23	2,03	1,89	3,93	<0,5	2,56	2,37	1,88
50x10	4,04	5,31	4,25	4,05	6,09	<0,5	4,73	4,54	4,04
50x20	12,57	10,25	8,86	8,53	10,63	<0,5	9,87	12,63	8,84
100x5	5,64	5,39	4,17	4,08	8,16	<0,5	6,52	6,30	4,14
100x10	16,96	11,02	8,98	9,05	13,19	0,59	15,00	17,03	9,44
100x20	42,78	43,64	34,59	42,86	46,21	2,16	45,68	44,48	42,66
200x10	45,97	39,34	33,15	46,27	46,65	9,50	53,29	48,74	46,53
200x20	101,64	90,67	92,10	102,87	97,94	26,83	108,69	104,49	102,05
Media	21,50	19,83	17,60	20,42	21,98	3,60	22,96	22,41	20,43

RANDOM=Regla aleatoria, GA_H=GA propuesto, SAOP_H=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit_H=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev_H=GA de Reeves (1995) (adaptación), NEH_H=Algoritmo de Nawaz, Encore y Ham (1983) (adaptación), GACHen_H=GA de Chen, Vempati y Aljaber (1995) (adaptación), GAPAC_H=GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), GAMIT_H=GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.18 – Tiempos de proceso (en segundos) utilizados por los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P2.

Problema	RANDOM	GA _H	SAOP _H	Spirit _H	GAReev _H	NEH _H	GACHen _H	GAPAC _H	GAMIT _H
20x5	1,03	2,54	1,14	1,02	2,31	<0,5	1,30	1,25	0,99
20x10	1,86	3,38	2,02	1,89	3,20	<0,5	2,14	2,14	1,86
20x20	3,85	5,34	4,09	3,90	5,21	<0,5	4,18	4,15	3,87
50x5	2,56	3,99	2,70	2,60	4,60	<0,5	3,24	3,02	2,53
50x10	5,96	6,98	5,81	5,57	7,58	<0,5	6,32	6,49	5,57
50x20	19,86	16,46	13,17	13,79	16,26	<0,5	15,42	20,17	14,73
100x5	9,33	7,47	6,01	5,98	10,03	<0,5	9,79	9,82	6,06
100x10	25,91	21,21	13,17	18,61	23,06	0,95	25,15	26,47	19,28
100x20	63,63	66,04	56,53	66,61	68,95	3,95	67,72	65,85	65,58
200x10	69,36	61,99	54,57	71,13	69,37	17,33	76,70	72,55	70,92
200x20	151,33	135,18	118,27	153,48	142,33	40,64	158,51	154,31	151,41
Media	32,24	30,05	25,23	31,32	32,08	5,78	33,68	33,29	31,16

RANDOM=Regla aleatoria, GA_H=GA propuesto, SAOP_H=Simulated annealing de Osman y Potts (1989) (adaptación), Spirit_H=Tabu search de Widmer y Hertz (1989) (adaptación), GAReev_H=GA de Reeves (1995) (adaptación), NEH_H=Algoritmo de Nawaz, Encore y Ham (1983) (adaptación), GACHen_H=GA de Chen, Vempati y Aljaber (1995) (adaptación), GAPAC_H=GA de Ponnambalam, Aravindan y Chandrasekaran (2001) (adaptación), GAMIT_H=GA de Murata, Ishibuchi y Tanaka (1996a) (adaptación)

Tabla 6.19 – Tiempos de proceso (en segundos) utilizados por los métodos propuestos para el problema de la programación de la producción en el sector cerámico. Grupo de problemas SSD10_P3.

Como era de esperar, el método heurístico NEH_H es el más rápido con una clara diferencia. Resulta interesante observar como los tiempos de CPU para este método llegan a ser de más de medio minuto para los problemas más grandes de 200 trabajos, algo que contrasta con la heurística NEH con las mejoras de Taillard que se comentó en el Capítulo 3. El resto de métodos obtienen las soluciones en un tiempo muy similar, independientemente del número de máquinas que se consideran en cada etapa. Por ejemplo, para el caso de 3 máquinas por etapa y en los ejemplos de 200 trabajos y 20 etapas, (Tabla 6.19), los métodos necesitan todos entre algo menos de dos minutos y algo más de dos minutos y medio, y aunque existen algunas diferencias, estas no son lo suficientemente importantes como para insistir en ellas. El motivo de estas similitudes es muy claro. Prácticamente la totalidad del tiempo se invierte en el cálculo del C_{max} de las 5.000 secuencias que constituyen el criterio de parada, el resto de cálculos de los algoritmos consumen muy poco tiempo en comparación con éste. Debido a esto no planteamos una evaluación estableciendo como criterio de parada el tiempo transcurrido, ya que

los resultados serán esencialmente similares a los ya vistos. Al no haber ningún método más rápido ni más lento que los demás (al menos no de una manera importante), no obtendremos diferencias de eficacia con el criterio de parada establecido al tiempo transcurrido.

A partir de la comparativa realizada podemos decir que de nuevo el algoritmo genético propuesto, con la adaptación realizada para el problema de la programación de la producción en el sector azulejero, ha resultado ser mejor a todos los demás métodos adaptados, con unas mejoras que van desde el 52,4 % hasta el 135,26 % en el *IPSOM* total.

En la Sección 6.5.4.4 se comentó que por el momento la restricción de uso de máquinas se relajaba para el conjunto de 1320 problemas estudiados anteriormente. En colaboración con la empresa Cerypsa Cerámicas S.A. (<http://www.cerypsa.com>), se ha obtenido un conjunto de 10 problemas reales de programación de la producción donde existen todas las situaciones consideradas; tiempos de cambio de partida dependientes de la secuencia, dos etapas con tres máquinas no relacionadas en cada etapa y restricciones de uso de máquinas. Para cada uno de estos problemas conocemos la programación que el responsable de producción realizó en fábrica y disponemos de todos los datos necesarios, por lo que podemos comparar el C_{max} obtenido en este caso con el C_{max} que obtiene el algoritmo GA_H propuesto. Obtener todos los datos necesarios para poder alimentar a los algoritmos ha resultado ser una tarea muy costosa a la que la empresa se la dedicado mucho tiempo. Si tenemos en cuenta que por término medio las empresas cerámicas tienen más de 300 productos diferentes, en principio necesitaríamos tantas matrices de 300x300 con los tiempos de cambio de partida como máquinas haya disponibles, aparte de las matrices con los tiempos de proceso.

Los 10 problemas reales se han resuelto con el algoritmo GA_H propuesto con el criterio de optimización fijado a la evaluación de 50.000 C_{max} . Se han realizado cinco ejecuciones independientes y se ha calculado el C_{max} medio. Este resultado se compara con el C_{max} obtenido por el responsable de producción. La solución obtenida por este responsable es “manual” es decir, se ha obtenido mediante el uso de un listado con información sobre lo que hay que producir, un papel en blanco

y la experiencia. Este es el método más utilizado en el sector. Los resultados se muestran en Tabla 6.20.

Problema	n	m	M	C_{max} medio para el algoritmo GA_H	C_{max} “manual”	Diferencia (%)
PRUEBA1P	32	2	6	22943,1	24752	7,88
PRUEBA2P	32	2	6	23252,2	25061	7,78
PRUEBA3P	32	2	6	23245,4	25089	7,93
PRUEBA4P	32	2	6	23937,1	25842	7,96
PRUEBA5P	32	2	6	24339,2	24904	2,32
PRUEBA6P	32	2	6	24719,1	25330	2,47
PRUEBA7P	32	2	6	25929,8	28135	8,50
PRUEBA8P	32	2	6	26784,1	30302	13,13
PRUEBA9P	32	2	6	27889,5	31807	14,05
PRUEBA10P	32	2	6	27780,5	32490	16,95
Media						8,90

n =número de trabajos, m =número de etapas, M =número de máquinas totales, GA_H =GA propuesto

Tabla 6.20 – Resultados del algoritmo genético propuesto y la programación manual para los diez ejemplos reales de producción.

Cerypsa es una empresa de tamaño medio y tiene la salida de los hornos conectada a las líneas de clasificación, de ahí que solo dispongamos de dos etapas. La empresa dispone de tres líneas completas, y por lo tanto las seis máquinas “virtuales” que consideramos en los problemas. El número de productos a fabricar también es reducido, esto es debido a que Cerypsa prefiere producir pocos productos con grandes lotes de fabricación.

Como podemos observar, los resultados son muy prometedores. El algoritmo GA_H es entre un 2,32 % y un 16,95 % mejor que el método manual. Los resultados, junto con la secuenciación y programación obtenida por el algoritmo GA_H se han contrastado por parte de la empresa y la programación es factible y realizable. Todo esto quiere decir que la empresa puede realizar una programación

de la producción que es, de media, casi un 9 % más corta, o lo que es lo mismo, sin adquirir ninguna máquina adicional, sin realizar ningún tipo de cambio en las líneas productivas, la empresa dispone de un 9 % de capacidad productiva adicional. Se podría pensar que 50.000 evaluaciones del C_{max} son muchas y que el método propuesto ha necesitado mucho tiempo de CPU para obtener los resultados. La realidad es que los problemas reales que se han considerado son “pequeños”, ya que tienen pocos trabajos, máquinas y etapas. De media, el método GA_H ha necesitado 2.273,83 milisegundos, o lo que es lo mismo, apenas 2,27 segundos, un tiempo muy inferior a las tres o cuatro horas que de media necesita el responsable de producción para realizar la programación de la producción.

Todo indica que conforme el tamaño del problema aumente, las diferencias entre el método GA_H y el método manual se harán más y más grandes. Ejemplos son empresas con tres o incluso cuatro etapas, con cinco o seis máquinas no relacionadas por etapa y órdenes de producción de 50 o 60 trabajos.

6.7. Conclusiones del capítulo

En este capítulo se ha presentado un resumen de las principales características del sector de fabricación de azulejos o baldosas cerámicas así como un detallado estudio sobre el proceso de producción del azulejo. La caracterización del problema de producción nos ha permitido identificarlo como un problema de tipo taller de flujo híbrido con tiempos de cambio de partida dependientes de la secuencia, máquinas paralelas no relacionadas en cada etapa y elegibilidad de máquinas. La extensa revisión del estado del arte en este tipo de problemas ha puesto de manifiesto que no existen algoritmos que contemplen este problema y todas sus características. De esta manera, y siguiendo los resultados de capítulos anteriores, hemos realizado una adaptación del algoritmo presentado en el Capítulo 5. La característica principal de esta adaptación estriba en que el algoritmo considera el problema de la asignación a máquinas dentro de una etapa de producción de manera que las tareas se asignen siempre a la primera máquina que pueda terminar el trabajo, teniendo en cuenta los tiempos de cambio y la posibilidad de que las máquinas no tengan las mismas características o que no todas puedan procesar la tarea. De nuevo, este algoritmo se ha calibrado mediante el uso del diseño

experimental y del análisis de la varianza. En este caso se ha utilizado un conjunto de 1320 problemas con distintas características en lo que se refiere a número de trabajos, número de etapas, máquinas por etapa y magnitud de los tiempos de cambio de partida.

Una vez calibrado, el algoritmo se ha comparado con otros ocho métodos resultantes de realizar adaptaciones sobre los mejores algoritmos vistos en capítulos anteriores. Bajo el criterio de máximo número de C_{max} evaluados, el algoritmo es entre un 53 % y un 135 % más eficaz que el segundo mejor método.

Tras esta evaluación se ha probado el comportamiento del algoritmo con datos reales extraídos de una empresa del sector. Los resultados indican que el algoritmo es capaz de obtener programas de producción en apenas unos segundos que son realistas y realizables y que además son entre un 2,32 % y un 16,95 % más cortos que las programaciones proporcionadas por el responsable de producción en la empresa.

7

CAPÍTULO

CONCLUSIONES

La programación de la producción es un problema muy importante dentro de las industrias, sobre todo en el entorno actual tan competitivo, donde al mismo tiempo las empresas deben ofrecer un catálogo de productos diferenciado y diversificado a un precio muy ajustado. La existencia de más productos unido al hecho de que los clientes cada vez hacen pedidos más pequeños y más frecuentes provoca que cada vez los tamaños de los lotes de producción sean más reducidos, por lo que nos encontramos con la paradoja de que las empresas tienen que producir una mayor variedad de productos con sistemas de producción altamente automatizados y productivos que están diseñados para producir grandes lotes de producto y donde los cambios en la producción provocan paradas de las máquinas, pérdidas de productividad e incrementos de los costes. Además, los clientes son cada vez más exigentes, por lo que sus requerimientos de calidad, precio y rapidez en la entrega son cada vez mayores. Por todos estos motivos se hace necesario un método para realizar programaciones flexibles de la producción que sean realistas y que ayuden a las empresas a utilizar de una manera más eficiente los recursos disponibles, así como a dar respuesta a las demandas de los clientes.

Una de las configuraciones productivas más frecuente es la del taller de flujo. En la presente Tesis Doctoral se han estudiado diversos problemas de tipo taller de flujo, concretamente el taller de flujo estándar, el taller de flujo con tiempos de cambio de partida dependientes de la secuencia y una versión mucho más realista y general como es el taller de flujo híbrido con máquinas paralelas no relacionadas, restricción de uso de máquinas y tiempos de cambio de partida dependientes de la secuencia. Todos estos problemas, especialmente el último, son de mucha utilidad para realizar la programación de la producción en entornos de producción reales como los que se han comentado anteriormente. Es importante destacar que hasta el momento no se han propuesto algoritmos de carácter general que permitan dar una solución en estos casos.

Para cada uno de los tres problemas se ha realizado una extensa revisión del estado del arte y en los dos primeros casos se ha llevado a cabo una completa evaluación comparativa que comprende más de 35 métodos y algoritmos existentes. Se han propuesto un total de cinco nuevos algoritmos genéticos para resolver estos problemas con una serie de características novedosas. Concretamente se han presentado cuatro operadores de cruce y un esquema generacional que busca al mismo tiempo aumentar la presión del algoritmo genético y evitar la convergencia prematura. También se ha desarrollado un operador de reinicialización que previene la convergencia hacia óptimos locales. Asimismo, algunos de estos algoritmos incorporan un esquema híbrido con búsqueda local basado en probabilidades que mantiene un control sobre el número de veces que se ha intentado mejorar un individuo sin éxito, de manera que si el mejor individuo no se puede mejorar se intenta mejorar el segundo mejor y así sucesivamente.

Estos nuevos operadores y esquemas se han considerado junto con los operadores tradicionales en un total de 17 amplios experimentos computacionales encaminados a obtener una calibración precisa de los distintos niveles y/o variantes de los operadores y probabilidades que se sabe afectan al comportamiento de los algoritmos genéticos. En estos experimentos se han evaluado un total de 20.160 algoritmos distintos con los que se han resuelto 8.091.000 problemas. Los resultados de la parametrización muestran que, en todos los experimentos

realizados, uno de los operadores de cruce propuestos, el “*Similar Block 2-Point Order Crossover*” (SB2OX) resulta ser más eficaz que otros operadores de cruce, en especial el “*Partially Matched Crossover*” (PMX), que está reconocido como uno de los mejores operadores de cruce para el problema del taller de flujo. Los resultados también indican que los algoritmos propuestos presentan un comportamiento robusto, ya que la mayoría de los niveles y/o variantes de los parámetros y operadores utilizados se mantienen a pesar de cambiar considerablemente el tipo de problema considerado.

Esta parametrización ha resultado en unos algoritmos muy eficaces y eficientes. Concretamente, para el taller de flujo, los dos algoritmos propuestos, el GA y el HGA han resultado ser un 9 % y un 49 % mejores que el método más eficaz de entre 25 métodos evaluados con el criterio de parada fijado a un máximo de secuencias evaluadas, donde para las pruebas se ha utilizado el banco de 120 problemas de Taillard (1993). Entre estos métodos se incluyen técnicas heurísticas y metaheurísticas como recocido simulado, búsqueda tabú u otros algoritmos genéticos de los más citados y reconocidos en la literatura. Bajo el criterio de máximo tiempo transcurrido, los dos algoritmos resultan ser entre un 24 % y un 46 % mejores que el resto. Diversos estudios realizados indican que los algoritmos mantienen su nivel de prestaciones independientemente del número de secuencias evaluadas que se permiten. Consideramos estos resultados como buenos, dado que el taller de flujo, como se ha visto, ha sido intensamente estudiado en los últimos 50 años y se han propuesto numerosas técnicas muy eficaces.

Para el caso del taller de flujo con tiempos de cambio de partida dependientes de la secuencia se han evaluado seis métodos heurísticos y metaheurísticos específicamente diseñados para este problema. También se han realizado adaptaciones de los mejores métodos propuestos para el taller de flujo estándar así como de los dos algoritmos genéticos propuestos. En este caso se ha utilizado un nuevo conjunto de pruebas que comprende 480 ejemplos de distintas características. Los resultados indican que el primer algoritmo propuesto, GA_{sd} , es entre un 51 % y un 55 % mejor que el resto de métodos bajo el criterio de máximo número de secuencias evaluadas y entre un 171 % y un 183 % mejor bajo el criterio

de máximo tiempo transcurrido. El segundo algoritmo propuesto, el HGA_{sd} ha obtenido mejores resultados, siendo entre un 366 % y un 488 % más eficaz para el criterio de máximas secuencias evaluadas y entre un 256 % y un 300 % mejor con el criterio de máximo tiempo transcurrido. Todos estos resultados posicionan los algoritmos propuestos como los más adecuados hasta la fecha para resolver el problema considerado. Adicionalmente, las adaptaciones realizadas de otros métodos existentes para el taller de flujo estándar a este problema han proporcionado muy buenos resultados, en muchos casos mejores a los métodos diseñados específicamente para el problema con tiempos de cambio de partida.

De igual manera se ha propuesto un nuevo algoritmo genético para resolver un complejo problema de programación de la producción que contiene todos los aspectos considerados anteriormente además de máquinas paralelas no relacionadas y restricción de uso de máquinas. Este problema no ha sido considerado de manera general con anterioridad. Se ha propuesto la adaptación de siete métodos heurísticos y metaheurísticos con un nuevo esquema de resolución que contempla la separación, pero no completa, de las dimensiones de *secuenciación* de los trabajos y de *asignación* a máquinas dentro de cada etapa. Adicionalmente, se ha propuesto un nuevo esquema de asignación basado en determinar qué máquina dentro de una etapa puede terminar antes el trabajo considerado en el menor espacio de tiempo, teniendo en cuenta las restricciones de uso de máquinas, máquinas no relacionadas y los tiempos de cambio de partida dependientes de la secuencia. Para evaluar este algoritmo se ha utilizado un nuevo conjunto de 1320 pruebas que comprenden distintas configuraciones de etapas, número de máquinas por etapa y magnitud de tiempos de cambio de partida. Los resultados indican que el algoritmo propuesto, el GA_H resulta ser entre un 53 % y un 135 % mejor al segundo mejor método de entre los adaptados con el criterio de optimización fijado al máximo número de evaluaciones. Lo cual de nuevo resulta ser un resultado interesante. En resumen, podemos decir que los algoritmos genéticos propuestos han resultado ser mejores para los tres tipos de problemas considerados, de creciente dificultad, que el resto de métodos considerados.

Por último, el algoritmo GA_H se ha probado en un caso real con 10 problemas extraídos de una empresa azulejera y se ha comparado con las soluciones obtenidas con métodos manuales y proporcionadas por el responsable de la planificación y programación de la producción en la empresa. Los resultados indican que el algoritmo propuesto es entre un 2,32 % y un 16,95 % mejor que el método manual. Las secuencias y la programación obtenidas son factibles y realizables en la empresa. Luego es posible realizar una programación de la producción eficaz y eficiente que puede aumentar en gran medida la capacidad de producción y de respuesta en las empresas del sector considerado.

Este algoritmo se ha implementado en un prototipo de software para la programación de la producción que actualmente se está implantando en dos empresas cerámicas del sector.

A lo largo de la presente Tesis Doctoral se ha considerado la minimización del tiempo máximo de flujo o “makespan” (C_{max}), que es el criterio más utilizado en la literatura y un criterio importante desde el punto de vista práctico para su utilización en el sector azulejero y otros como el textil. Líneas futuras de investigación podrían considerar otros objetivos como puede ser la minimización del tiempo de flujo medio (\bar{C}), criterios que contemplen fechas de entrega o de finalización como la minimización del retraso máximo (T_{max}), el retraso medio (\bar{T}) o el número de trabajos retrasados (N_T). También se podrían diseñar técnicas de programación de la producción que puedan considerar dos o más objetivos de manera simultánea.

Asimismo es posible considerar aspectos adicionales en la programación de la producción como son la existencia de capacidad de almacenamiento de producto en curso limitada entre máquinas o etapas, tiempos de transporte no nulos de producto en curso entre etapas y/o máquinas, solapes de los trabajos en máquinas consecutivas, etc...

Otro trabajo futuro de interés es el diseño de algoritmos generales que consideren problemas de tipo taller de flujo híbrido sin separar las decisiones de secuenciación y asignación, es decir, que la asignación no se haga con independencia del algoritmo.

Actualmente estamos trabajando en todas estas líneas en el marco de dos proyectos interdisciplinares financiados por la Universidad Politécnica de Valencia y en un proyecto coordinado financiado por el Ministerio De Ciencia y Tecnología. El objetivo principal de estos proyectos es el de proponer métodos y algoritmos de utilidad en entornos productivos reales, concretamente en el sector cerámico. Dentro de este sector también estamos colaborando activamente a través de dos convenios con las empresas Halcón Cerámicas S.A. y Cerypsa Cerámicas S.A.

REFERENCIAS

- Adler, L., Fraiman, N., Kobacker, E., Pinedo, M., Plotnicoff, J. C., y Wu, T. P. (1993). BPSS: A Scheduling Support System for the Packaging Industry. *Operations Research*, 41(4):641–648.
- Aghezzaf, E. H., Artiba, A., Moursli, O., y C., T. (1995). Hybrid flowshop problems, a decomposition based heuristic approach. En *Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'95*, páginas 43–56, Marrakech. FUCAM - INRIA.
- Aho, A. V., Hopcroft, J. E., y Ullman, J. (1983). *Data Structures and Algorithms*. Addison-Wesley series in computer science and information. Addison-Wesley, Reading.
- Alcaraz, J. (2001). *Algoritmos Genéticos para Programación de Proyectos con Recursos Limitados*. Tesis Doctoral, Departamento de Estadística e Investigación Operativa Aplicadas y Calidad. Universidad Politécnica de Valencia.
- Alcaraz, J. y Maroto, C. (2001). A Robust Genetic Algorithm for Resource Allocation in Project Scheduling. *Annals of Operations Research*, 102:83–109.
- Alcaraz, J., Maroto, C., y Ruiz, R. (2003). Solving the Multi-Mode Resource-Constrained Project Scheduling Problem with Genetic Algorithms. *Journal of the Operational Research Society*, 54:614–626.
- Allahverdi, A. y Aldowaisan, T. (2001). Minimizing total completion time in a no-wait flowshop with sequence-dependent additive changeover times. *Journal of the Operational Research Society*, 52:449–462.

- Allahverdi, A., Gupta, J. N. D., y Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *OMEGA, The international Journal of Management Science*, 27:219–239.
- Anderson, E. J., Glass, C. A., y Potts, C. N. (1997). Machine scheduling. En Aarts, E. y Lenstra, J. K., editores, *Local Search in Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, páginas 361–414, Chichester. John Wiley & Sons.
- Andrés, C. (2001). *Programación de la Producción en Talleres de Flujo Híbridos con Tiempos de Cambio de Partida Dependientes de la Secuencia. Modelos, Métodos y Algoritmos de Resolución. Aplicación a Empresas del Sector Cerámico*. Tesis Doctoral, Departamento de Organización de Empresas. Universidad Politécnica de Valencia.
- Arthanary, T. S. y Ramaswamy, K. G. (1971). An Extension of Two Machine Sequencing Problem. *OPSEARCH, The Journal of the Operational Research Society of India*, 8(4):10–22.
- Artiba, A. y Elmaghraby, S. E., editores (1997). *The Planning and Scheduling of Production Systems. Methodologies and Applications*. Chapman & Hall, London.
- ASCER (1998). El Sector Azulejero Español en 1997. Informe técnico, Asociación Española de Fabricantes de Azulejos y Pavimentos Cerámicos (ASCER), Castellón.
- ASCER (1999). El sector español de fabricantes de baldosas cerámicas, año 1998. Informe técnico, Asociación Española de Fabricantes de Azulejos y Pavimentos Cerámicos (ASCER), Castellón.
- ASCER (2000). El sector español de fabricantes de baldosas cerámicas, año 1999. Informe técnico, Asociación Española de Fabricantes de Azulejos y Pavimentos Cerámicos (ASCER), Castellón.

- ASCER (2001). El sector español de fabricantes de baldosas cerámicas, año 2000. Informe técnico, Asociación Española de Fabricantes de Azulejos y Pavimentos Cerámicos (ASCER), Castellón.
- ASCER (2002). Los sectores español y mundial de fabricantes de baldosas cerámicas, año 2001. Informe técnico, Asociación Española de Fabricantes de Azulejos y Pavimentos Cerámicos (ASCER), Castellón.
- Ashour, S. (1970). An experimental investigation and comparative evaluation of flow-shop scheduling techniques. *Operations Research*, 18(3):541–549.
- ATECE (1990). *Tecnología de la Fabricación de Azulejos*. Asociación Española de Técnicos Cerámicos e IMPIVA.
- Bagchi, T. P. (1999). *Multiobjective Scheduling by Genetic Algorithms*. Kluwer Academic Publishers, Dordrecht.
- Bagchi, T. P. y Deb, K. (1996). Calibration of GA Parameters: The Design of Experiments Approach. *Computer Science and Informatics*, 26(3):46–56.
- Baker, J. E. (1985). Adaptive Selection Methods for Genetic Algorithms. En Grefenstette, J. J., editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, páginas 101–111, Hillside. Lawrence Erlbaum associates.
- Baker, K. R. (1974). *Introduction to Sequencing and Scheduling*. John Wiley & Sons, New York.
- Ben-Daya, M. y Al-Fawzan, M. (1998). A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research*, 109:88–95.
- Bestwick, P. F. y Hastings, N. A. J. (1976). A new bound for machine scheduling. *Operational Research Quarterly*, 27(2):479–487.
- Biegel, J. E. y Davern, J. J. (1990). Genetic Algorithms and Job Shop Scheduling. *Computers and Industrial Engineering*, 19(1):81–91.

- Blackstone, Jr, J. H., Phillips, D. T., y Hogg, G. L. (1982). A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research*, 20(1):27–45.
- Błazewicz, J., Dror, M., y Węglarz, J. (1991). Mathematical programming formulations for machine scheduling: A survey. *European Journal of Operational Research*, 51:283–300.
- Błazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., y Węglarz, J. (1996). *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, Berlin.
- Błazewicz, J., Lenstra, J. K., y Rinnooy Kan, A. H. G. (1983). Scheduling Subject to Constraints: Classification and Complexity. *Discrete Applied Mathematics*, 5:11–24.
- Bonney, M. C. y Gundry, S. W. (1976). Solutions to the Constrained Flowshop Sequencing Problem. *Operational Research Quarterly*, 27(4):869–883.
- Botta-Genoulaz, V. (2000). Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. *International Journal of Production Economics*, 64:101–111.
- Bowman, E. H. (1959). The schedule-sequencing problem. *Operations Research*, 7(5):621–624.
- Brah, S. A. y Hunsucker, J. L. (1991). Branch and bound algorithm for the flow shop with multiple processors. *European Journal of Operational Research*, 51:88–99.
- Brah, S. A. y Loo, L. L. (1999). Heuristics for scheduling in a flow shop with multiple processors. *European Journal of Operational Research*, 113:113–122.
- Brown, A. P. G. y Lomnicki, Z. A. (1966). Some applications of the branch-and-bound algorithm to the machine scheduling problem. *Operational Research Quarterly*, 17(2):173–186.

- Bruns, R. (1993). Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling. En Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA)*, páginas 352–359. Morgan Kaufmann.
- Bucknall, J. (2001). *The Tomes of Delphi: Algorithms and Data Structures*. Wordware Publishing, Inc., Plano.
- Byung Park, Y., Pegden, C. D., y Enscore, Jr, E. E. (1984). A survey and evaluation of static flowshop scheduling heuristics. *International Journal of Production Research*, 22(1):127–141.
- Campbell, H. G., Dudek, R. A., y Smith, M. L. (1970). A Heuristic Algorithm for the n Job, m Machine Sequencing Problem. *Management Science*, 16(10):B–630–B–637.
- Cantú, M. (2001). *Mastering Delphi 6*. Sybex, Inc., San Francisco.
- Carlier, J. y Rebaï, I. (1996). Two branch and bound algorithms for the permutation flow shop problem. *European Journal of Operational Research*, 90:238–251.
- Chandrasekharan, R. y Chaudhuri, D. (1992a). A multi-stage parallel-processor flowshop problem with minimum flowtime. *European Journal of Operational Research*, 57:111–122.
- Chandrasekharan, R. y Chaudhuri, D. (1992b). Scheduling in n -jobs, m -stage flowshops with parallel processors to minimize makespan. *International Journal of Production Economics*, 27:137–143.
- Chandrasekharan, R. y Ziegler, H. (1997). Heuristics for scheduling in a flowshop with setup, processing and removal times separated. *Production Planning and Control*, 8(6):568–576.
- Charte Ojeda, F. (2001a). *Guía práctica para usuarios de Delphi 6*. guías prácticas. Anaya Multimedia, Madrid.
- Charte Ojeda, F. (2001b). *Programación con Delphi 6 y Kylix*. Anaya Multimedia, Madrid.

- Charte Ojeda, F. (2003). *Ensamblador para DOS, Linux y Windows. programación*. Anaya Multimedia, Madrid.
- Chen, B., Glass, C. A., Potts, C. N., y Strusevich, V. A. (1996). A New Heuristic for Three-Machine Flow Shop Scheduling. *Operations Research*, 44(6):891–897.
- Chen, C.-L., Neppalli, R. V., y Aljaber, N. (1996). Genetic Algorithms Applied to the Continuous Flow Shop Problem. *Computers and Industrial Engineering*, 30(4):919–929.
- Chen, C.-L., Vempati, V. S., y Aljaber, N. (1995). An application of genetic algorithms for flow shop problems. *European Journal of Operational Research*, 80:389–396.
- Cheng, T. C. E., Gupta, J. N. D., y Wang, G. (2000). A Review of Flowshop Scheduling Research with Setup Times. *Production and Operations Management*, 9(3):262–282.
- Cleveland, G. A. y Smith, S. F. (1989). Using Genetic Algorithms to Schedule Flow Shop Releases. En Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, páginas 160–169, San Mateo. Morgan Kaufmann.
- Companys Pascual, R. y Corominas Subias, A. (1996). *Organización de la producción II. Dirección de operaciones 4*. Edicions UPC.
- Conway, R. W., Maxwell, W. L., y Miller, L. W. (1967). *Theory of Scheduling*. Addison-Wesley, Reading.
- Corwin, B. D. y Esogbue, A. O. (1974). Two Machine Flow Shop Scheduling Problems with Sequence Dependent Setup Times: A Dynamic Programming Approach. *Naval Research Logistics Quarterly*, 21:515–524.
- Cotta, C. y Troya, J. M. (1998). Genetic Formal Recombination in Permutation Flowshop Problems. *Evolutionary Computation*, 6(1):25–44.

- Dalmau Porta, J. I. y De Miguel Fernández, E. (1991). *El Azulejo, Estudio Sectorial*. Universidad Politécnica de Valencia, primera edición.
- Danneberg, D., Tautenhahn, T., y Werner, F. (1999). A Comparison of Heuristic Algorithms for Flow Shop Scheduling Problems with Setup Times and Limited Batch Size. *Mathematical and Computer Modelling*, 29:101–126.
- Dannenbring, D. G. (1977). An Evaluation of Flow Shop Sequencing Heuristics. *Management Science*, 23(11):1174–1182.
- Das, S. R., Gupta, J. N. D., y Khumawala, B. M. (1995). A Savings Index Heuristic Algorithm for Flowshop Scheduling with Sequence Dependent Set-up Times. *Journal of the Operational Research Society*, 46:1365–1373.
- Davis, L. (1985a). Applying Adaptive Algorithms to Epistatic Domains. En Aravind, K. J., editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence, IJCAI*, páginas 162–164, Los Angeles. Morgan Kaufmann.
- Davis, L. (1985b). Job Shop Scheduling with Genetic Algorithms. En Grefenstette, J. J., editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, páginas 136–140, Hillsdale. Lawrence Erlbaum Associates.
- Davis, L., editor (1996). *Handbook of Genetic Algorithms*. International Thomson Computer Press, London.
- Davoud Pour, H. (2001). A new heuristic for the n -job, m -machine flow-shop problem. *Production Planning and Control*, 12(7):648–653.
- De Jong, K. A. (1985). Genetic Algorithms: A 10 Year Perspective. En Grefenstette, J. J., editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, páginas 169–177, Hillsdale. Lawrence Erlbaum associates.
- Drexl, A. y Kimms, A., editores (1998). *Beyond Manufacturing Resource Planning (MRP II). Advanced Models and Methods for Production Planning*. Springer-Verlag, Berlin.

- Dudek, R. A., Panwalkar, S. S., y Smith, M. L. (1992). The Lessons of Flowshop Scheduling Research. *Operations Research*, 40(1):7–13.
- Dudek, R. A. y Teuton, Jr, O. F. (1964). Development of m -Stage decision Rule for Scheduling n Jobs Through m Machines. *Operations Research*, 12(3):471–497.
- Escardino Benlloch, A. y González Cudilleiro, M. (1991). *Azulejos y Pavimentos Cerámicos Españoles*. Ministerio de Industria, Comercio y Turismo, Madrid, primera edición.
- Fogarty, T. C. (1989). Varying the Probability of Mutation in the Genetic Algorithm. En Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, páginas 104–109, San Mateo. Morgan Kaufmann.
- Ford, F. N., Bradbard, D. A., Ledbetter, W. N., y Cox, J. F. (1987). Use of Operations Research in Production Management. *Production and Inventory Management*, 28(3):59–62.
- Framinan, J. M., Leisten, R., y Rajendran, C. (2003). Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, 41(1):121–148.
- Framinan, J. M., Leisten, R., y Ruiz-Usano, R. (2002). Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. *European Journal of Operational Research*, 141:559–569.
- French, S. (1982). *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood Limited, Chichester.
- Frieze, A. M. y Yadegar, J. (1989). A new integer programming formulation for the permutation flowshop problem. *European Journal of Operational Research*, 40:90–98.
- Garey, M. R. y Johnson, D. S. (1979). *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

- Garey, M. R., Johnson, D. S., y Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2):117–129.
- Gelders, L. F. y Sambandam, N. (1978). Four simple heuristics for scheduling a flow-shop. *International Journal of Production Research*, 16(3):221–231.
- Gómez López, J. D. (1999). *Las baldosas cerámicas en Castellón: el impacto de la globalización en una industria tradicional*. Universidad de Alicante.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading.
- Goldberg, D. E. y Lingle, Jr, R. (1985). Alleles, loci , and the traveling salesman problem. En Grefenstette, J. J., editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, páginas 154–159, Hillsdale. Lawrence Erlbaum Associates.
- Gourgand, M., Grangeon, N., y Norre, S. (1999). Metaheuristics for the deterministic hybrid flow shop problem. En *Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'99*, páginas 136–145, Glasgow. FUCAM - INRIA.
- Grabowski, J. (1980). On Two-Machine Scheduling with Release and Due Dates to Minimize Maximum Lateness. *OPSEARCH, The Journal of the Operational Research Society of India*, 17(4):133–154.
- Grabowski, J., Nowicki, E., y Zdrzałka, S. (1986). A block approach for single-machine scheduling with release and due dates. *European Journal of Operational Research*, 26:278–285.
- Grabowski, J., Skubalska, E., y Smutnicki, C. (1983). On Flow Shop Scheduling with Release and Due Dates to Minimize Maximum Lateness. *Journal of the Operational Research Society*, 34(7):615–620.
- Graham, R. L., Lawler, E. L., Lenstra, J. K., y Rinnooy Kan, A. H. G. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, 5:287–326.

- Graves, S. C. (1981). A Review of Production Scheduling. *Operations Research*, 29(4):646–675.
- Grefenstette, J., Gopal, R., Rosmaita, B., y Van Gucht, D. (1985). Genetic Algorithms for the Traveling Salesman Problem. En Grefenstette, J. J., editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, páginas 160–168, Hillsdale. Lawrence Erlbaum Associates.
- Guinet, A. G. (1991). Textile Production Systems: A Succession of Non-identical Parallel Processor Shops. *Journal of the Operational Research Society*, 42(8):655–671.
- Guinet, A. G. y Solomon, M. M. (1996). Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time. *International Journal of Production Research*, 34(6):1643–1654.
- Gupta, J. N. D. (1971). A Functional Heuristic Algorithm for the Flowshop Scheduling Problem. *Operational Research Quarterly*, 22(1):39–47.
- Gupta, J. N. D. (1972). Heuristic Algorithms for Multistage Flowshop Scheduling Problem. *AIEE Transactions*, 4(1):11–18.
- Gupta, J. N. D. (1975). A Search Algorithm for the Generalized Flowshop Scheduling Problem. *Computers and Operations Research*, 2:83–90.
- Gupta, J. N. D. (1978). A Search Algorithm for the Traveling Salesman Problem. *Computers and Operations Research*, 5:243–250.
- Gupta, J. N. D. (1986). Flowshop Schedules with Sequence Dependent Setup Times. *Journal of the Operations Research Society of Japan*, 29(3):206–219.
- Gupta, J. N. D. (1988). Two-Stage, Hybrid Flowshop Scheduling Problem. *Journal of the Operational Research Society*, 39(4):359–364.
- Gupta, J. N. D. y Darrow, W. P. (1986). The two-machine sequence dependent flowshop scheduling problem. *European Journal of Operational Research*, 24:439–446.

- Gupta, J. N. D. y Dudek, R. A. (1971). Optimality Criteria for Flowshop Schedules. *AIEE Transactions*, 3(3):199–205.
- Gupta, J. N. D., Hariri, A. M. A., y Potts, C. N. (1997). Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. *Annals of Operations Research*, 69:171–191.
- Gupta, J. N. D. y Tunc, E. A. (1991). Schedules for a two-stage hybrid flowshop with parallel machines at the second stage. *International Journal of Production Research*, 29(7):1489–1502.
- Gupta, J. N. D. y Tunc, E. A. (1998). Minimizing tardy jobs in a two-stage hybrid flowshop. *International Journal of Production Research*, 36(9):2397–2417.
- Gupta, S. K. (1982). n jobs and m machines job-shop problems with sequence-dependent set-up times. *International Journal of Production Research*, 20(5):643–656.
- Ho, J. C. y Chang, Y.-L. (1991). A new heuristic for the n -job, m -machine flowshop problem. *European Journal of Operational Research*, 52:194–202.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor.
- Hong, T.-P. y Wang, H.-S. (1998). Automatically Adjusting Crossover Ratios of Multiple Crossover Operators. *Journal of Information Science and Engineering*, 14:369–390.
- Hundal, T. S. y Rajgopal, J. (1988). An extension of Palmer's heuristic for the flow shop scheduling problem. *International Journal of Production Research*, 26(6):1119–1124.
- Hunsucker, J. L. y Shah, J. R. (1992). Performance of Priority Rules in a Due Date Flow Shop. *OMEGA, The international Journal of Management Science*, 20(1):73–89.

- Hunsucker, J. L. y Shah, J. R. (1994). Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment. *European Journal of Operational Research*, 72:102–114.
- Ignall, E. y Schrage, L. E. (1965). Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems. *Operations Research*, 13(3):400–412.
- Ishibuchi, H., Misaki, S., y Tanaka, H. (1995). Modified simulated annealing algorithms for the flow shop sequencing problem. *European Journal of Operational Research*, 81:388–398.
- Jain, N. y Bagchi, T. P. (2000). Hybridized GAs: Some new results in flowshop scheduling. Pittsburg. IASTED International Conference on Modelling and Simulation (MS'2000).
- Jain, N., Bagchi, T. P., y Wagneur, E. (2000). Flowshop scheduling by hybridized GA: Some new results. *International Journal of Industrial Engineering*, 7(3):213–223.
- Jawahar, N., Aravindan, P., y Ponnambalam, S. G. (1998). A genetic Algorithm for Scheduling Flexible Manufacturing Systems. *The International Journal of Advanced Manufacturing Technology*, 14:588–607.
- Jawahar, N., Aravindan, P., Ponnambalam, S. G., y Karthikeyan Aravind, A. (1998). A genetic algorithm-based scheduler for setup-constrained FMC. *Computers in Industry*, 35:291–310.
- Johnson, L. A. y Montgomery, D. C. (1974). *Operations Research in Production Planning, Scheduling and Inventory Control*. John Wiley & Sons, New York.
- Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:61–68.
- Kelley, D. (1995). *Teoría de autómatas y lenguajes formales*. Prentice Hall, Madrid.

- Kim, S. C. y Bobrowski, P. M. (1994). Impact of sequence-dependent setup time on job shop scheduling performance. *International Journal of Production Research*, 32(7):1503–1520.
- Kim, Y.-D. (1993). Heuristics for Flowshop Scheduling Problems Minimizing Mean Tardiness. *Journal of the Operational Research Society*, 44(1):19–28.
- Kim, Y.-D., Lim, H.-G., y Park, M.-W. (1996). Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. *European Journal of Operational Research*, 91:124–143.
- King, J. R. y Spachis, A. S. (1980). Heuristics for flow-shop scheduling. *International Journal of Production Research*, 18(3):345–357.
- Knuth, D. E. (1998). Sorting and searching. En *The art of computer programming*, volumen 3 de *Addison-Wesley series in computer science and information*. Addison-Wesley, Reading, segunda edición.
- Kochhar, S. y Morris, R. J. T. (1987). Heuristic Methods for Flexible Flow Line Scheduling. *Journal of Manufacturing Systems*, 6(4):299–314.
- Koulamas, C. (1998). A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research*, 105:66–71.
- Lageweg, B. J., Lenstra, J. K., y Rinnooy Kan, A. H. G. (1978). A general bounding scheme for the permutation flow-shop problem. *Operations Research*, 26(1):53–67.
- Lawler, E. L., Lenstra, J. K., y Rinnooy Kan, A. H. G. (1993). Sequencing and Scheduling: Algorithms and Complexity. En Graves, S. C., Rinnooy Kan, A. H. G., y Zipkin, P. H., editores, *Logistics of Production and Inventory*, volumen 4 de *Handbooks in Operations Research and Management Science*, Amsterdam. Elsevier Science Publishers, B. V.
- Ledbetter, W. N. y Cox, J. F. (1977). Operations Research in Production Management: An Investigation of Past and Present Utilization. *Production and Inventory Management*, 18(3):84–91.

- Li, S. (1997). A hybrid two-stage flowshop with part family, batch production, major and minor set-ups. *European Journal of Operational Research*, 102:142–156.
- Lindo Systems Inc. (1999). *LINGO user's guide, versión 6.0*.
- Linn, R. y Zhang, W. (1999). Hybrid Flow Shop Scheduling: A Survey. *Computers and Industrial Engineering*, 37:57–61.
- Lomnicki, Z. A. (1965). A Branch and Bound Algorithm for the Exact Solution of the Three-Machine Scheduling Problem. *Operational Research Quarterly*, 16(1):89–100.
- MacCarthy, B. L. y Liu, J. (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31(1):59–79.
- Mattfeld, D. C. (1996). *Evolutionary Search and the Job Shop; Investigations on Genetic Algorithms for Production Scheduling*. Production and Logistics. Springer/Physica Verlag, Berlin.
- McKay, K. N., Pinedo, M., y Webster, S. (2002). Practice-Focused Research Issues for Scheduling Systems. *Production and Operations Management*, 11(2):249–258.
- McKay, K. N., Safayeni, F. R., y Buzacott, J. A. (1988). Job-Shop Scheduling Theory: What is Relevant? *Interfaces*, 4(18):84–90.
- McMahon, G. B. y Burton, P. G. (1967). Flow-shop scheduling with the branch-and-bound method. *Operations Research*, 15(3):473–481.
- Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, tercera edición.
- Miyazaki, S., Nishiyama, N., y Hashimoto, F. (1978). An Adjacent Pairwise Approach to the Mean Flow-Time Scheduling Problem. *Journal of the Operations Research Society of Japan*, 21(2):287–299.

- Moccellin, J. V. (1995). A New Heuristic Method for the Permutation Flow Shop Scheduling Problem. *Journal of the Operational Research Society*, 46:883–886.
- Moccellin, J. V. y dos Santos, M. O. (2000). An adaptive hybrid metaheuristic for permutation flowshop scheduling. *Control and Cybernetics*, 29(3):761–771.
- Montgomery, D. C. (2000). *Design and Analysis of Experiments*. John Wiley & Sons, quinta edición.
- Montgomery, D. C. y Runger, G. C. (1994). *Applied Statistics and Probability for Engineers*. John Wiley & Sons, New York.
- Moursli, O. (1999). *Scheduling the Hybrid Flowshop*. Tesis Doctoral, Faculté des Sciences Économiques, Sociales et Politiques, Université Catholique de Louvain.
- Murata, T., Ishibuchi, H., y Tanaka, H. (1996a). Genetic Algorithms for Flowshop Scheduling Problems. *Computers and Industrial Engineering*, 30(4):1061–1071.
- Murata, T., Ishibuchi, H., y Tanaka, H. (1996b). Multi-Objective Genetic Algorithm and its Applications to Flowshop Scheduling. *Computers and Industrial Engineering*, 30(4):957–968.
- Nabeshima, I. (1967). On the bound of makespans and its application in m machine scheduling problem. *Journal of the Operations Research Society of Japan*, 9(3-4):98–136.
- Nawaz, M., Enscore, Jr, E. E., y Ham, I. (1983). A Heuristic Algorithm for the m -Machine, n -Job Flow-shop Sequencing Problem. *OMEGA, The International Journal of Management Science*, 11(1):91–95.
- Norman, B. A. (1999). Scheduling flowshops with finite buffers and sequence dependent setup times. *Computers and Industrial Engineering*, 36:163–177.

- Nowicki, E. y Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91:160–175.
- Nowicki, E. y Smutnicki, C. (1998). The flow shop with parallel machines: A tabu search approach. *European Journal of Operational Research*, 106:226–253.
- Nowicki, E. y Smutnicki, C. (1999). Flow Line Scheduling by Tabu Search. En Voß, S., Martello, S., Osman, I. H., y Roucairol, C., editores, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, páginas 175–189, Dordrecht. Kluwer Academic Publishers.
- Ochoa, G., Harvey, I., y Buxton, H. (1999). On Recombination and Optimal Mutation Rates. En *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-99)*, páginas 488–495, San Francisco. Morgan Kaufmann.
- Ogbu, F. A. y Smith, D. K. (1990a). Simulated Annealing for the Permutation Flowshop Problem. *OMEGA, The International Journal of Management Science*, 19(1):64–67.
- Ogbu, F. A. y Smith, D. K. (1990b). The Application of the Simulated Annealing Algorithm to the Solution of the $n/m/C_{max}$ Flowshop Problem. *Computers and Operations Research*, 17(3):243–253.
- Olhager, J. y Rapp, B. (1995). Operations Research Techniques in Manufacturing Planning and Control Systems. *International Transactions in Operational Research*, 2(1):29–43.
- Onwubolu, G. C. y Mutingi, M. (1999). Genetic algorithm for minimizing tardiness in flow-shop scheduling. *Production Planning and Control*, 10(5):462–471.
- Osman, I. H. y Potts, C. N. (1989). Simulated Annealing for Permutation Flowshop Scheduling. *OMEGA, The International Journal of Management Science*, 17(6):551–557.

- Page, E. S. (1961). An Approach to the Scheduling of Jobs on Machines. *Journal of the Royal Statistical Society, B Series*, 23(2):484–492.
- Palmer, D. S. (1965). Sequencing Jobs through a Multi-Stage Process in the Minimum Total Time - A Quick Method of Obtaining a Near Optimum. *Operational Research Quarterly*, 16(1):101–107.
- Panwalkar, S. S. y Iskander, W. (1977). A survey of scheduling rules. *Operations Research*, 25(1):45–61.
- Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.
- Parthasarathy, S. y Rajendran, C. (1997). An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs. *International Journal of Production Economics*, 49:255–263.
- Pinedo, M. (2002). *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, New Jersey, segunda edición.
- Ponnambalam, S. G., Aravindan, P., y Chandrasekaran, S. (2001). Constructive and improvement flow shop scheduling heuristics: an extensive evaluation. *Production Planning and Control*, 12(4):335–344.
- Portmann, M. C. (1997). Scheduling methodology: optimization and computer search approaches I. En Artiba, A. y Elmaghraby, S. E., editores, *The Planning and Scheduling of Production Systems. Methodologies and Applications*, London. Chapman & Hall.
- Portmann, M. C., Vignier, A., Dardilhac, D., y Dezelay, D. (1998). Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research*, 107:389–400.
- Potts, C. N. (1980). An adaptive branching rule for the permutation flow-shop problem. *European Journal of Operational Research*, 5:19–25.
- Potts, C. N., Shmoys, D. B., y Williamson, D. P. (1991). Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, 10:281–284.

- Proust, C., Gupta, J. N. D., y Deschamps, V. (1991). Flowshop scheduling with set-up, processing and removal times separated. *International Journal of Production Research*, 29(3):479–493.
- Rajendran, C. (1993). Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, 29:65–73.
- Rajendran, C. y Chaudhuri, D. (1991). An efficient heuristic approach to the scheduling of jobs in a flowshop. *European Journal of Operational Research*, 61:318–325.
- Reeves, C. R. (1993). Improving the Efficiency of Tabu Search for Machine Scheduling Problems. *Journal of the Operational Research Society*, 44(4):375–382.
- Reeves, C. R. (1995). A Genetic Algorithm for Flowshop Sequencing. *Computers and Operations Research*, 22(1):5–13.
- Reeves, C. R. y Höhn, C. (1996). Integrating Local Search into Genetic Algorithms. En Rayward-Smith, V. J., Osman, I. H., Reeves, C. R., y Smith, G. D., editores, *Modern Heuristic Search Methods*, páginas 99–115, New York. John Wiley & Sons.
- Reeves, C. R. y Yamada, T. (1998). Genetic Algorithms, Path Relinking, and the Flowshop Sequencing Problem. *Evolutionary Computation*, 6(1):45–60.
- Riane, F., Artiba, A., y Elmaghraby, S. E. (1998). A hybrid three-stage flowshop problem: Efficient heuristics to minimize makespan. *European Journal of Operational Research*, 109:321–329.
- Rinnooy Kan, A. H. G. (1976). *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Hague.
- Ríos-Mercado, R. Z. y Bard, J. (1998a). Computational Experience with a Branch-and-Cut Algorithm for Flowshop Scheduling with Setups. *Computers and Operations Research*, 25(5):351–366.

- Ríos-Mercado, R. Z. y Bard, J. (1998b). Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 110:76–98.
- Ríos-Mercado, R. Z. y Bard, J. (1999a). An enhanced TSP-Based Heuristic for Makespan Minimization in a Flow Shop with Setup Times. *Journal of Heuristics*, 5:53–70.
- Ríos-Mercado, R. Z. y Bard, J. F. (1999b). A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. *IIE Transactions*, 31:721–731.
- Rubin, P. A. y Ragatz, G. L. (1995). Scheduling in a Sequence Dependent Setup Environment with Genetic Search. *Computers and Operations Research*, 22(1):85–99.
- Salvador, M. S. (1973). A solution to a special case of flow shop scheduling problems. En Elmaghraby, S. E., editor, *Symposium of the Theory of Scheduling and Applications*, páginas 83–91, New York. Springer-Verlag.
- Santos, D. L., Hunsucker, J. L., y Deal, D. E. (1995). Global lower bounds for flow shops with multiple processors. *European Journal of Operational Research*, 80:112–120.
- Santos, D. L., Hunsucker, J. L., y Deal, D. E. (1996). An Evaluation of Sequencing Heuristics in Flow Shops With Multiple Processors. *Computers and Industrial Engineering*, 30(4):681–692.
- Sarin, S. y Lefoka, M. (1993). Scheduling Heuristic for the n -Job m -Machine Flow Shop. *OMEGA, The International Journal of Management Science*, 21(2):229–234.
- Schrage, L. (1998). *Optimization Modeling with LINGO*. Lindo Systems Inc., segunda edición.
- Selen, W. J. y Hott, D. D. (1986). A mixed-integer goal-programming formulation of the standard flow-shop scheduling problem. *Journal of the Operational Research Society*, 37:1121–1128.

- Sherali, H. D., Sarin, S. C., y Kodialam, M. S. (1990). Models and algorithms for a two-stage production process. *Production Planning and Control*, 1(1):27–39.
- Simons, Jr, J. V. (1992). Heuristics in Flow Shop Scheduling with Sequence Dependent Setup Times. *OMEGA, The international Journal of Management Science*, 20(2):215–225.
- Smith, J. y Fogarty, T. C. (1996). Self Adaptation of Mutation Rates in a Steady State Genetic Algorithm. En *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation*, páginas 318–323, Piscataway. IEEE Press.
- Soewnadi, H. y Elmaghraby, S. E. (2001). Sequencing three-stage flexible flowshops with identical machines to minimize makespan. *IIE Transactions*, 31:985–993.
- Sotskov, Y. N., Tautenhahn, T., y Werner, F. (1996). Heuristics for permutation flow shop scheduling with batch setup times. *OR Spektrum*, 18:67–80.
- Spears, W. M. y De Jong, K. A. (1991). On the Virtues of Parametrized Uniform Crossover. En Belew, R. y Booker, L., editores, *Proceedings of the Fourth International Conference on Genetic Algorithms and their Applications*, páginas 230–236, San Mateo. Morgan Kaufmann.
- Srikar, B. R. y Ghosh, S. (1986). A MILP model for the n -job, M -stage flowshop with sequence dependent set-up times. *International Journal of Production Research*, 24(6):1459–1474.
- Sriskandarajah, C. y Sethi, S. P. (1989). Scheduling algorithms for flexible flowshops: Worst and average case performance. *European Journal of Operational Research*, 43:143–160.
- Stafford, Jr, E. F. y Tseng, F. T. (1990). On the Srikanth-Ghosh MILP model for the $N \times M$ SDST flowshop problem. *International Journal of Production Research*, 28(10):1817–1830.

- Stafford, Jr, E. F. y Tseng, F. T. (2001). Two MILP Models for the SSIST Flowshop Sequencing Problem. En *Proceedings of the Decision Sciences Institute National Meeting*, páginas 1010–1012, San Francisco.
- Stafford, Jr, E. F. y Tseng, F. T. (2002). Two models for a family of flowshop sequencing problems. *European Journal of Operational Research*, 142:282–293.
- Stinson, J. P. y Smith, A. W. (1982). A heuristic programming procedure for sequencing the static flowshop. *International Journal of Production Research*, 20(6):753–764.
- Stützle, T. (1998). Applying Iterated Local Search to the Permutation Flow Shop Problem. Informe técnico, AIDA-98-04, FG Intellektik, TU Darmstadt.
- Sule, D. R. (1982). Sequencing n Jobs on Two Machines With Setup, Processing and Removal Times Separated. *Naval Research Logistics Quarterly*, 29(3):517–519.
- Sule, D. R. y Huang, K. Y. (1983). Sequency on two and three machines with setup, processing and removal times separated. *International Journal of Production Research*, 21(5):723–732.
- Suliman, S. M. A. (2000). A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics*, 64:143–152.
- Syswerda, G. (1996). Scheduling Optimization Using Genetic Algorithms. En Davis, L., editor, *Handbook of Genetic Algorithms*, páginas 332–349, London. International Thomson Computer Press.
- Szwarc, W. (1983). Flow Shop Problems with Time Lags. *Management Science*, 29(4):477–481.
- Szwarc, W. y Gupta, J. N. (1987). A flow-shop problem with sequence-dependent additive setup times. *Naval Research Logistics*, 34:619–627.

- Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47:67–74.
- Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285.
- Tang, L. y Liu, J. (2002). A modified genetic algorithm for the flow shop sequencing problem to minimize mean flow time. *Journal of Intelligent Manufacturing*, 13:61–67.
- Texeira, S. y Pacheco, X. (2002). *Borland Delphi™ 6 Developer's Guide*. SAMS, Indianapolis.
- Thomas, J. L. y McClain, J. O. (1993). An overview of production planning. En Graves, S. C., Rinnooy Kan, A. H. G., y Zipkin, P. H., editores, *Logistics of Production and Inventory*, volumen 4 de *Handbooks in Operations Research and Management Science*, páginas 333–370, Amsterdam. Elsevier Science Publishers, B. V.
- T'Kindt, V. y Billaut, J.-C. (2002). *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer-Verlag, Berlin.
- Tseng, F. T. y Stafford, Jr, E. F. (2001). Two MILP models for the $N \times M$ SDST flowshop sequencing problem. *International Journal of Production Research*, 39(8):1777–1809.
- Turner, S. y Booth, D. (1987). Comparison of Heuristics for Flow Shop Sequencing. *OMEGA, The International Journal of Management Science*, 15(1):75–78.
- Tuson, A. y Ross, P. (1996). Co-evolution of Operator Settings In Genetic Algorithms. En Fogarty, T. C., editor, *Evolutionary Computing, AISB Workshop*, volumen 1143 de *Lecture Notes in Computer Science*, páginas 286–296, Brighton. Springer.
- Uskup, E. y Smith, S. B. (1975). A Branch-and-Bound Algorithm for Two-Stage Production-Sequencing Problems. *Operations Research*, 23(1):118–136.

- Vallada, E., Maroto, C., Ruiz, R., y Segura, B. (2003a). Análisis del Sistema de Operaciones en Empresas del Sector Cerámico Español. Informe técnico, Universidad Politécnica de Valencia, Valencia, España, Grupo de Investigación Operativa GIO.
- Vallada, E., Maroto, C., Ruiz, R., y Segura, B. (2003b). Problemas de Programación de la Producción en el Sector Cerámico Español. Informe técnico, Universidad Politécnica de Valencia, Valencia, España, Grupo de Investigación Operativa GIO.
- Vallada, E., Ruiz, R., y Maroto, C. (2003). Synthetic and Real Benchmarks for Complex Flow-Shop Problems. Informe técnico, Universidad Politécnica de Valencia, Valencia, España, Grupo de Investigación Operativa GIO.
- Vavak, F. y Fogarty, T. C. (1996). A Comparative Study of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments. En *Proceedings of the Society for the Study of Artificial Intelligence and Simulation of Behaviour Workshop on Evolutionary Computing*, páginas 301–307.
- Vignier, A., Billaut, J. C., y Proust, C. (1999). Les Problèmes D’Ordonnancement de Type Flow-Shop Hybride: État de L’Art. *RAIRO Recherche opérationnelle*, 33(2):117–183.
- Wagner, H. M. (1959). An Integer Linear-Programming Model for Machine Scheduling. *Naval Research Logistics Quarterly*, 6:131–140.
- Watson, J.-P., Barbulescu, L., Whitley, Darrell, L., y Howe, A. E. (2002). Contrasting Structured and Random Permutation Flow-Shop Scheduling Problems: Search-Space Topology and Algorithm Performance. *INFORMS Journal on Computing*, 14(2):98–123.
- Watson, J.-P., Barbulescu, P., Howe, A. E., y Whitley, D. L. (1999). Algorithm Performance and Problem Structure for Flow-shop Scheduling. En Hendler, J. y Kautz, H., editores, *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, páginas 688–695, Orlando. AAAI Press/The MIT Press.

- Werner, F. (1993). On the Heuristic Solution of the Permutation Flow Shop Problem by Path Algorithms. *Computers and Operations Research*, 20(7):707–722.
- White, C. H. y Wilson, R. C. (1977). Sequence dependent set-up times and job sequencing. *International Journal of Production Research*, 15(2):191–202.
- Widmer, M. y Hertz, A. (1989). A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41:186–193.
- Wilbrecht, J. K. y Prescott, W. B. (1969). The Influence of Setup Time on Job Shop Performance. *Management Science*, 16(4):274–280.
- Wittrock, R. J. (1985). Scheduling algorithms for flexible flow lines. *IBM Journal of Research and Development*, 29(4):401–412.
- Wittrock, R. J. (1988). An adaptable scheduling algorithm for flexible flow lines. *Operations Research*, 36(3):445–453.
- Wodecki, M. y Bozejko, W. (2002). Solving the Flow Shop Problem by Parallel Simulated Annealing. En Wyrzykowski, R., Dongarra, J., Paprzycki, M., y Waśniewski, J., editores, *Parallel Processing and Applied Mathematics, 4th International Conference, PPAM 2001*, volumen 2328 de *Lecture Notes in Computer Science*, páginas 236–244. Springer-Verlag.
- Woo, H.-S. y Yim, D.-S. (1998). A Heuristic Algorithm for Mean Flowtime Objective in Flowshop Scheduling. *Computers and Operations Research*, 25(3):175–182.
- Yamada, T. y Reeves, C. R. (1997). Permutation Flowshop Scheduling by Genetic Local Search. páginas 232–238, Glasgow. GALESIA '97 IEE Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications.
- Yang, W.-H. y Liao, C.-J. (1999). Survey of scheduling research involving setup times. *International Journal of Systems Science*, 30(2):143–155.

- Yoshida, T. y Hitomi, K. (1979). Optimal Two-Stage Production Scheduling with Setup Times Separated. *AIIE Transactions*, 11(3):261–263.
- Zegordi, S. H., Itoh, K., y Enkawa, T. (1995). Minimizing makespan for flowshop scheduling by combining simulated annealing with sequencing knowledge. *European Journal of Operational Research*, 85:515–531.

ANEXO



TESTS ESTADÍSTICOS

A.1. Tests de comprobación de la hipótesis de homocedasticidad (ANOVA Capítulo 4)

En esta sección mostramos el resto de gráficos de residuos frente a los niveles o variantes de los factores del ANOVA que se realizó en la Sección 4.3.6.

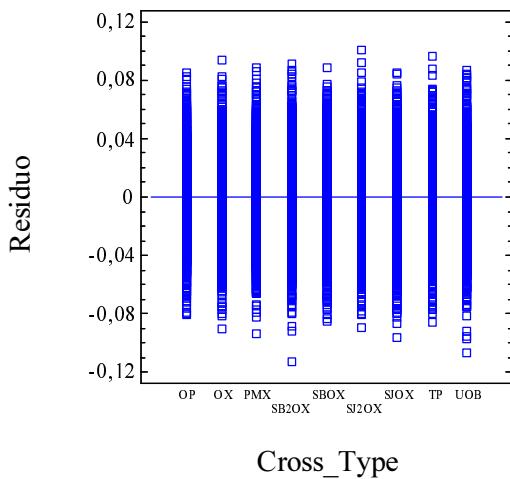


Figura A.1 – Gráfico de los residuos frente a los niveles del factor Cross_Type.

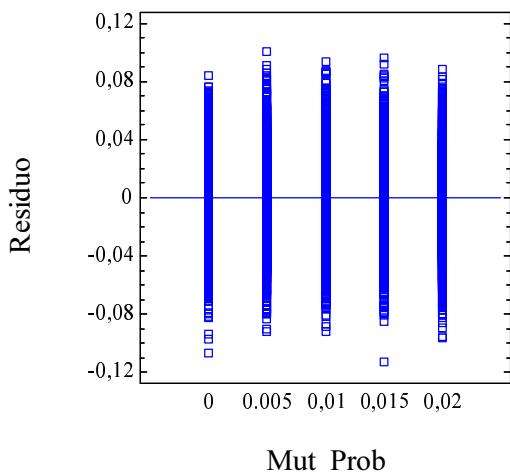


Figura A.2 – Gráfico de los residuos frente a los niveles del factor Mut_Prob.

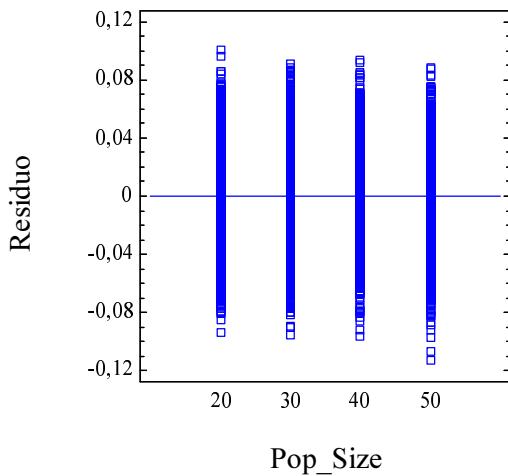


Figura A.3 – Gráfico de los residuos frente a los niveles del factor Pop_Size.

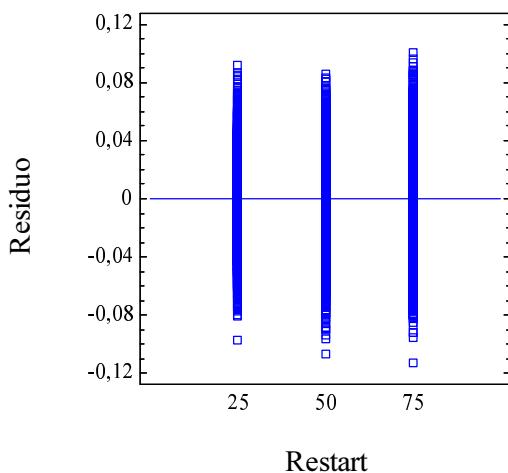


Figura A.4 – Gráfico de los residuos frente a los niveles del factor Restart.

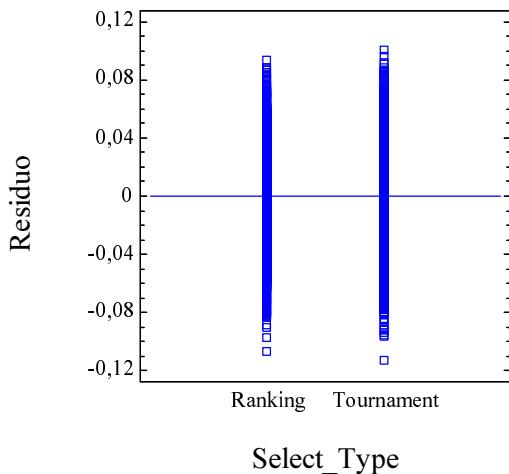


Figura A.5 – Gráfico de los residuos frente a los niveles del factor `Select_Type`.

A.2. Experimentos y tablas ANOVA, Capítulo 5

A.2.1. Tests de comprobación de la hipótesis de homocedasticidad, experimento SSD10

En esta sección se detallan los gráficos de residuos frente a los niveles o variantes de los factores del ANOVA que se realizó en la Sección 5.5.4.

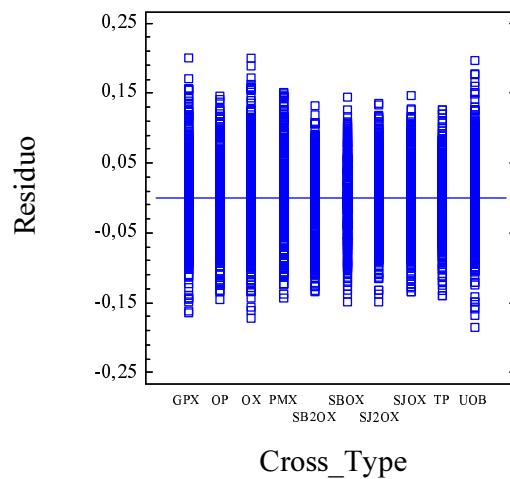


Figura A.6 – Gráfico de los residuos frente a los niveles del factor *Cross_Type*, experimento SSD10.

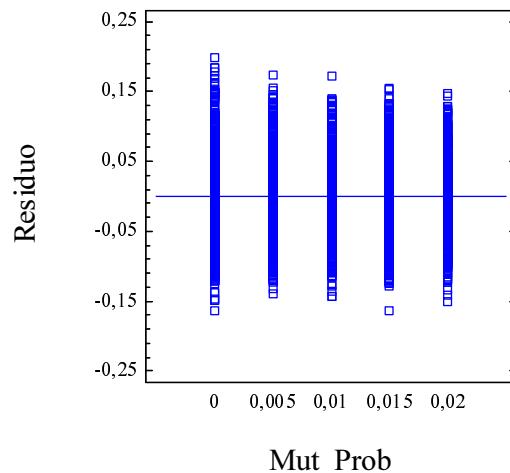


Figura A.7 – Gráfico de los residuos frente a los niveles del factor *Mut_Prob*, experimento SSD10.

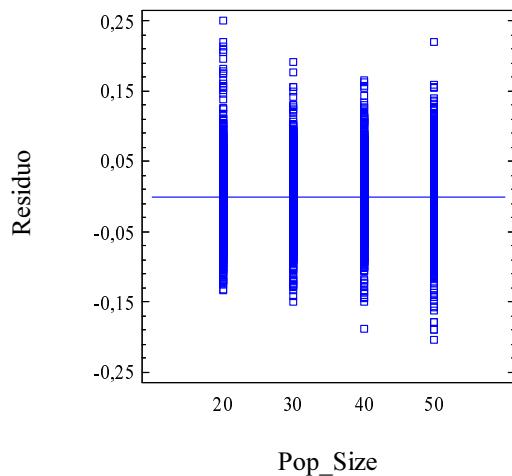


Figura A.8 – Gráfico de los residuos frente a los niveles del factor Pop_Size, experimento SSD10.

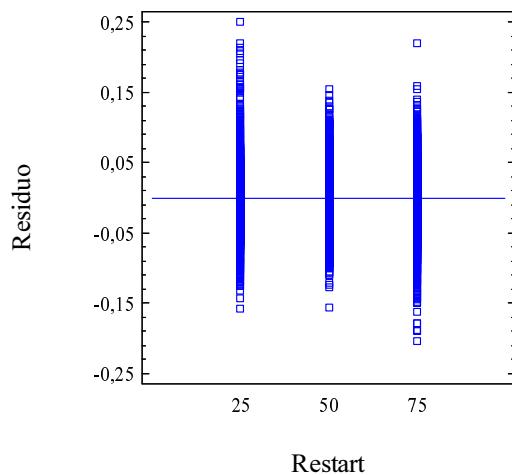


Figura A.9 – Gráfico de los residuos frente a los niveles del factor Restart, experimento SSD10.

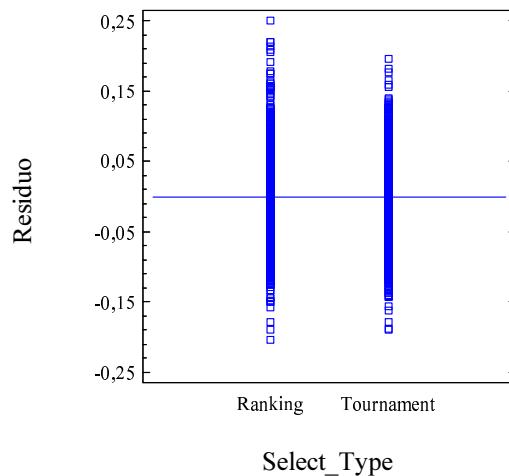


Figura A.10 – Gráfico de los residuos frente a los niveles del factor `Select_Type`, experimento SSD10.

A.2.2. Experimento SSD50

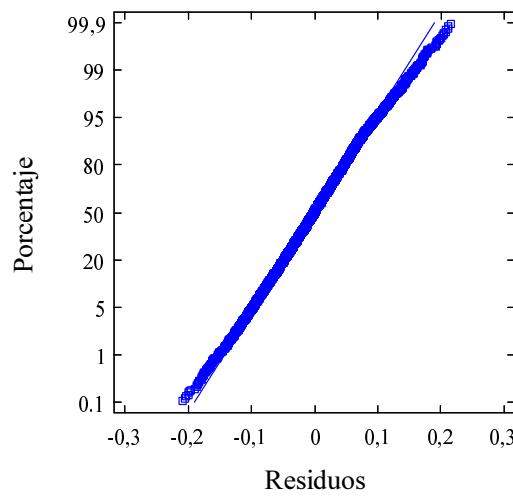


Figura A.11 – Gráfico de probabilidad Normal, experimento SSD50.

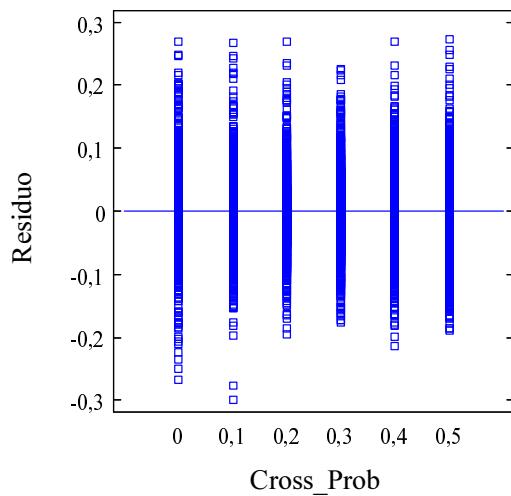


Figura A.12 – Gráfico de los residuos frente a los niveles del factor Cross_Prob, experimento SSD50.

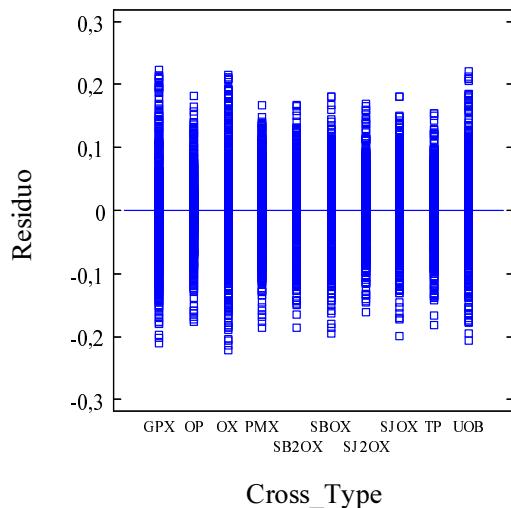


Figura A.13 – Gráfico de los residuos frente a los niveles del factor Cross_Type, experimento SSD50.

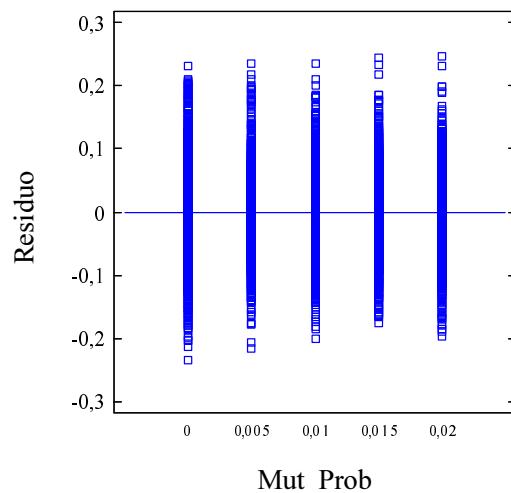


Figura A.14 – Gráfico de los residuos frente a los niveles del factor Mut_Prob, experimento SSD50.

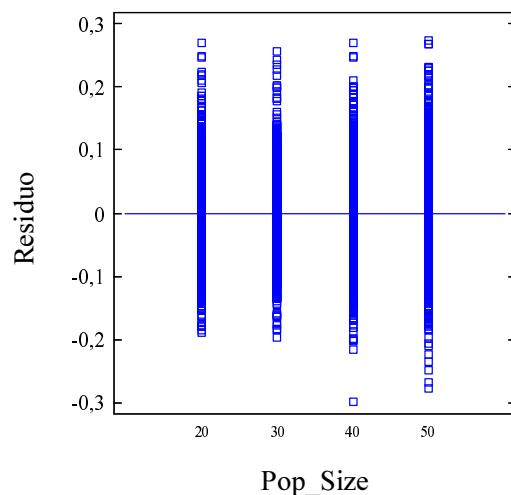


Figura A.15 – Gráfico de los residuos frente a los niveles del factor Pop_Size, experimento SSD50.

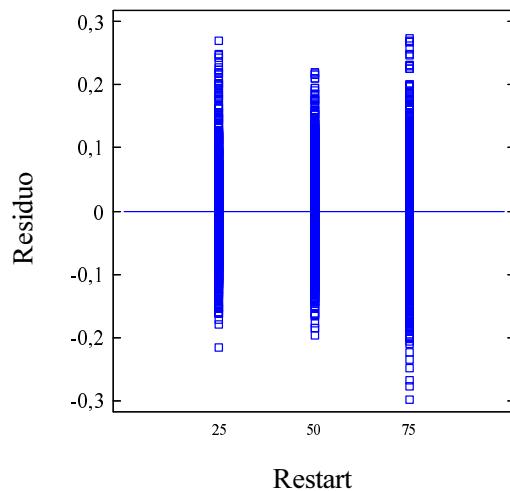


Figura A.16 – Gráfico de los residuos frente a los niveles del factor `Restart`, experimento SSD50.

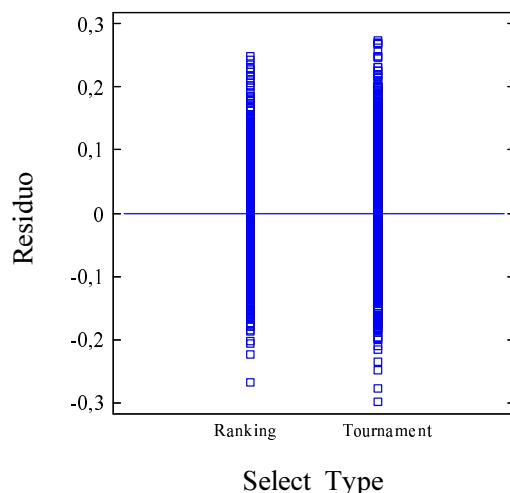


Figura A.17 – Gráfico de los residuos frente a los niveles del factor `Select_Type`, experimento SSD50.

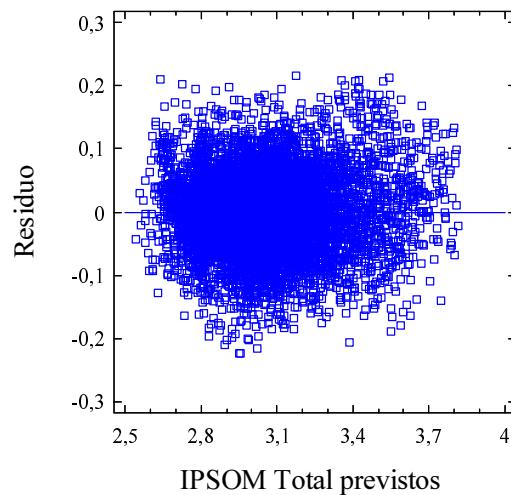


Figura A.18 – Gráfico de los residuos frente los valores previstos de *IP SOM* total, experimento SSD50.

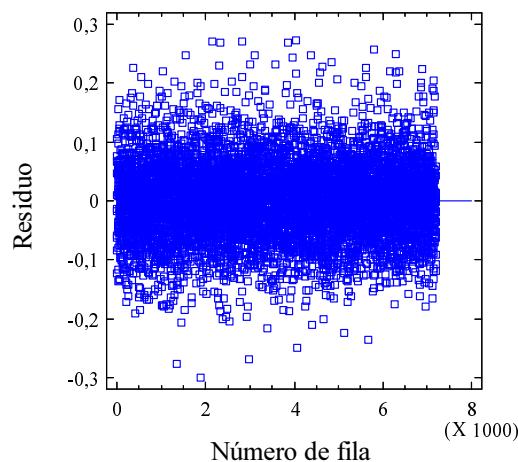


Figura A.19 – Gráfico de residuos frente al orden de ejecución de las pruebas, experimento SSD50.

Analysis of Variance for IPSOM Total - Type III Sums of Squares

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	9,24202	5	1,8484	426,34	0,0000
B:Cross_Type	106,686	9	11,854	2734,12	0,0000
C:Mut_Prob	53,4743	4	13,3686	3083,47	0,0000
D:Pop_Size	4,35847	3	1,45282	335,09	0,0000
E:Restart	53,8229	2	26,9114	6207,13	0,0000
F:Select_Type	38,1973	1	38,1973	8810,23	0,0000
INTERACTIONS					
AB	12,7896	45	0,284213	65,55	0,0000
AC	26,6816	20	1,33408	307,71	0,0000
AD	0,0781015	15	0,00520677	1,20	0,2623
AE	1,76098	10	0,176098	40,62	0,0000
AF	0,968849	5	0,19377	44,69	0,0000
BC	8,18017	36	0,227227	52,41	0,0000
BD	8,29145	27	0,307091	70,83	0,0000
BE	14,1192	18	0,784402	180,92	0,0000
BF	4,41692	9	0,490769	113,20	0,0000
CD	1,39544	12	0,116287	26,82	0,0000
CE	0,562675	8	0,0703344	16,22	0,0000
CF	4,00558	4	1,0014	230,97	0,0000
DE	0,8196	6	0,1366	31,51	0,0000
DF	0,209208	3	0,0697358	16,08	0,0000
EF	0,671299	2	0,335649	77,42	0,0000
RESIDUAL	30,1539	6955	0,00433557		
TOTAL (CORRECTED)	380,885	7199			

All F-ratios are based on the residual mean square error.

Tabla A.1 – Tabla ANOVA con los resultados del experimento para el algoritmo GA_{sd} , conjunto de datos SSD50.

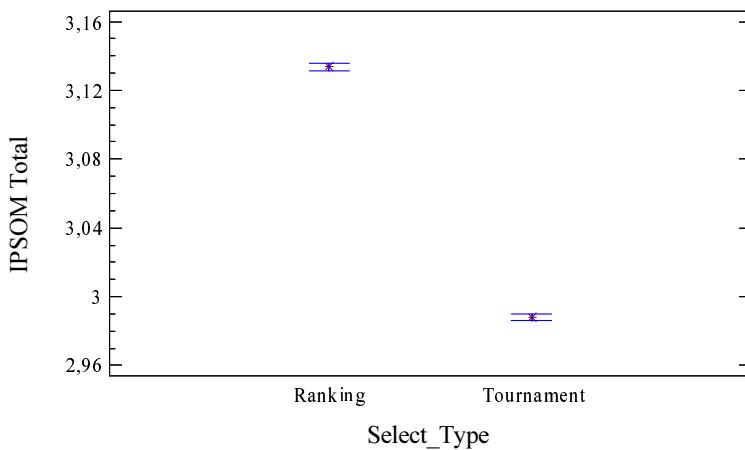


Figura A.20 – Gráfico de medias para el factor Select_Type, experimento SSD50.

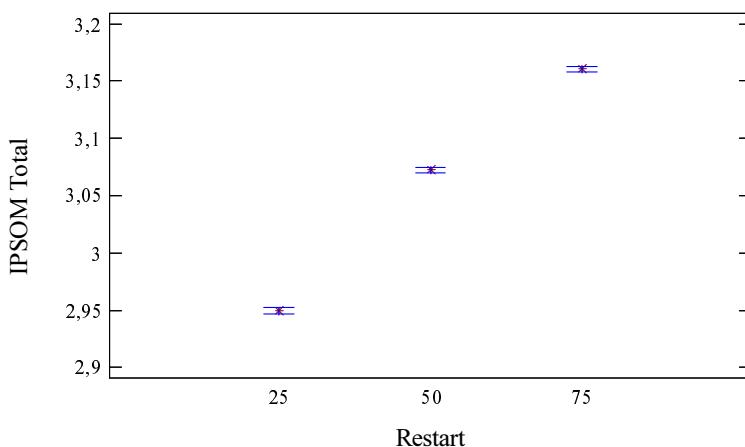


Figura A.21 – Gráfico de medias para el factor Restart, experimento SSD50.

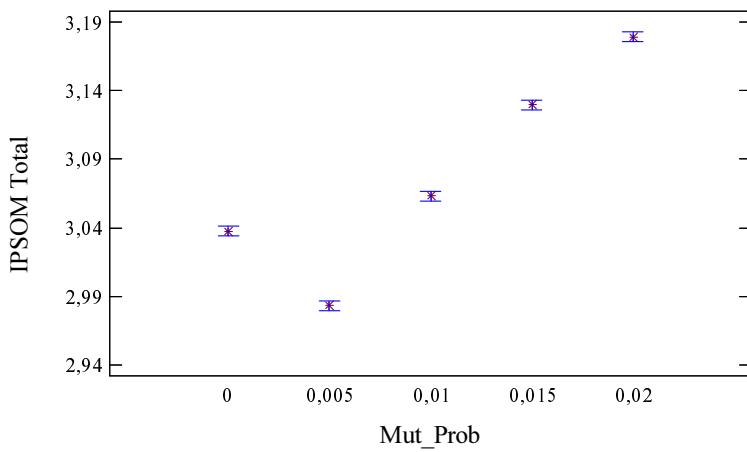


Figura A.22 – Gráfico de medias para el factor Mut_Prob, experimento SSD50.

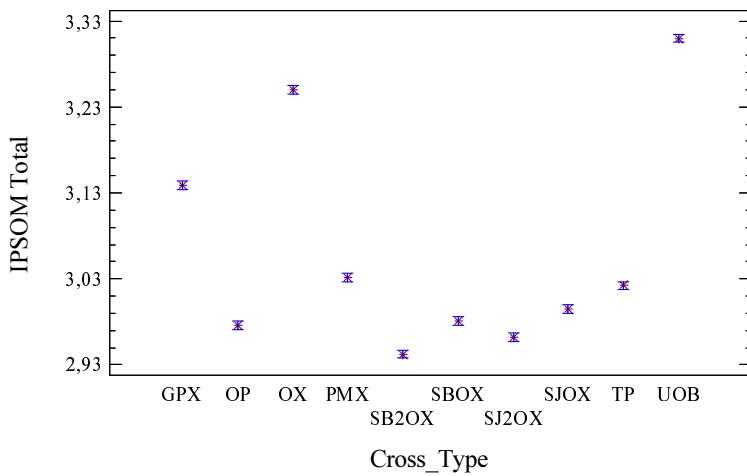


Figura A.23 – Gráfico de medias para el factor Cross_Type, experimento SSD50.

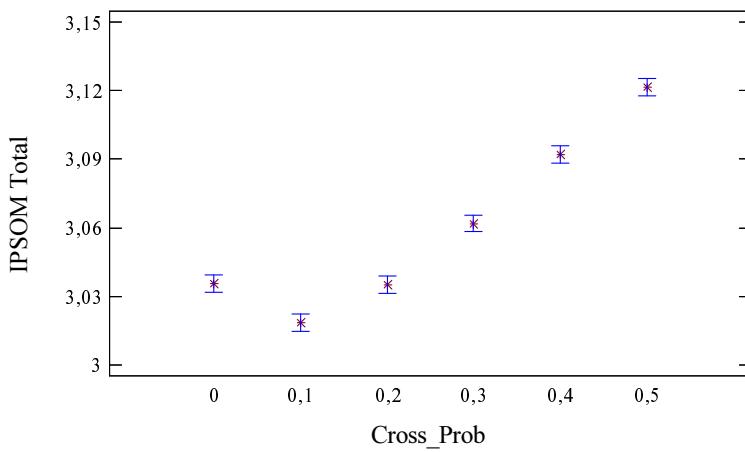


Figura A.24 – Gráfico de medias para el factor Cross_Prob, experimento SSD50.

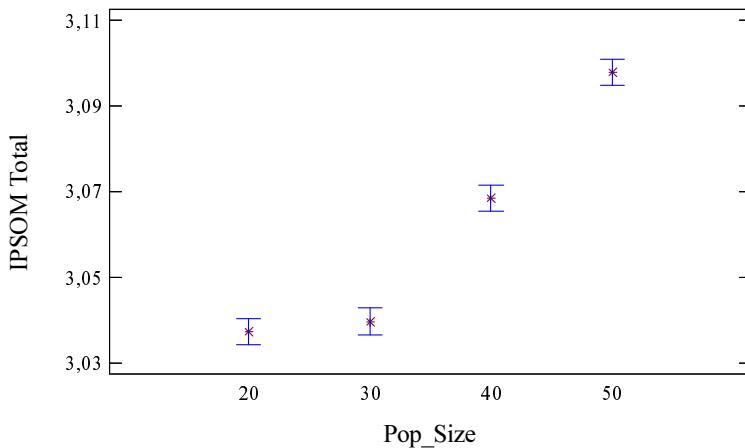


Figura A.25 – Gráfico de medias para el factor Pop_Size, experimento SSD50.

A.2.3. Experimento SSD100

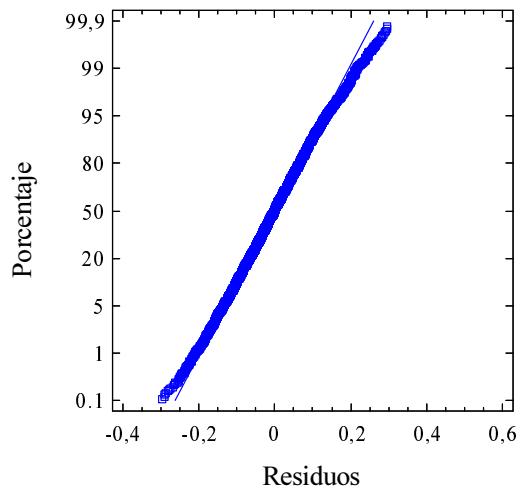


Figura A.26 – Gráfico de probabilidad Normal, experimento SSD100.

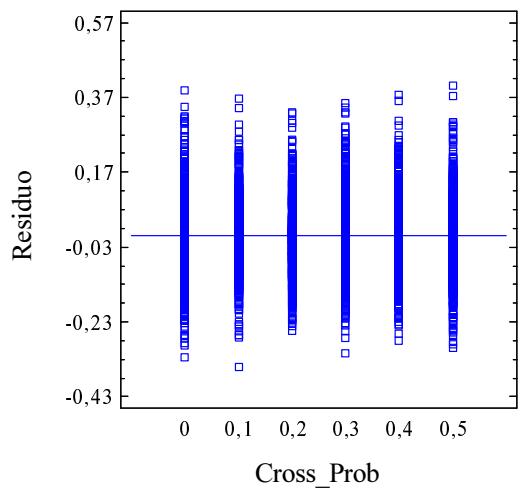


Figura A.27 – Gráfico de los residuos frente a los niveles del factor Cross_Prob, experimento SSD100.

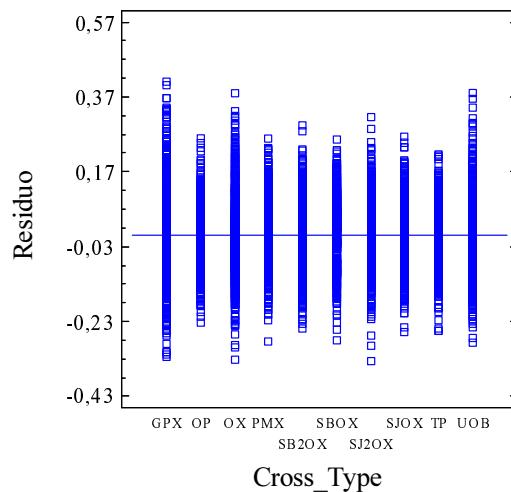


Figura A.28 – Gráfico de los residuos frente a los niveles del factor `Cross_Type`, experimento SSD100.

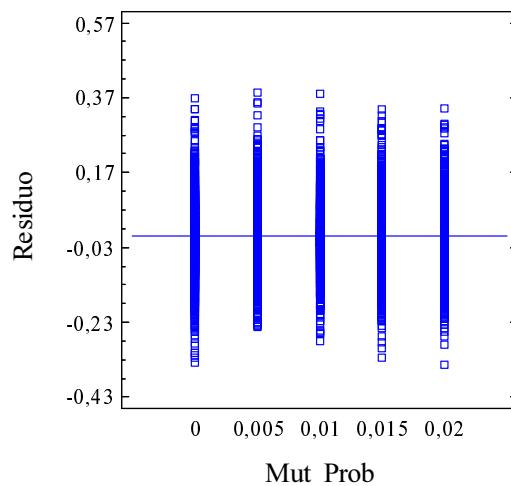


Figura A.29 – Gráfico de los residuos frente a los niveles del factor `Mut_Prob`, experimento SSD100.

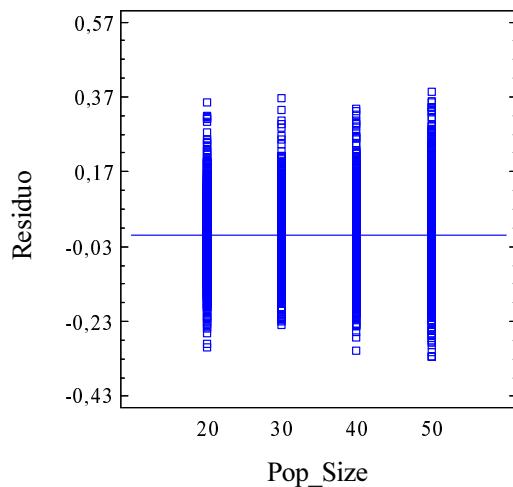


Figura A.30 – Gráfico de los residuos frente a los niveles del factor Pop_Size, experimento SSD100.

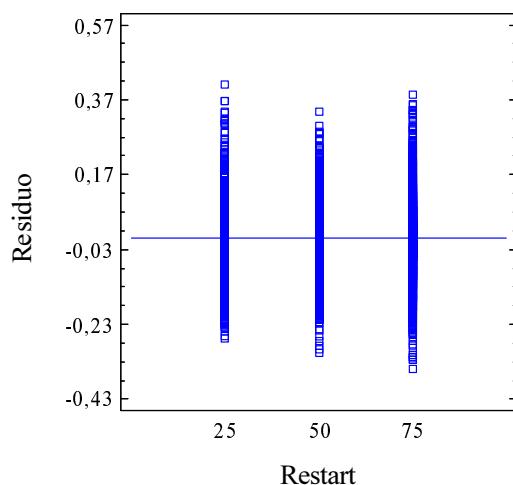


Figura A.31 – Gráfico de los residuos frente a los niveles del factor Restart, experimento SSD100.

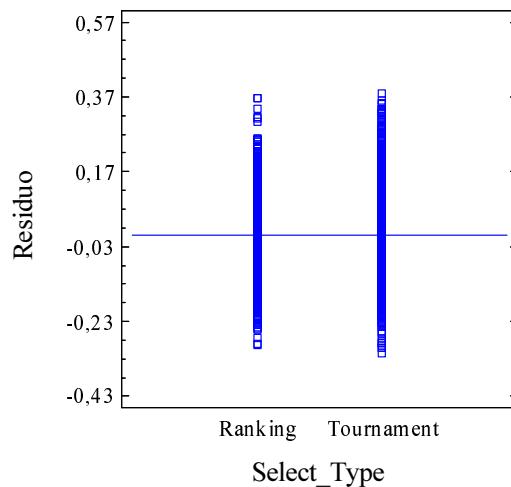


Figura A.32 – Gráfico de los residuos frente a los niveles del factor `Select_Type`, experimento SSD100.

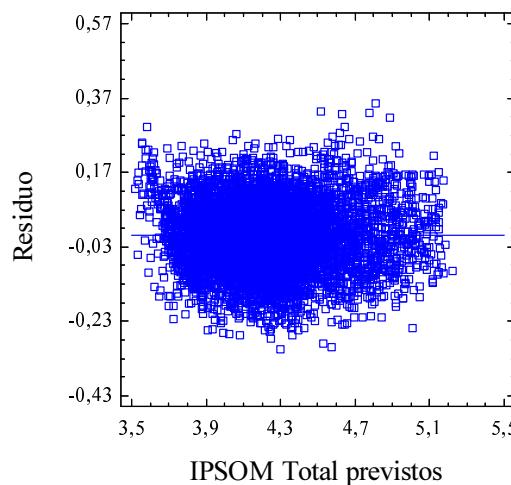


Figura A.33 – Gráfico de los residuos frente los valores previstos de *IP SOM* total, experimento SSD100.

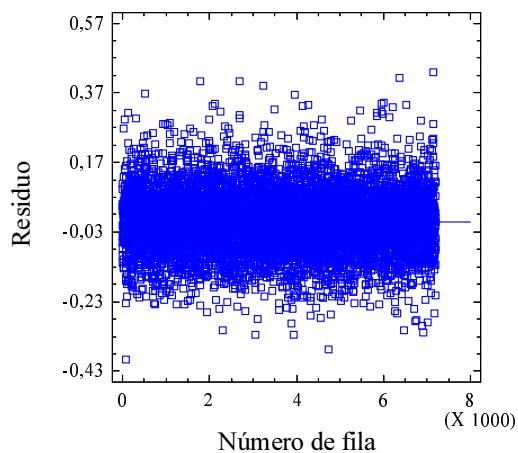


Figura A.34 – Gráfico de residuos frente al orden de ejecución de las pruebas, experimento SSD100.

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	28,6887	5	5,73774	686,02	0,0000
B:Cross_Type	185,284	9	20,5871	2461,45	0,0000
C:Mut_Prob	88,2083	4	22,0521	2636,61	0,0000
D:Pop_Size	6,7599	3	2,2533	269,41	0,0000
E:Restart	107,832	2	53,9162	6446,37	0,0000
F:Select_Type	59,9949	1	59,9949	7173,16	0,0000
INTERACTIONS					
AB	25,2984	45	0,562187	67,22	0,0000
AC	42,9918	20	2,14959	257,01	0,0000
AD	0,072495	15	0,004833	0,58	0,8941
AE	3,54777	10	0,354777	42,42	0,0000
AF	1,78184	5	0,356368	42,61	0,0000
BC	15,95	36	0,443055	52,97	0,0000
BD	13,0639	27	0,483849	57,85	0,0000
BE	25,3392	18	1,40773	168,31	0,0000
BF	7,18052	9	0,797836	95,39	0,0000
CD	2,31343	12	0,192786	23,05	0,0000
CE	1,71189	8	0,213987	25,58	0,0000
CF	5,63968	4	1,40992	168,57	0,0000
DE	1,53099	6	0,255164	30,51	0,0000
DF	0,107806	3	0,0359352	4,30	0,0049
EF	1,64975	2	0,824874	98,62	0,0000
RESIDUAL	58,1702	6955	0,0083638		
TOTAL (CORRECTED)	683,118	7199			

All F-ratios are based on the residual mean square error.

Tabla A.2 – Tabla ANOVA con los resultados del experimento para el algoritmo GA_{sd} , conjunto de datos SSD100.

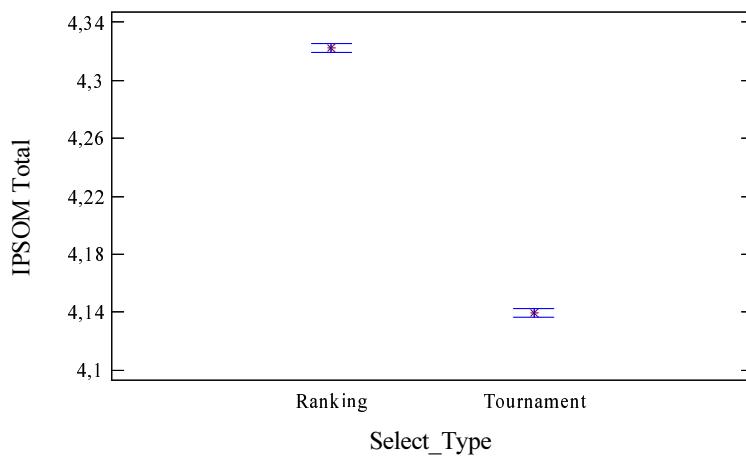


Figura A.35 – Gráfico de medias para el factor Select_Type, experimento SSD100.

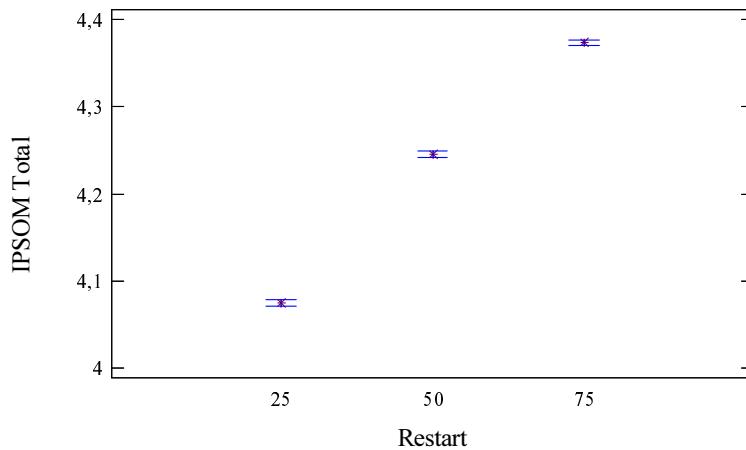


Figura A.36 – Gráfico de medias para el factor Restart, experimento SSD100.

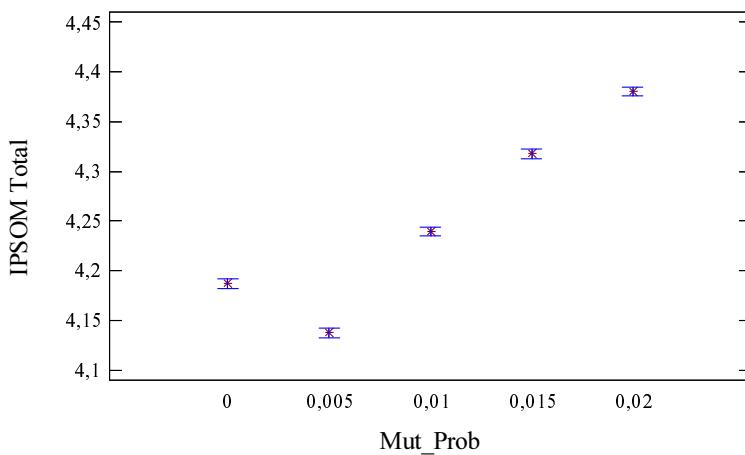


Figura A.37 – Gráfico de medias para el factor Mut_Prob, experimento SSD100.

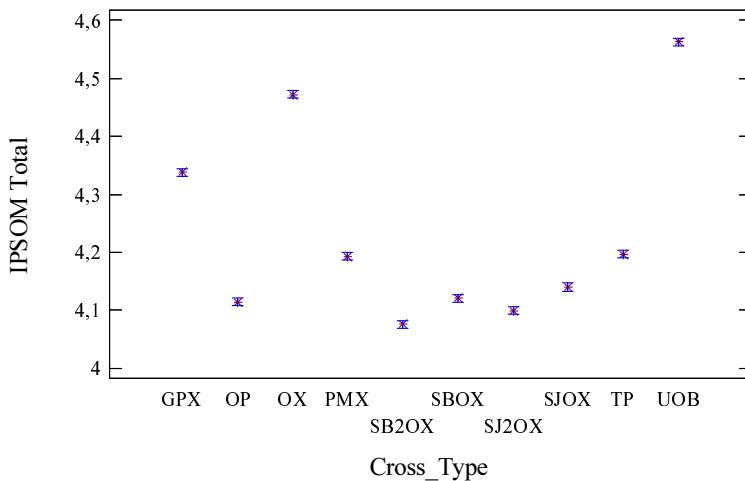


Figura A.38 – Gráfico de medias para el factor Cross_Type, experimento SSD100.

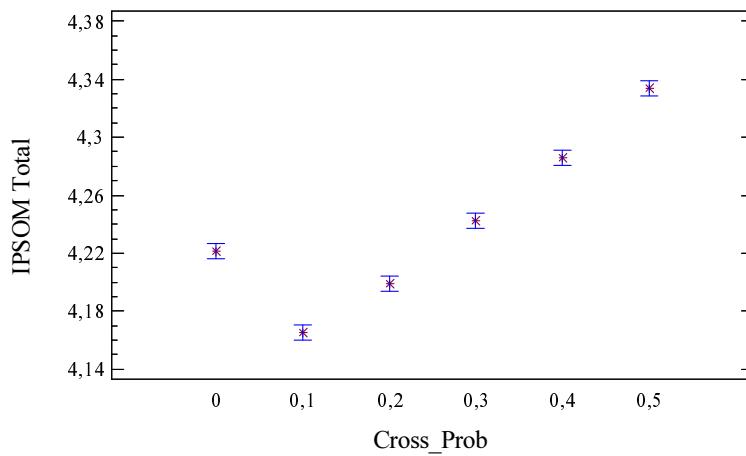


Figura A.39 – Gráfico de medias para el factor Cross_Prob, experimento SSD100.

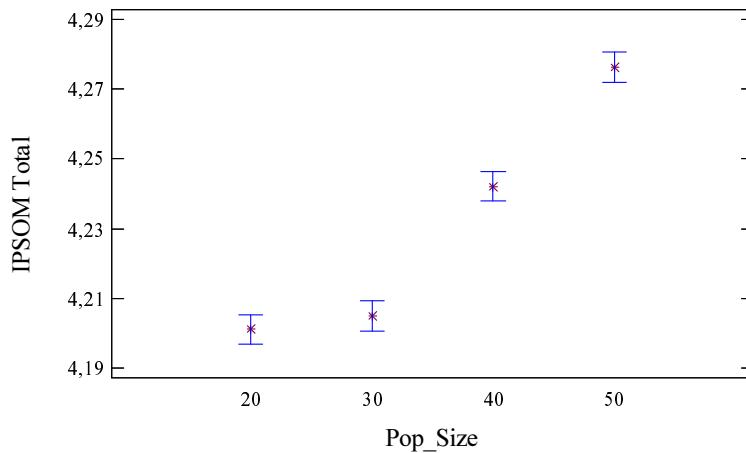


Figura A.40 – Gráfico de medias para el factor Pop_Size, experimento SSD100.

A.2.4. Experimento SSD125

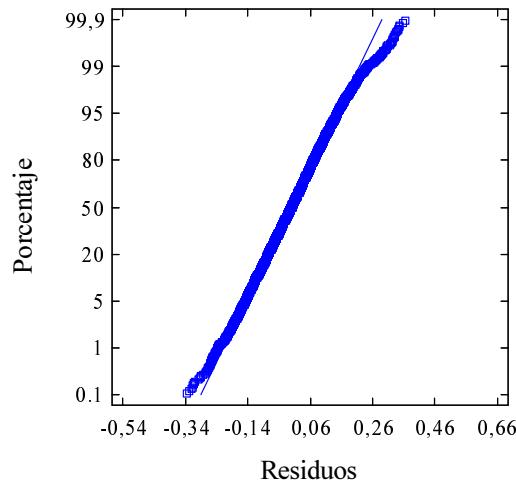


Figura A.41 – Gráfico de probabilidad Normal, experimento SSD125.

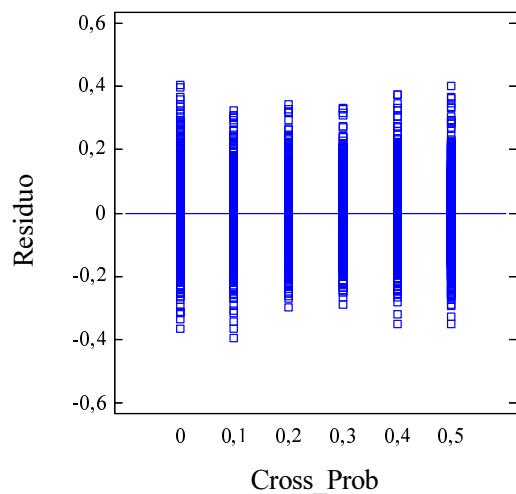


Figura A.42 – Gráfico de los residuos frente a los niveles del factor Cross_Prob, experimento SSD125.

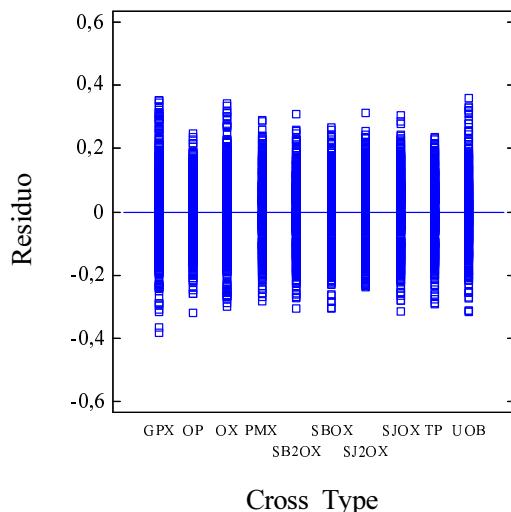


Figura A.43 – Gráfico de los residuos frente a los niveles del factor *Cross_Type*, experimento SSD125.

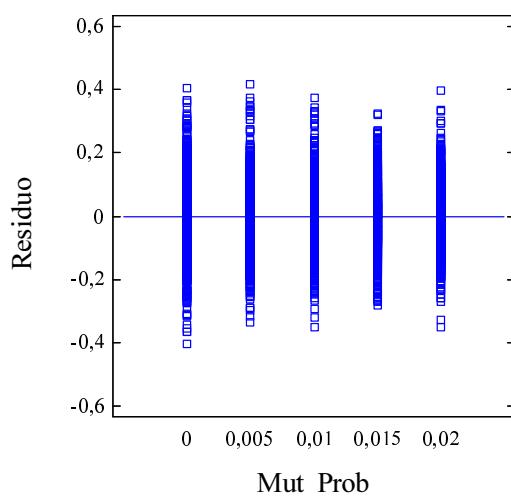


Figura A.44 – Gráfico de los residuos frente a los niveles del factor *Mut_Prob*, experimento SSD125.

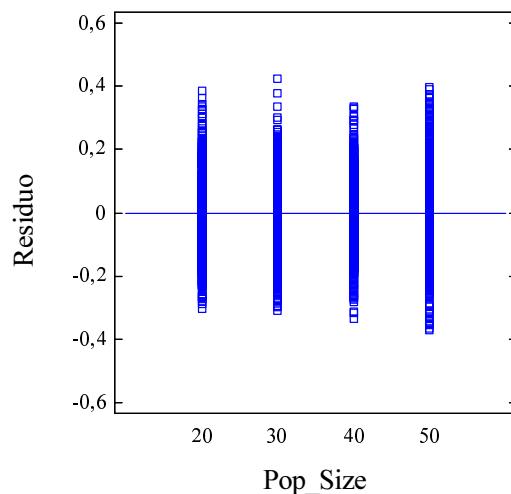


Figura A.45 – Gráfico de los residuos frente a los niveles del factor Pop_Size, experimento SSD125.

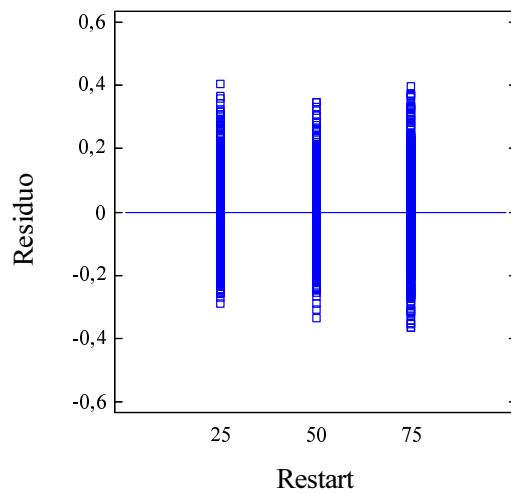


Figura A.46 – Gráfico de los residuos frente a los niveles del factor Restart, experimento SSD125.

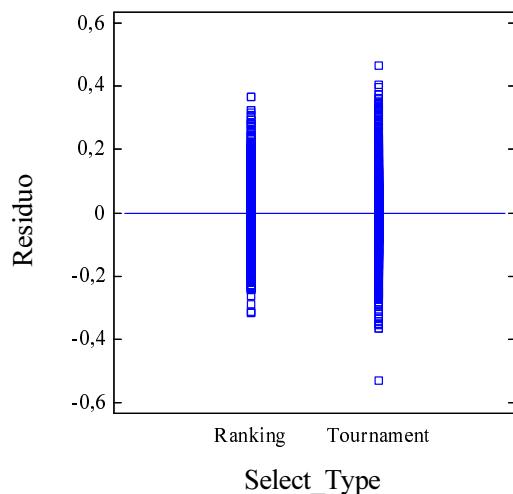


Figura A.47 – Gráfico de los residuos frente a los niveles del factor *Select_Type*, experimento SSD125.

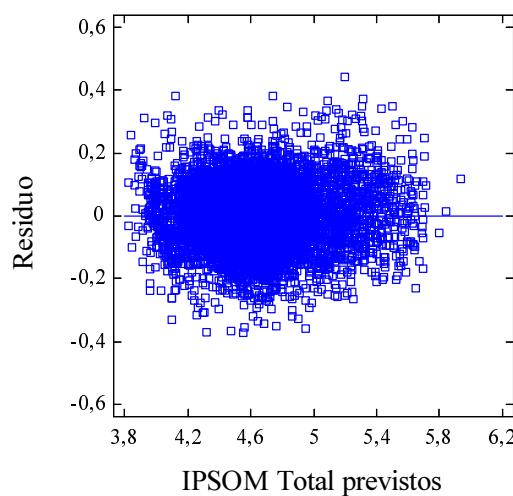


Figura A.48 – Gráfico de los residuos frente los valores previstos de *IPSON* total, experimento SSD125.

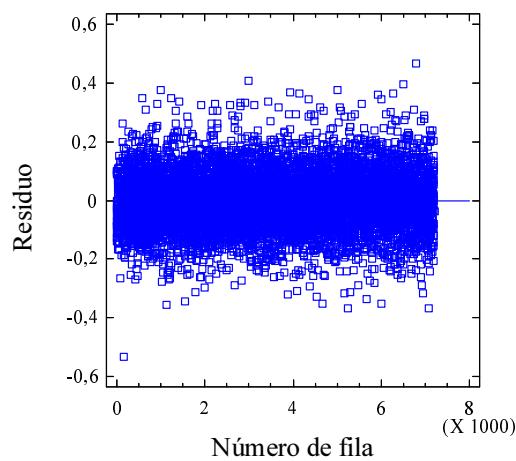


Figura A.49 – Gráfico de residuos frente al orden de ejecución de las pruebas, experimento SSD125.

Analysis of Variance for IPSOM Total - Type III Sums of Squares

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	46,1944	5	9,23887	935,66	0,0000
B:Cross_Type	198,669	9	22,0744	2235,57	0,0000
C:Mut_Prob	122,238	4	30,5595	3094,90	0,0000
D:Pop_Size	7,9723	3	2,65743	269,13	0,0000
E:Restart	143,73	2	71,865	7278,09	0,0000
F:Select_Type	69,0158	1	69,0158	6989,54	0,0000
INTERACTIONS					
AB	25,7618	45	0,572484	57,98	0,0000
AC	65,7659	20	3,28829	333,02	0,0000
AD	0,242548	15	0,0161699	1,64	0,0564
AE	3,29628	10	0,329628	33,38	0,0000
AF	1,44195	5	0,288389	29,21	0,0000
BC	16,7303	36	0,46473	47,07	0,0000
BD	15,5163	27	0,574679	58,20	0,0000
BE	28,1726	18	1,56515	158,51	0,0000
BF	8,26141	9	0,917934	92,96	0,0000
CD	2,59043	12	0,215869	21,86	0,0000
CE	1,77608	8	0,22201	22,48	0,0000
CF	6,67763	4	1,66941	169,07	0,0000
DE	1,91963	6	0,319938	32,40	0,0000
DF	0,207322	3	0,0691074	7,00	0,0001
EF	1,51129	2	0,755647	76,53	0,0000
RESIDUAL	68,6747	6955	0,00987415		
TOTAL (CORRECTED)	836,366	7199			

All F-ratios are based on the residual mean square error.

Tabla A.3 – Tabla ANOVA con los resultados del experimento para el algoritmo GA_{sd}, conjunto de datos SSD125.

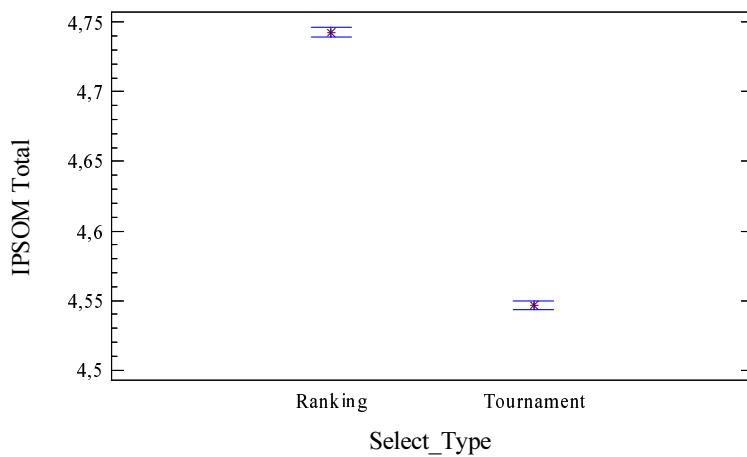


Figura A.50 – Gráfico de medias para el factor Select_Type, experimento SSD125.

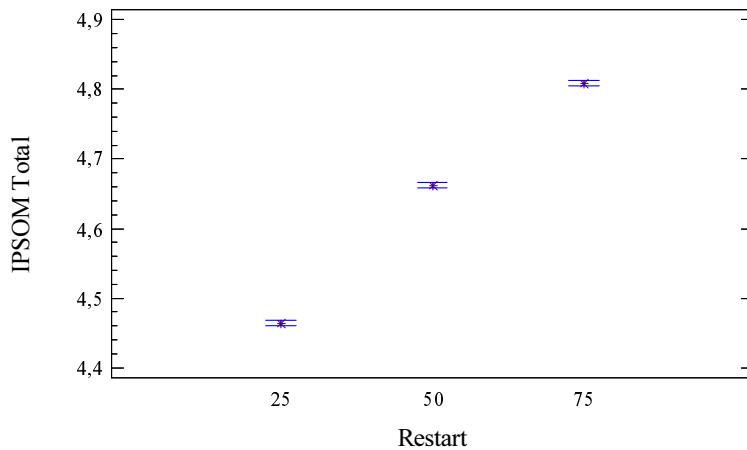


Figura A.51 – Gráfico de medias para el factor Restart, experimento SSD125.

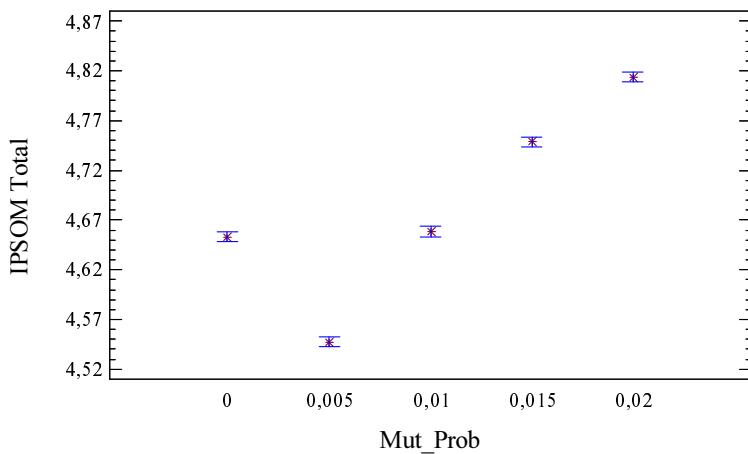


Figura A.52 – Gráfico de medias para el factor Mut_Prob, experimento SSD125.

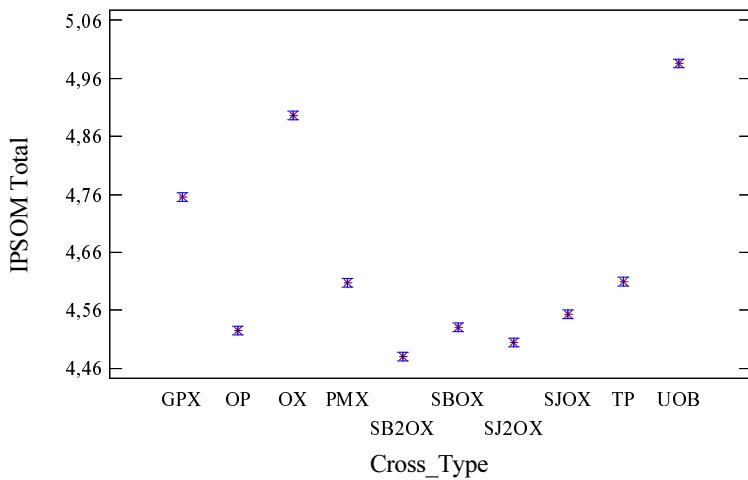


Figura A.53 – Gráfico de medias para el factor Cross_Type, experimento SSD125.

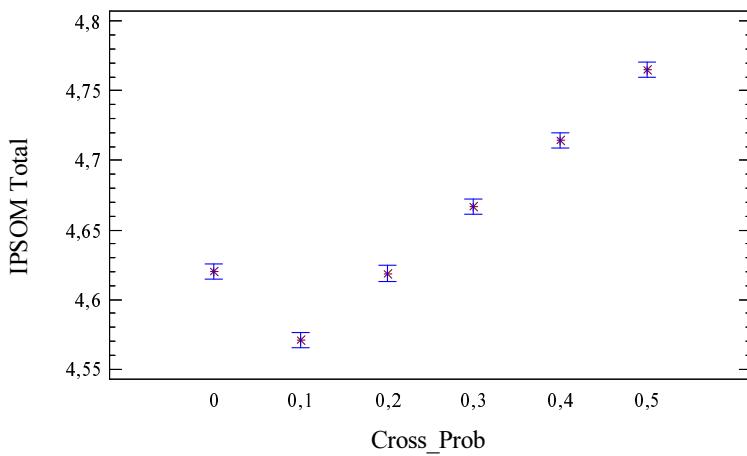


Figura A.54 – Gráfico de medias para el factor *Cross_Prob*, experimento SSD125.

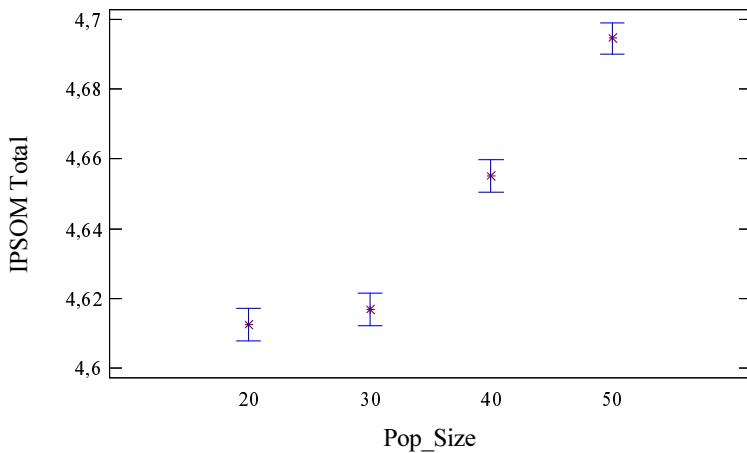


Figura A.55 – Gráfico de medias para el factor *Pop_Size*, experimento SSD125.

A.3. Experimentos y tablas ANOVA, Capítulo 6

A.3.1. Experimento SSD50_P13

Analysis of Variance for IPSOM Total - Type III Sums of Squares

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	25,4064	5	5,08128	489,51	0,0000
B:Cross_Type	50,6743	8	6,33429	610,22	0,0000
C:Mut_Prob	358,044	4	89,5111	8623,17	0,0000
D:Pop_Size	15,471	3	5,15699	496,81	0,0000
E:Restart	3,2392	2	1,6196	156,03	0,0000
F:Select_Type	292,244	1	292,244	28153,67	0,0000
INTERACTIONS					
AB	13,5947	40	0,339868	32,74	0,0000
AC	12,6473	20	0,632367	60,92	0,0000
AD	0,30353	15	0,0202353	1,95	0,0151
AE	2,27608	10	0,227608	21,93	0,0000
AF	4,00956	5	0,801911	77,25	0,0000
BC	15,3096	32	0,478424	46,09	0,0000
BD	0,340135	24	0,0141723	1,37	0,1097
BE	0,292377	16	0,0182736	1,76	0,0305
BF	2,54367	8	0,317959	30,63	0,0000
CD	2,8014	12	0,23345	22,49	0,0000
CE	11,7344	8	1,46681	141,31	0,0000
CF	28,6279	4	7,15697	689,48	0,0000
DE	1,72693	6	0,287822	27,73	0,0000
DF	11,8423	3	3,94745	380,28	0,0000
EF	1,77211	2	0,886056	85,36	0,0000
RESIDUAL	64,8873	6251	0,0103803		
TOTAL (CORRECTED)	919,788	6479			

All F-ratios are based on the residual mean square error.

Tabla A.4 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD50_P13.

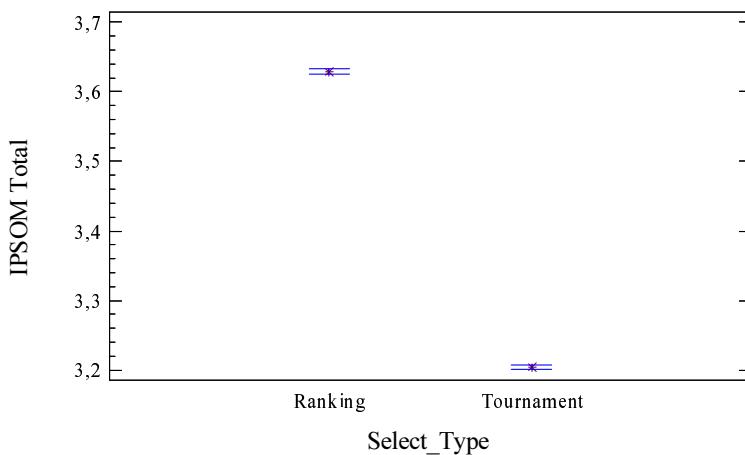


Figura A.56 – Gráfico de medias para el factor Select_Type, experimento SSD50_P13.

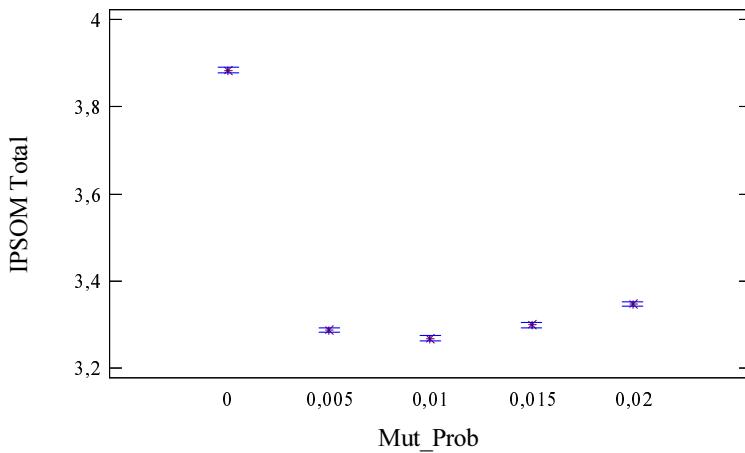


Figura A.57 – Gráfico de medias para el factor Mut_Prob, experimento SSD50_P13.

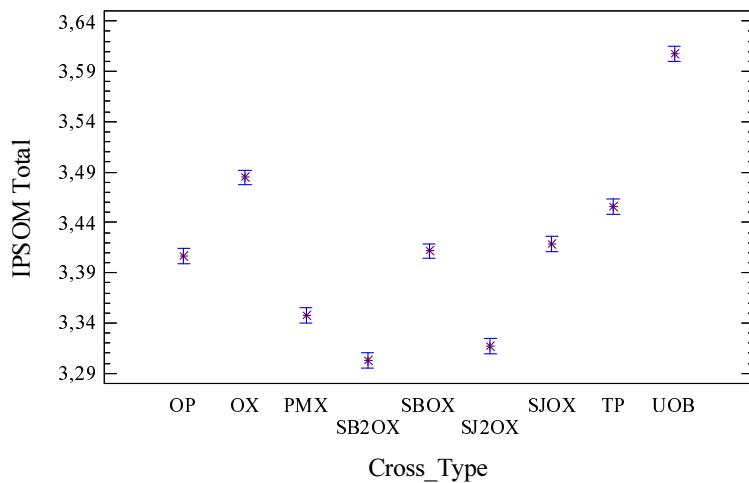


Figura A.58 – Gráfico de medias para el factor Cross_Type, experimento SSD50_P13.

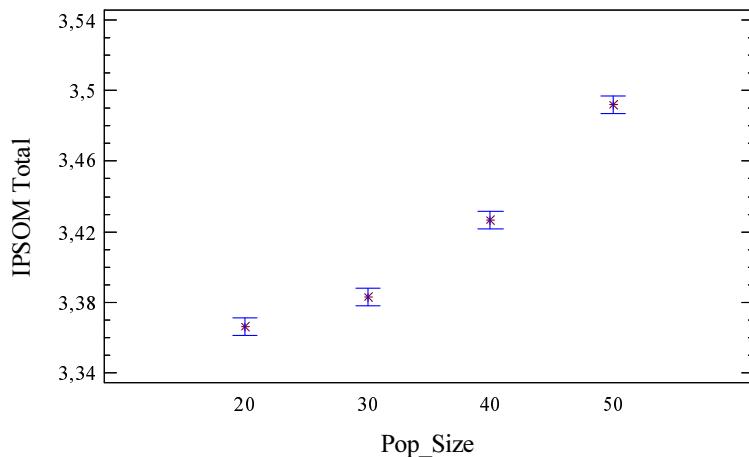


Figura A.59 – Gráfico de medias para el factor Pop_Size, experimento SSD50_P13.

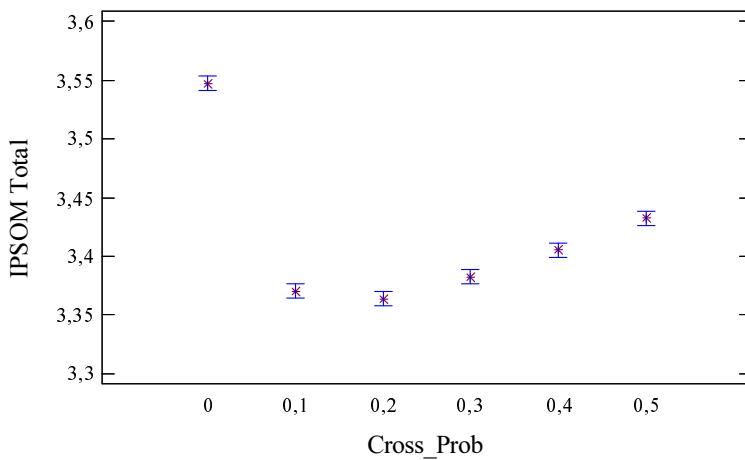


Figura A.60 – Gráfico de medias para el factor Cross_Prob, experimento SSD50_P13.

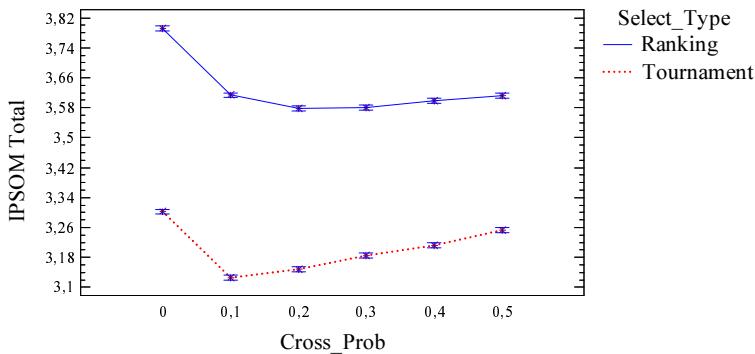


Figura A.61 – Gráfico de medias para la interacción entre los factores Cross_Prob y Select_Type, que sirve para clarificar el anterior gráfico, experimento SSD50_P13.

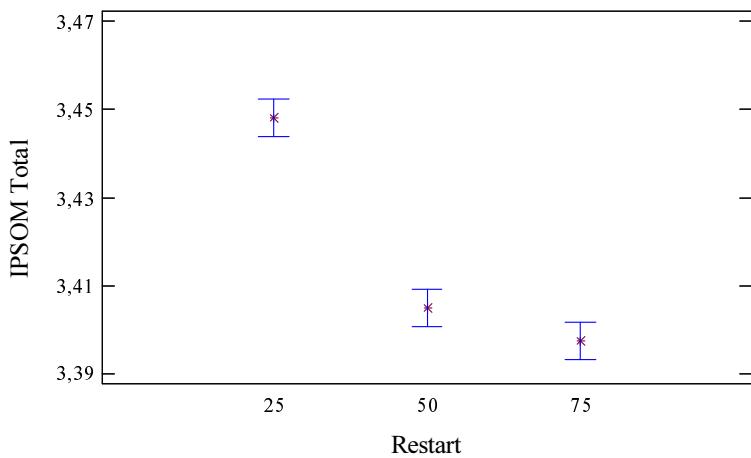


Figura A.62 – Gráfico de medias para el factor `Restart`, experimento SSD50_P13.

A.3.2. Experimento SSD100_P13

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	56,1717	5	11,2343	509,91	0,0000
B:Cross_Type	144,138	8	18,0172	817,77	0,0000
C:Mut_Prob	808,152	4	202,038	9170,19	0,0000
D:Pop_Size	38,3879	3	12,796	580,79	0,0000
E:Restart	7,23523	2	3,61762	164,20	0,0000
F:Select_Type	689,621	1	689,621	31300,82	0,0000
INTERACTIONS					
AB	35,8194	40	0,895484	40,64	0,0000
AC	27,9288	20	1,39644	63,38	0,0000
AD	0,84591	15	0,056394	2,56	0,0008
AE	4,57805	10	0,457805	20,78	0,0000
AF	8,48875	5	1,69775	77,06	0,0000
BC	43,0842	32	1,34638	61,11	0,0000
BD	0,84824	24	0,0353433	1,60	0,0312
BE	1,21562	16	0,0759761	3,45	0,0000
BF	5,91831	8	0,739789	33,58	0,0000
CD	8,52587	12	0,710489	32,25	0,0000
CE	20,6527	8	2,58158	117,17	0,0000
CF	75,1602	4	18,7901	852,85	0,0000
DE	4,09151	6	0,681918	30,95	0,0000
DF	29,4627	3	9,8209	445,76	0,0000
EF	3,94	2	1,97	89,42	0,0000
RESIDUAL	137,722	6251	0,0220321		
TOTAL (CORRECTED)	2151,99	6479			

All F-ratios are based on the residual mean square error.

Tabla A.5 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD100_P13.

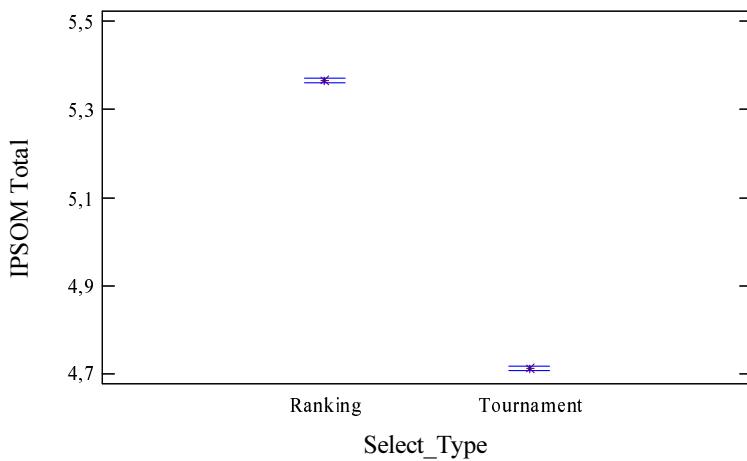


Figura A.63 – Gráfico de medias para el factor Select_Type, experimento SSD100_P13.

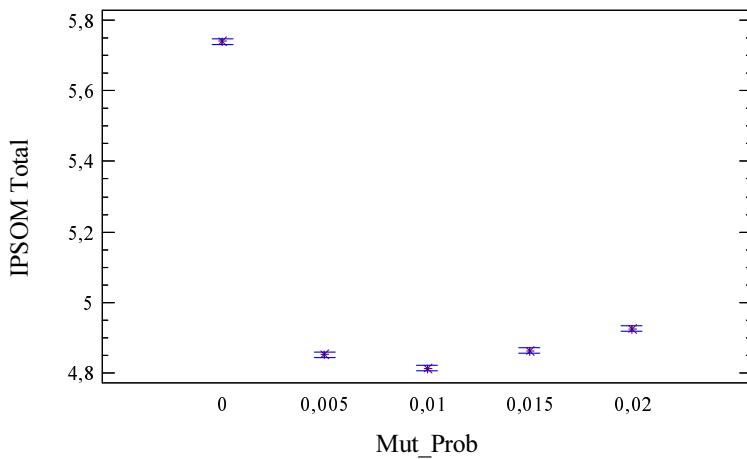


Figura A.64 – Gráfico de medias para el factor Mut_Prob, experimento SSD100_P13.

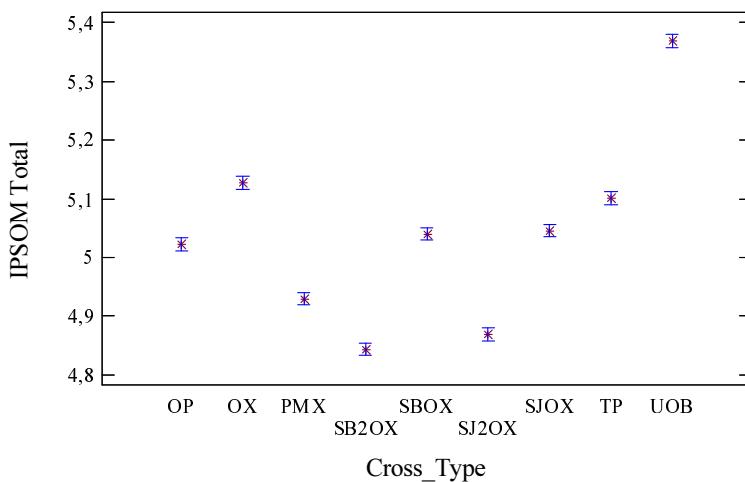


Figura A.65 – Gráfico de medias para el factor `Cross_Type`, experimento SSD100_P13.

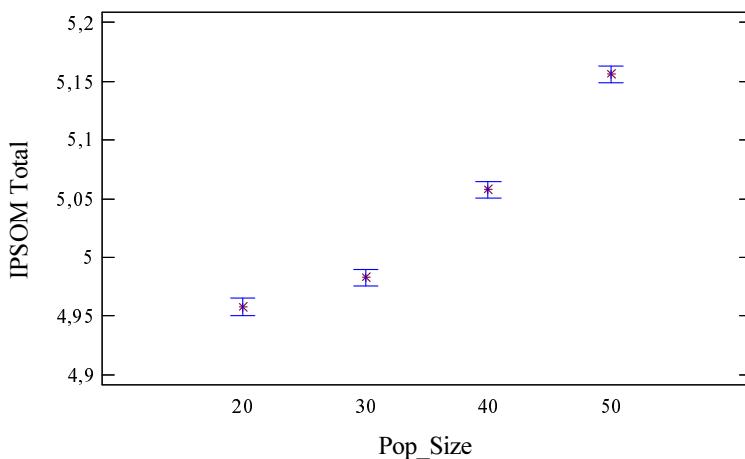


Figura A.66 – Gráfico de medias para el factor `Pop_Size`, experimento SSD100_P13.

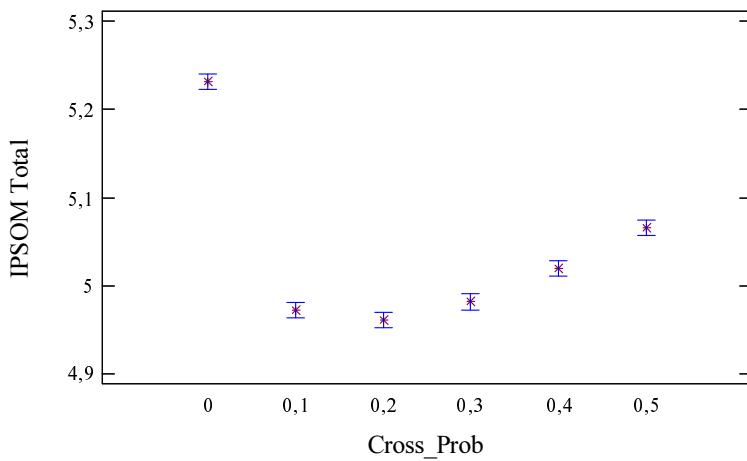


Figura A.67 – Gráfico de medias para el factor Cross_Prob, experimento SSD100_P13.

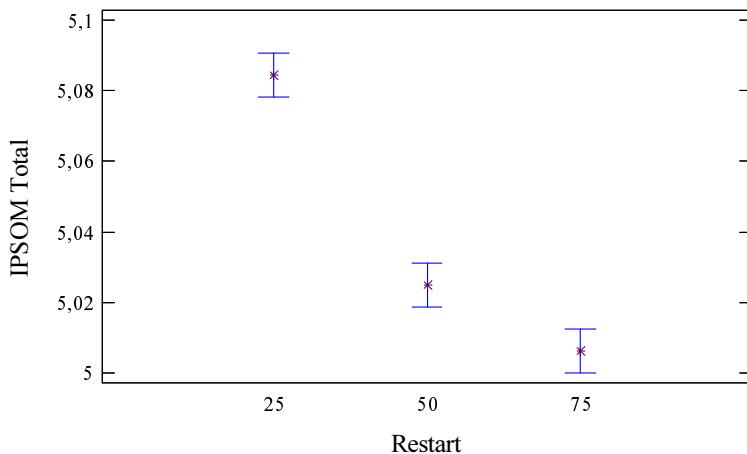


Figura A.68 – Gráfico de medias para el factor Restart, experimento SSD100_P13.

A.3.3. Experimento SSD125_P13

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	66,5167	5	13,3033	477,09	0,0000
B:Cross_Type	171,246	8	21,4058	767,67	0,0000
C:Mut_Prob	921,141	4	230,285	8258,61	0,0000
D:Pop_Size	45,2796	3	15,0932	541,28	0,0000
E:Restart	11,2815	2	5,64077	202,29	0,0000
F:Select_Type	890,467	1	890,467	31934,40	0,0000
INTERACTIONS					
AB	42,4797	40	1,06199	38,09	0,0000
AC	45,6976	20	2,28488	81,94	0,0000
AD	0,816285	15	0,054419	1,95	0,0150
AE	5,12197	10	0,512197	18,37	0,0000
AF	11,9285	5	2,38569	85,56	0,0000
BC	44,6324	32	1,39476	50,02	0,0000
BD	1,25254	24	0,0521893	1,87	0,0061
BE	1,05553	16	0,0659708	2,37	0,0016
BF	9,83655	8	1,22957	44,10	0,0000
CD	10,1105	12	0,84254	30,22	0,0000
CE	25,3485	8	3,16857	113,63	0,0000
CF	92,3887	4	23,0972	828,32	0,0000
DE	4,71748	6	0,786246	28,20	0,0000
DF	33,2279	3	11,076	397,21	0,0000
EF	5,35229	2	2,67614	95,97	0,0000
RESIDUAL	174,305	6251	0,0278843		
TOTAL (CORRECTED)	2614,2	6479			

All F-ratios are based on the residual mean square error.

Tabla A.6 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD125_P13.

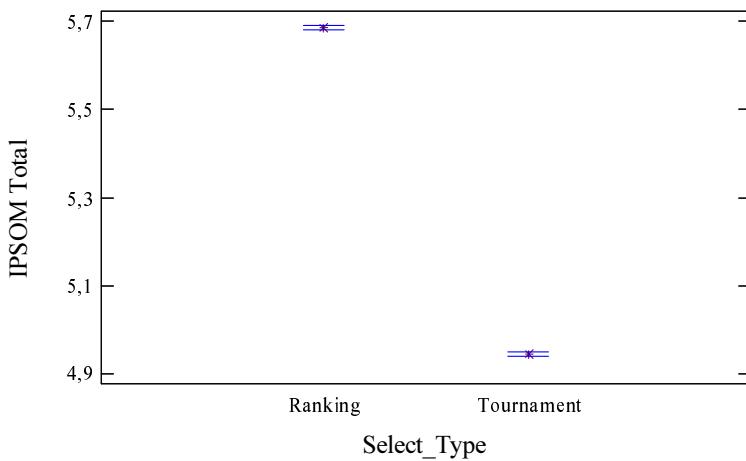


Figura A.69 – Gráfico de medias para el factor Select_Type, experimento SSD125_P13.

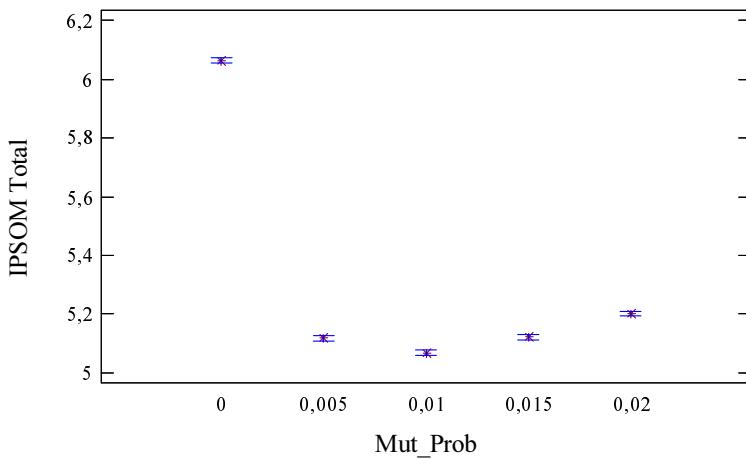


Figura A.70 – Gráfico de medias para el factor Mut_Prob, experimento SSD125_P13.

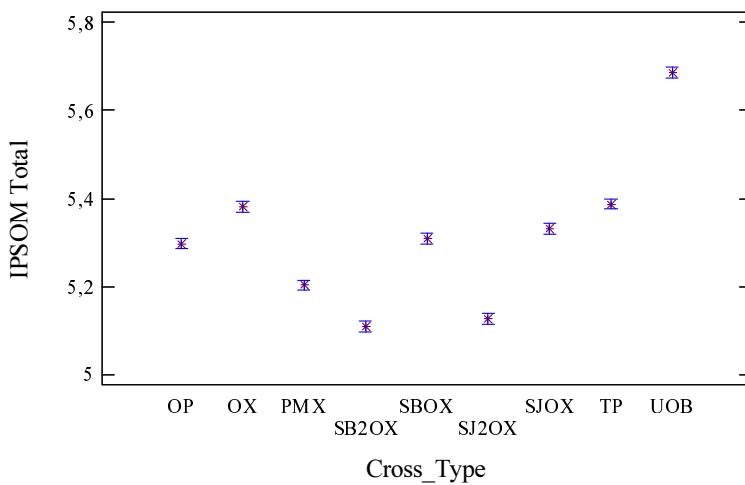


Figura A.71 – Gráfico de medias para el factor `Cross_Type`, experimento SSD125_P13.

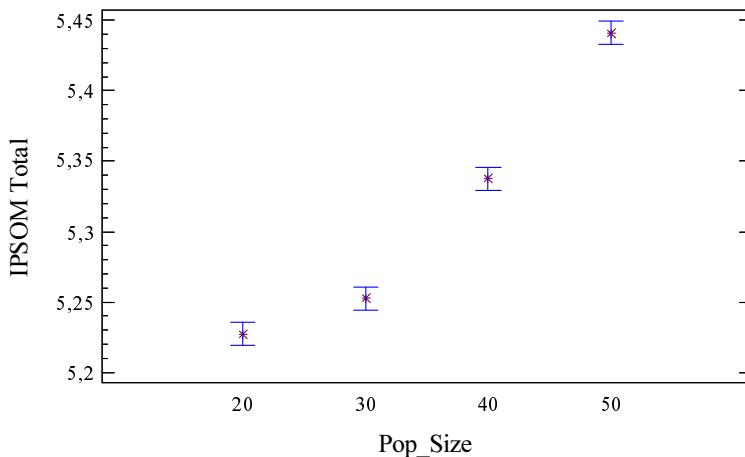


Figura A.72 – Gráfico de medias para el factor `Pop_Size`, experimento SSD125_P13.

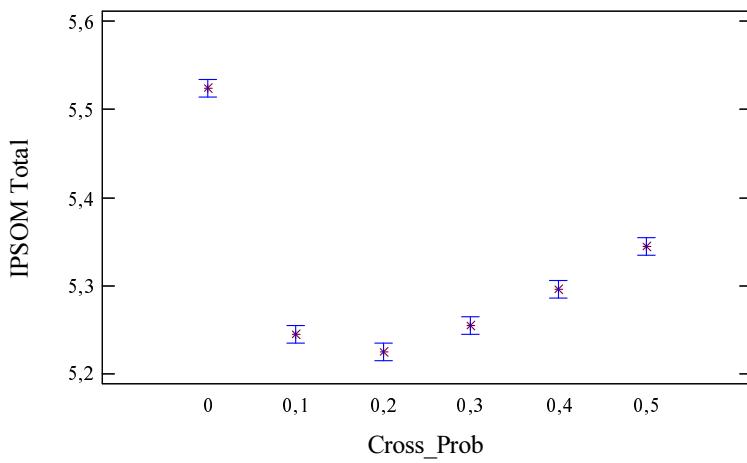


Figura A.73 – Gráfico de medias para el factor Cross_Prob, experimento SSD125_P13.

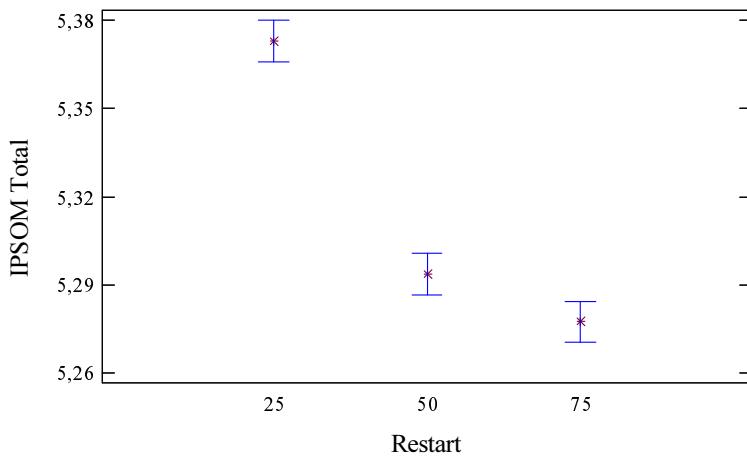


Figura A.74 – Gráfico de medias para el factor Restart, experimento SSD125_P13.

A.3.4. Experimento SSD10_P2

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	31,4274	5	6,28547	319,17	0,0000
B:Cross_Type	27,0984	8	3,3873	172,01	0,0000
C:Mut_Prob	486,839	4	121,71	6180,38	0,0000
D:Pop_Size	4,26068	3	1,42023	72,12	0,0000
E:Restart	3,39159	2	1,6958	86,11	0,0000
F:Select_Type	19,0886	1	19,0886	969,31	0,0000
INTERACTIONS					
AB	7,75963	40	0,193991	9,85	0,0000
AC	47,2469	20	2,36235	119,96	0,0000
AD	0,376403	15	0,0250936	1,27	0,2090
AE	3,68836	10	0,368836	18,73	0,0000
AF	11,7593	5	2,35186	119,43	0,0000
BC	28,713	32	0,897282	45,56	0,0000
BD	1,64158	24	0,0683993	3,47	0,0000
BE	0,369106	16	0,0230691	1,17	0,2826
BF	2,69985	8	0,337481	17,14	0,0000
CD	3,00974	12	0,250812	12,74	0,0000
CE	7,56492	8	0,945615	48,02	0,0000
CF	9,78854	4	2,44713	124,26	0,0000
DE	1,8178	6	0,302967	15,38	0,0000
DF	14,0562	3	4,68538	237,92	0,0000
EF	2,23734	2	1,11867	56,81	0,0000
RESIDUAL	123,101	6251	0,0196929		
TOTAL (CORRECTED)	837,935	6479			

All F-ratios are based on the residual mean square error.

Tabla A.7 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD10_P2.

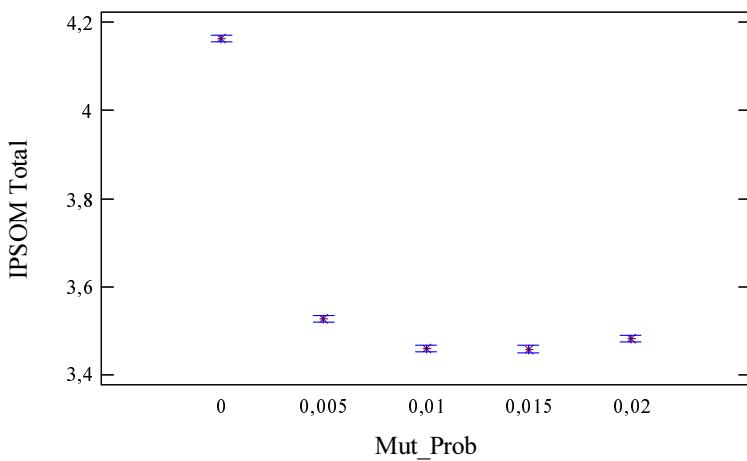


Figura A.75 – Gráfico de medias para el factor Mut_Prob, experimento SSD10_P2.

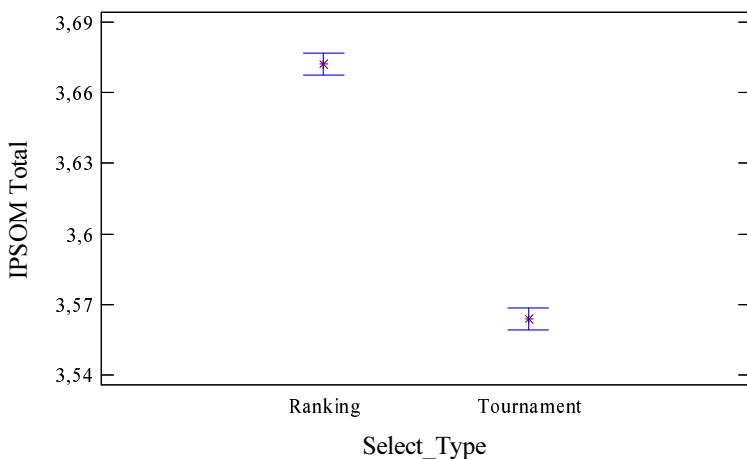


Figura A.76 – Gráfico de medias para el factor Select_Type, experimento SSD10_P2.

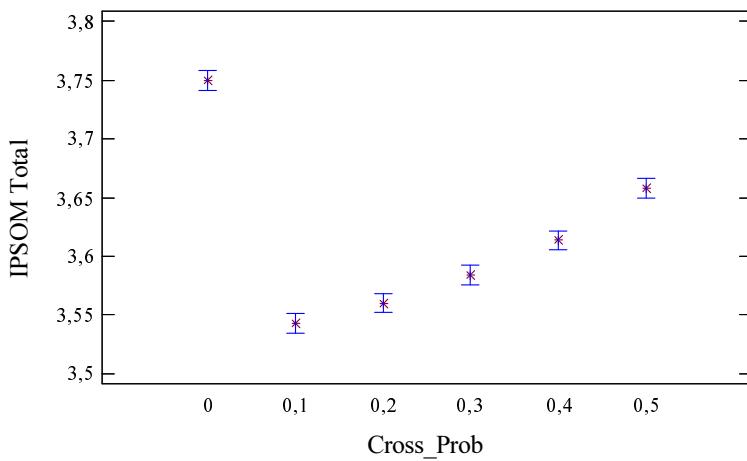


Figura A.77 – Gráfico de medias para el factor Cross_Prob, experimento SSD10_P2.

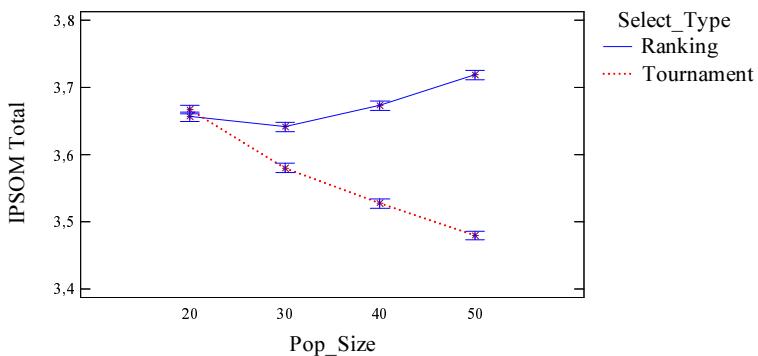


Figura A.78 – Gráfico de medias para la interacción entre los factores Pop_Size y Select_Type, experimento SSD10_P2.

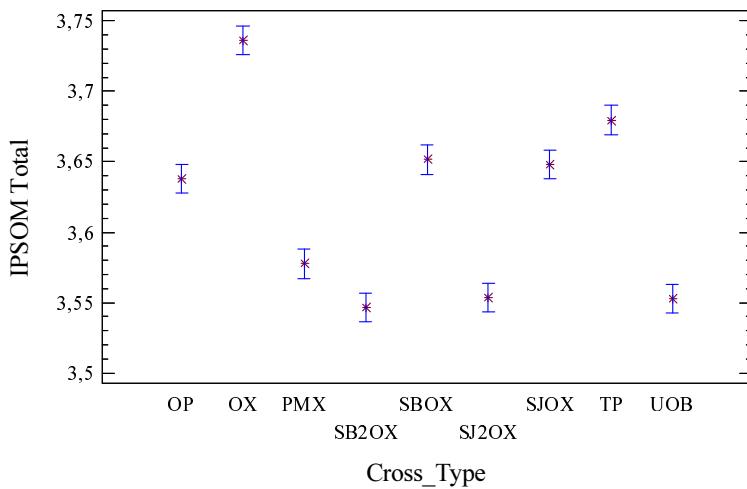


Figura A.79 – Gráfico de medias para el factor Cross_Type, experimento SSD10_P2.

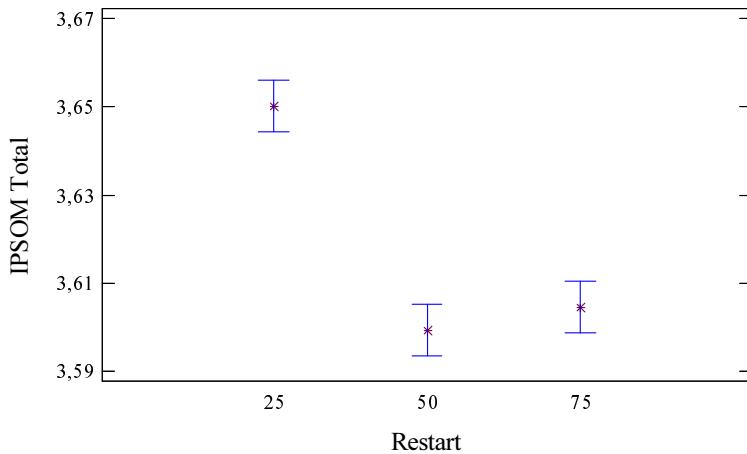


Figura A.80 – Gráfico de medias para el factor Restart, experimento SSD10_P2.

A.3.5. Experimento SSD50_P2

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	80,5404	5	16,1081	628,82	0,0000
B:Cross_Type	58,1193	8	7,26491	283,60	0,0000
C:Mut_Prob	574,478	4	143,619	5606,54	0,0000
D:Pop_Size	15,24	3	5,07999	198,31	0,0000
E:Restart	10,2601	2	5,13007	200,27	0,0000
F:Select_Type	51,3816	1	51,3816	2005,81	0,0000
INTERACTIONS					
AB	16,5148	40	0,412871	16,12	0,0000
AC	115,938	20	5,79691	226,30	0,0000
AD	0,479632	15	0,0319755	1,25	0,2269
AE	5,41081	10	0,541081	21,12	0,0000
AF	13,1812	5	2,63623	102,91	0,0000
BC	53,4576	32	1,67055	65,21	0,0000
BD	2,21473	24	0,0922803	3,60	0,0000
BE	1,05023	16	0,0656396	2,56	0,0006
BF	7,88999	8	0,986249	38,50	0,0000
CD	11,0419	12	0,920156	35,92	0,0000
CE	5,75497	8	0,719371	28,08	0,0000
CF	10,3632	4	2,59081	101,14	0,0000
DE	3,79675	6	0,632791	24,70	0,0000
DF	23,8923	3	7,96411	310,90	0,0000
EF	3,99901	2	1,9995	78,06	0,0000
RESIDUAL	160,128	6251	0,0256164		
TOTAL (CORRECTED)	1225,13	6479			

All F-ratios are based on the residual mean square error.

Tabla A.8 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD50_P2.

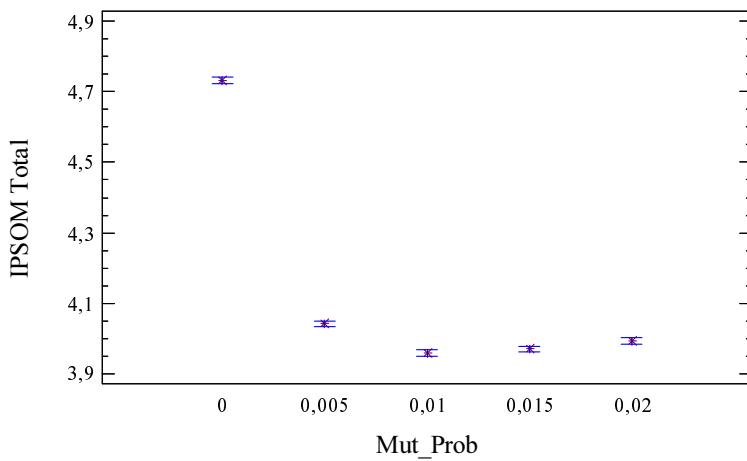


Figura A.81 – Gráfico de medias para el factor Mut_Prob, experimento SSD50_P2.

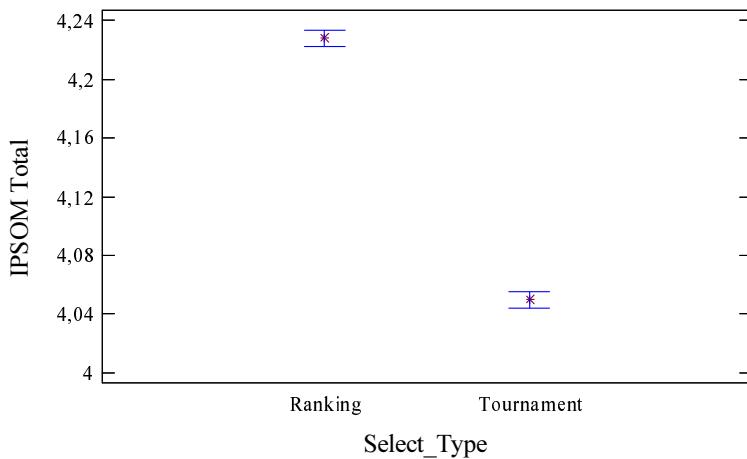


Figura A.82 – Gráfico de medias para el factor Select_Type, experimento SSD50_P2.

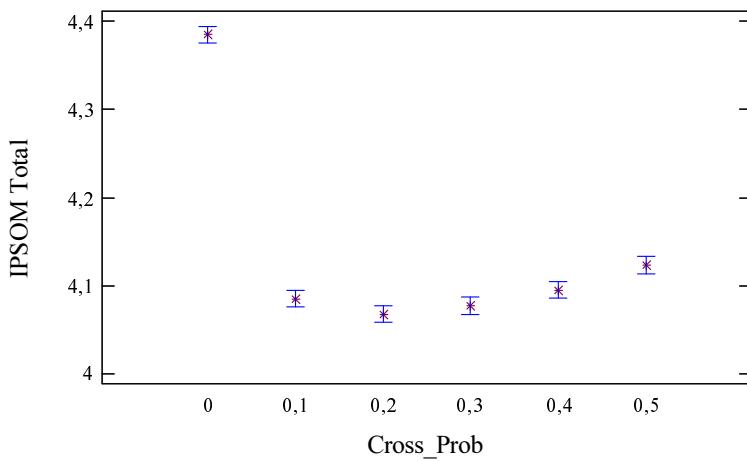


Figura A.83 – Gráfico de medias para el factor Cross_Prob, experimento SSD50_P2.

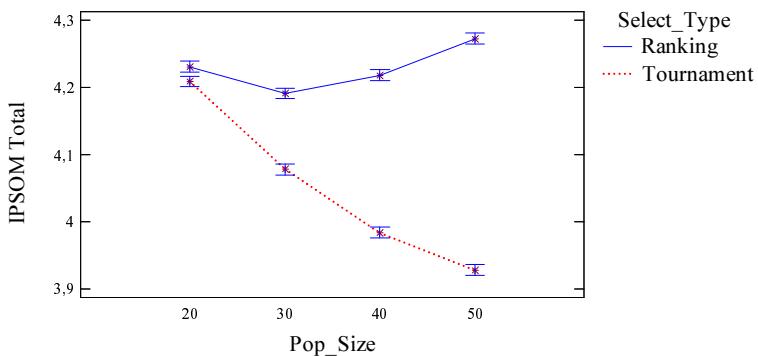


Figura A.84 – Gráfico de medias para la interacción entre los factores Pop_Size y Select_Type, experimento SSD50_P2.

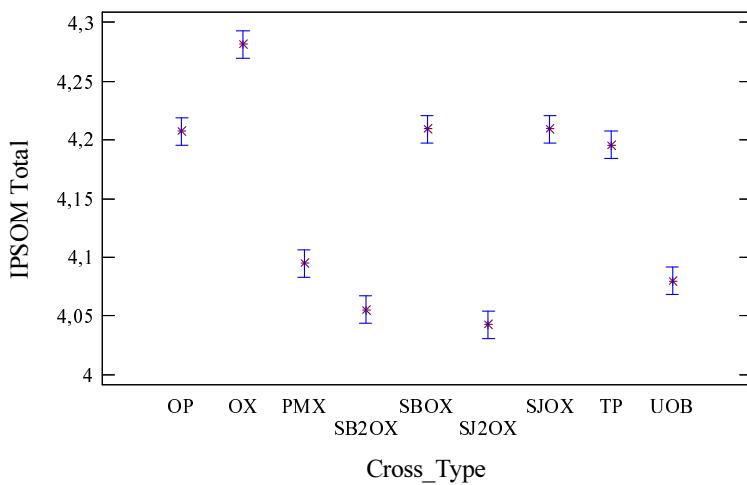


Figura A.85 – Gráfico de medias para el factor Cross_Type, experimento SSD50_P2.

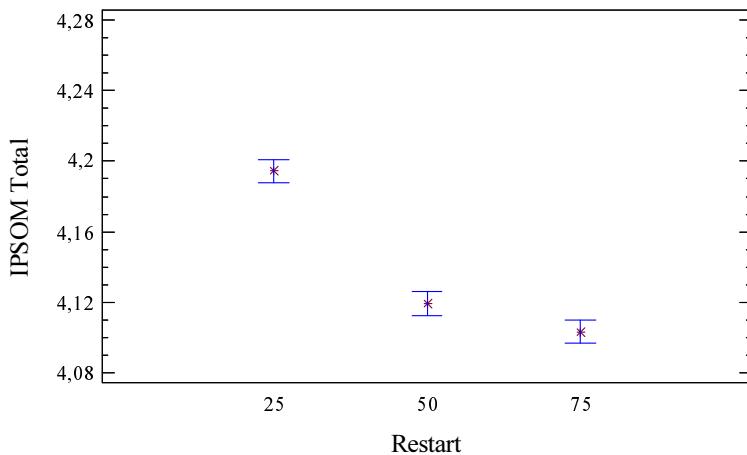


Figura A.86 – Gráfico de medias para el factor Restart, experimento SSD50_P2.

A.3.6. Experimento SSD100_P2

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	118,52	5	23,7039	663,56	0,0000
B:Cross_Type	100,858	8	12,6073	352,92	0,0000
C:Mut_Prob	720,069	4	180,017	5039,32	0,0000
D:Pop_Size	41,5539	3	13,8513	387,75	0,0000
E:Restart	18,2601	2	9,13007	255,58	0,0000
F:Select_Type	38,2525	1	38,2525	1070,82	0,0000
INTERACTIONS					
AB	32,9023	40	0,822558	23,03	0,0000
AC	150,503	20	7,52515	210,66	0,0000
AD	0,595721	15	0,0397147	1,11	0,3389
AE	9,06817	10	0,906817	25,39	0,0000
AF	17,0439	5	3,40878	95,42	0,0000
BC	66,0753	32	2,06485	57,80	0,0000
BD	3,29969	24	0,137487	3,85	0,0000
BE	1,22537	16	0,0765856	2,14	0,0050
BF	6,70163	8	0,837704	23,45	0,0000
CD	13,4245	12	1,11871	31,32	0,0000
CE	9,05609	8	1,13201	31,69	0,0000
CF	13,6659	4	3,41647	95,64	0,0000
DE	4,15841	6	0,693068	19,40	0,0000
DF	20,5148	3	6,83827	191,43	0,0000
EF	5,25739	2	2,62869	73,59	0,0000
RESIDUAL	223,301	6251	0,0357225		
TOTAL (CORRECTED)	1614,31	6479			

All F-ratios are based on the residual mean square error.

Tabla A.9 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD100_P2.

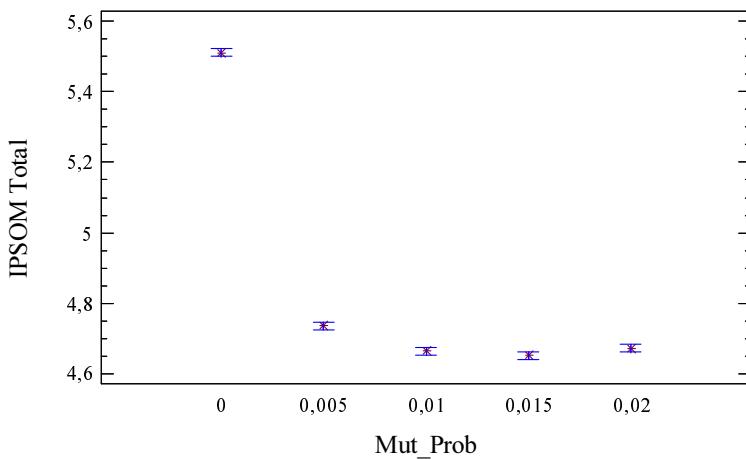


Figura A.87 – Gráfico de medias para el factor Mut_Prob, experimento SSD100_P2.

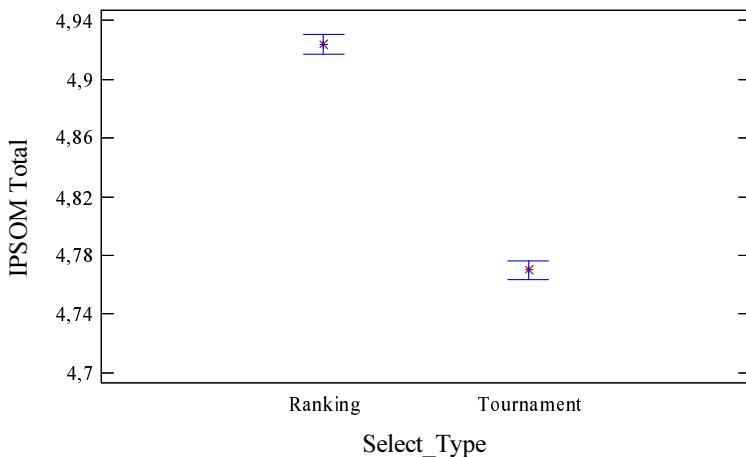


Figura A.88 – Gráfico de medias para el factor Select_Type, experimento SSD100_P2.

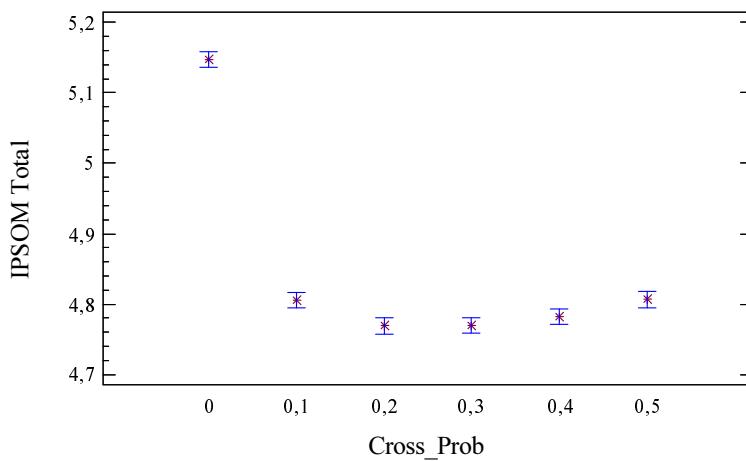


Figura A.89 – Gráfico de medias para el factor Cross_Prob, experimento SSD100_P2.

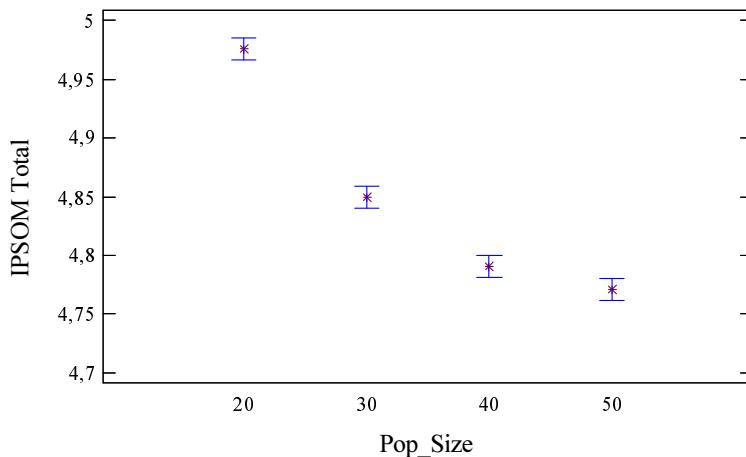


Figura A.90 – Gráfico de medias para el factor Pop_Size, experimento SSD100_P2.

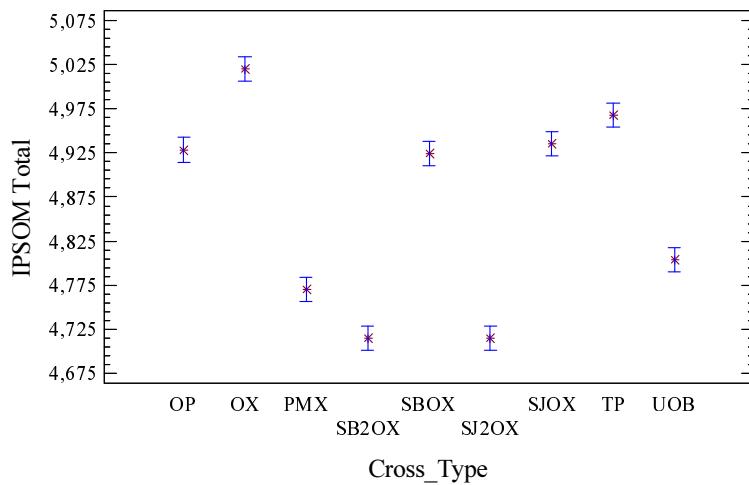


Figura A.91 – Gráfico de medias para el factor Cross_Type, experimento SSD100_P2.

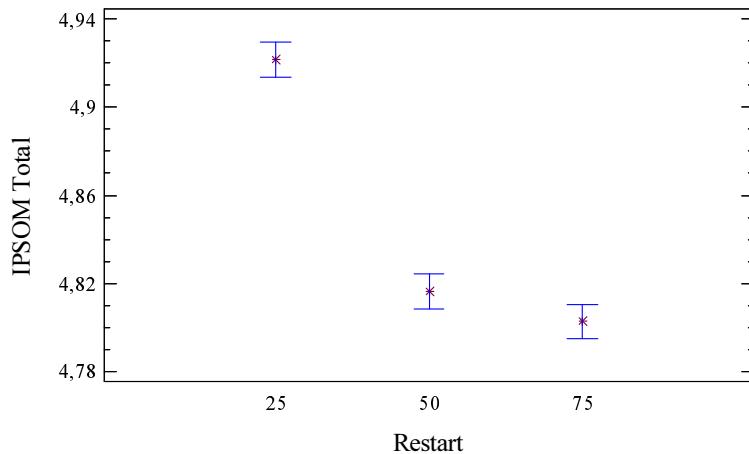


Figura A.92 – Gráfico de medias para el factor Restart, experimento SSD100_P2.

A.3.7. Experimento SSD125_P2

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	129,207	5	25,8414	571,12	0,0000
B:Cross_Type	132,652	8	16,5815	366,47	0,0000
C:Mut_Prob	941,648	4	235,412	5202,81	0,0000
D:Pop_Size	67,9697	3	22,6566	500,73	0,0000
E:Restart	23,7817	2	11,8908	262,80	0,0000
F:Select_Type	56,096	1	56,096	1239,77	0,0000
INTERACTIONS					
AB	38,1	40	0,952501	21,05	0,0000
AC	153,432	20	7,67162	169,55	0,0000
AD	0,711248	15	0,0474165	1,05	0,4012
AE	9,0814	10	0,90814	20,07	0,0000
AF	21,3922	5	4,27845	94,56	0,0000
BC	110,457	32	3,45178	76,29	0,0000
BD	4,85762	24	0,202401	4,47	0,0000
BE	1,60419	16	0,100262	2,22	0,0035
BF	8,33705	8	1,04213	23,03	0,0000
CD	17,8123	12	1,48436	32,81	0,0000
CE	10,1366	8	1,26708	28,00	0,0000
CF	35,623	4	8,90575	196,82	0,0000
DE	7,59129	6	1,26522	27,96	0,0000
DF	29,6168	3	9,87228	218,19	0,0000
EF	6,86801	2	3,434	75,89	0,0000
RESIDUAL	282,84	6251	0,0452471		
TOTAL (CORRECTED)	2089,82	6479			

All F-ratios are based on the residual mean square error.

Tabla A.10 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD125_P2.

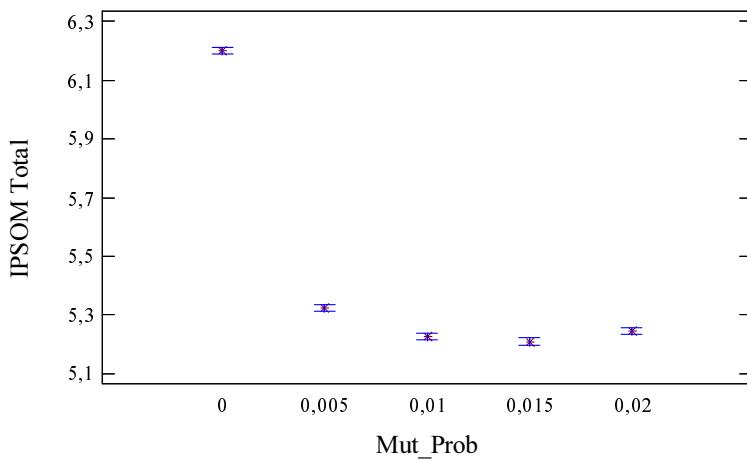


Figura A.93 – Gráfico de medias para el factor Mut_Prob, experimento SSD125_P2.

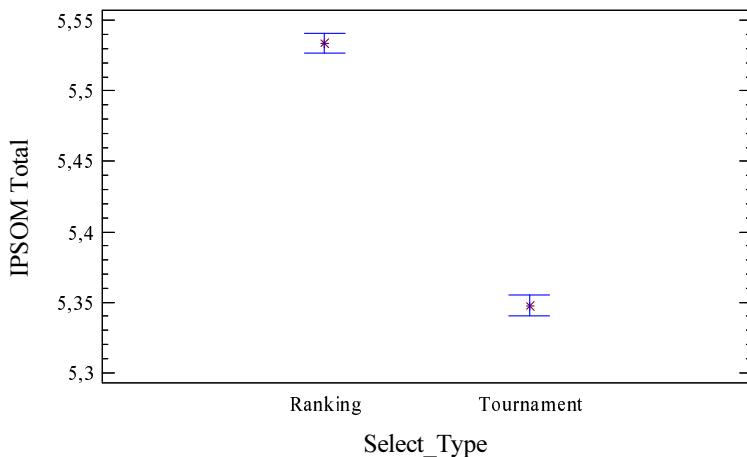


Figura A.94 – Gráfico de medias para el factor Select_Type, experimento SSD125_P2.

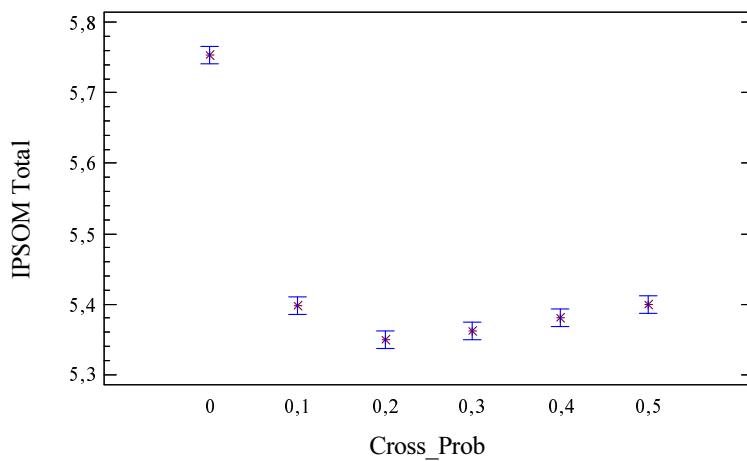


Figura A.95 – Gráfico de medias para el factor Cross_Prob, experimento SSD125_P2.

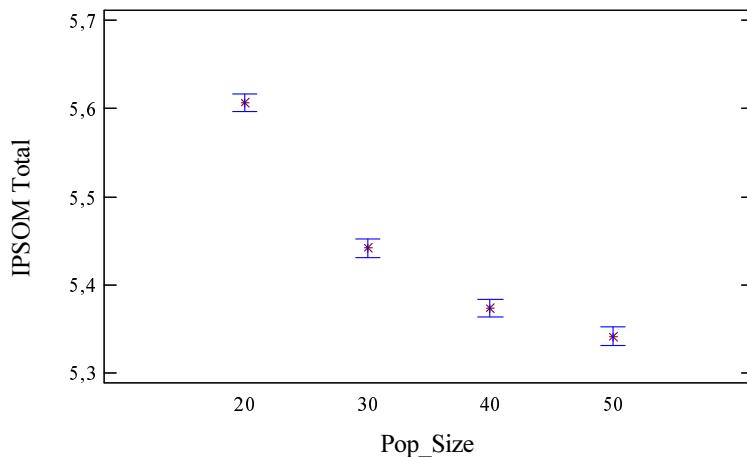


Figura A.96 – Gráfico de medias para el factor Pop_Size, experimento SSD125_P2.

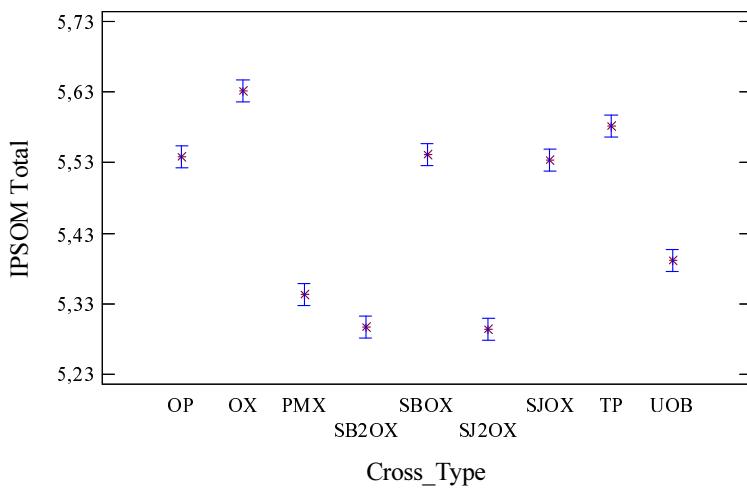


Figura A.97 – Gráfico de medias para el factor Cross_Type, experimento SSD125_P2.

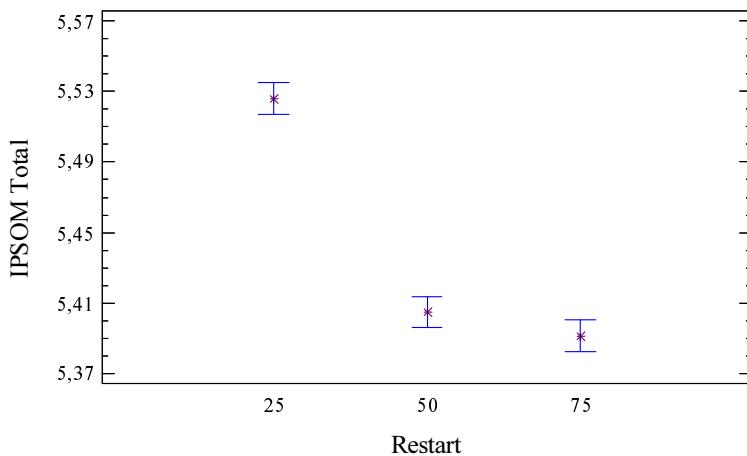


Figura A.98 – Gráfico de medias para el factor Restart, experimento SSD125_P2.

A.3.8. Experimento SSD10_P3

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	31,1961	5	6,23921	204,69	0,0000
B:Cross_Type	38,2728	8	4,7841	156,95	0,0000
C:Mut_Prob	739,075	4	184,769	6061,79	0,0000
D:Pop_Size	6,68362	3	2,22787	73,09	0,0000
E:Restart	0,665826	2	0,332913	10,92	0,0000
F:Select_Type	0,666282	1	0,666282	21,86	0,0000
INTERACTIONS					
AB	12,5519	40	0,313797	10,29	0,0000
AC	73,638	20	3,6819	120,79	0,0000
AD	0,578874	15	0,0385916	1,27	0,2145
AE	6,0795	10	0,60795	19,95	0,0000
AF	12,426	5	2,48519	81,53	0,0000
BC	46,5101	32	1,45344	47,68	0,0000
BD	2,54061	24	0,105859	3,47	0,0000
BE	0,640031	16	0,040002	1,31	0,1790
BF	2,529	8	0,316125	10,37	0,0000
CD	5,84263	12	0,486886	15,97	0,0000
CE	8,87983	8	1,10998	36,42	0,0000
CF	6,9966	4	1,74915	57,39	0,0000
DE	0,973611	6	0,162268	5,32	0,0000
DF	27,5795	3	9,19317	301,60	0,0000
EF	3,24523	2	1,62262	53,23	0,0000
RESIDUAL	190,536	6251	0,0304809		
TOTAL (CORRECTED)	1218,11	6479			

All F-ratios are based on the residual mean square error.

Tabla A.11 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD10_P3.

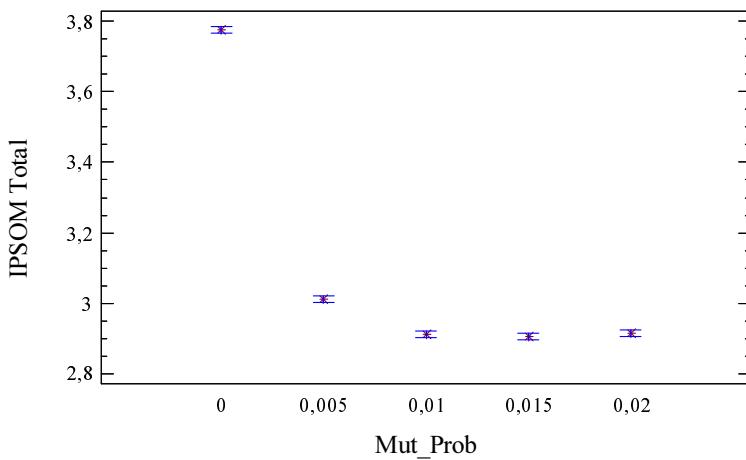


Figura A.99 – Gráfico de medias para el factor Mut_Prob, experimento SSD10_P3.

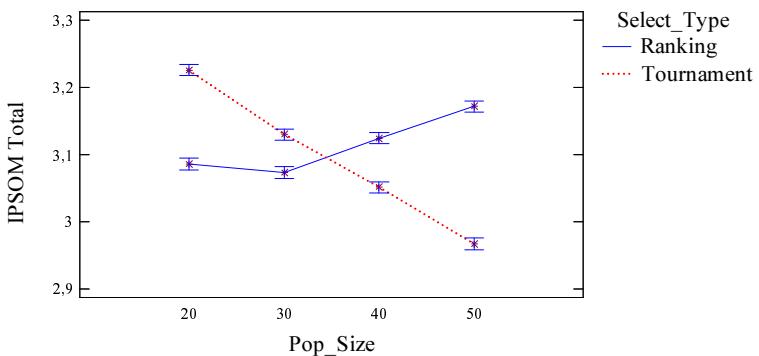


Figura A.100 – Gráfico de medias para la interacción entre los factores Pop_Size y Select_Type, experimento SSD10_P3.

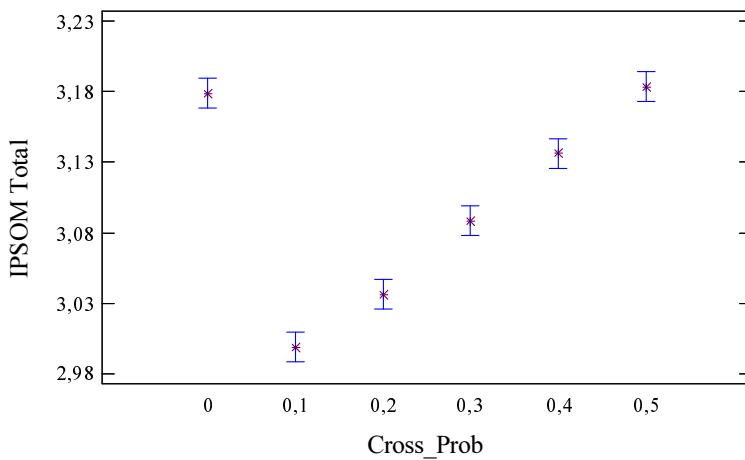


Figura A.101 – Gráfico de medias para el factor Cross_Prob, experimento SSD10_P3.

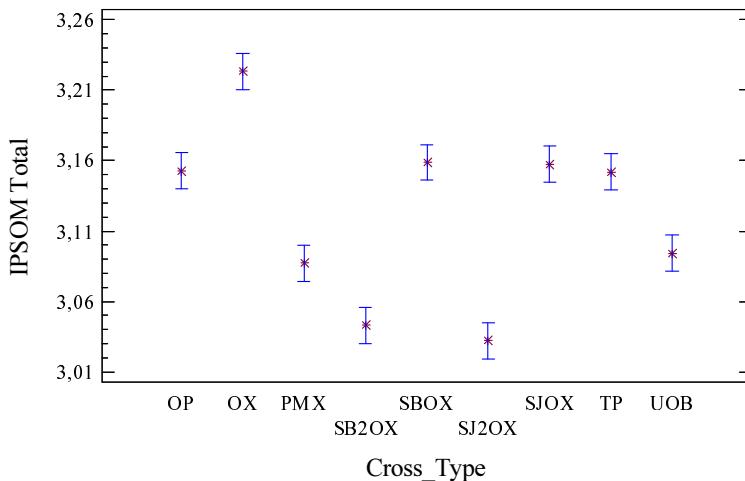


Figura A.102 – Gráfico de medias para el factor Cross_Type, experimento SSD10_P3.

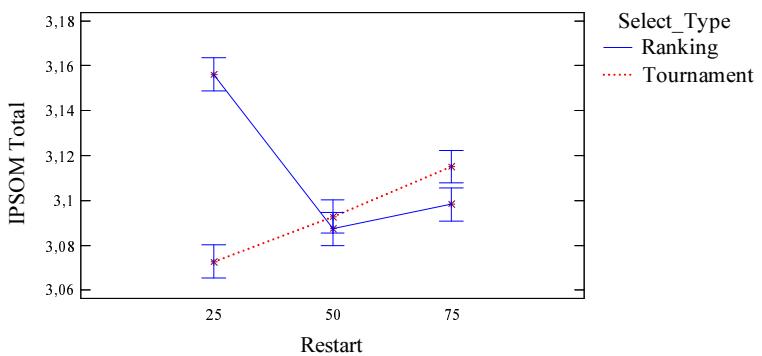


Figura A.103 – Gráfico de medias para la interacción entre los factores Restart y Select_Type, experimento SSD10_P3.

A.3.9. Experimento SSD50_P3

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	57,0747	5	11,4149	239,40	0,0000
B:Cross_Type	125,851	8	15,7314	329,93	0,0000
C:Mut_Prob	894,8	4	223,7	4691,56	0,0000
D:Pop_Size	78,9151	3	26,305	551,68	0,0000
E:Restart	4,4535	2	2,22675	46,70	0,0000
F:Select_Type	31,6665	1	31,6665	664,13	0,0000
INTERACTIONS					
AB	33,2104	40	0,83026	17,41	0,0000
AC	152,776	20	7,63878	160,20	0,0000
AD	1,55645	15	0,103764	2,18	0,0053
AE	10,9524	10	1,09524	22,97	0,0000
AF	35,2442	5	7,04885	147,83	0,0000
BC	106,136	32	3,31674	69,56	0,0000
BD	4,65542	24	0,193976	4,07	0,0000
BE	1,82828	16	0,114268	2,40	0,0014
BF	7,71902	8	0,964878	20,24	0,0000
CD	17,8427	12	1,48689	31,18	0,0000
CE	6,84026	8	0,855033	17,93	0,0000
CF	10,4465	4	2,61162	54,77	0,0000
DE	2,93262	6	0,48877	10,25	0,0000
DF	47,5775	3	15,8592	332,61	0,0000
EF	4,38477	2	2,19238	45,98	0,0000
RESIDUAL	298,056	6251	0,0476814		
TOTAL (CORRECTED)	1934,92	6479			

All F-ratios are based on the residual mean square error.

Tabla A.12 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD50_P3.

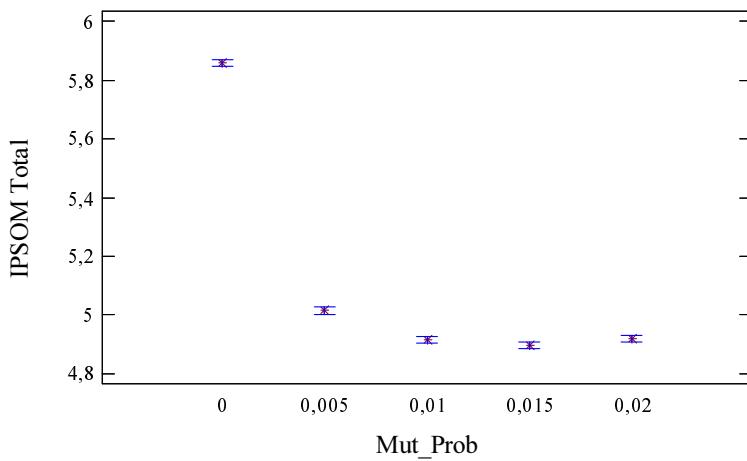


Figura A.104 – Gráfico de medias para el factor Mut_Prob, experimento SSD50_P3.

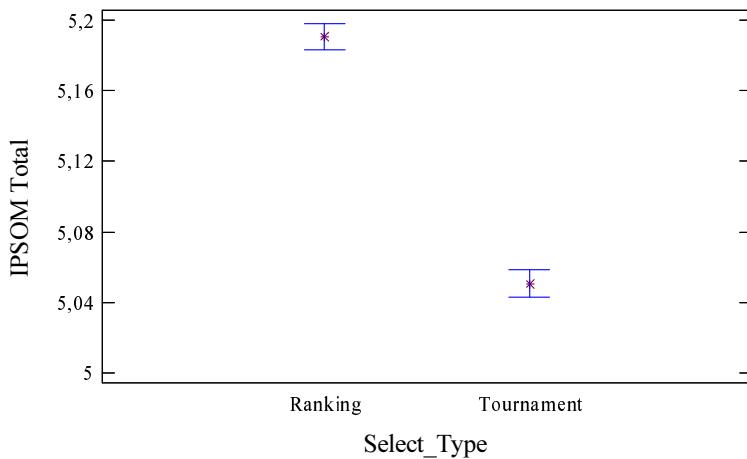


Figura A.105 – Gráfico de medias para el factor Select_Type, experimento SSD50_P3.

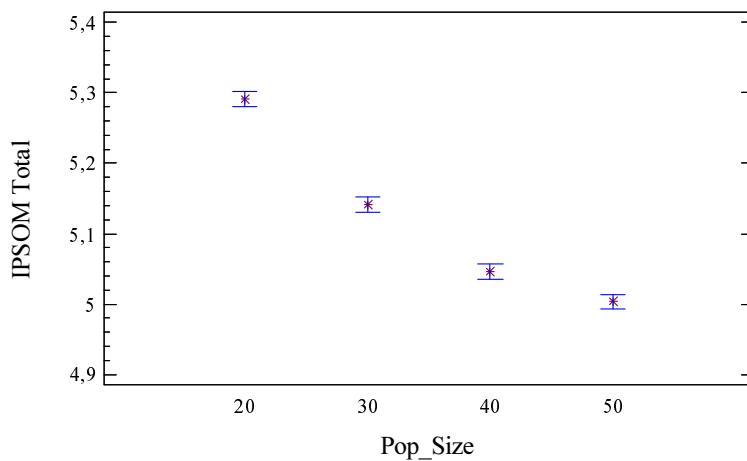


Figura A.106 – Gráfico de medias para el factor `Pop_Size`, experimento `SSD50_P3`.

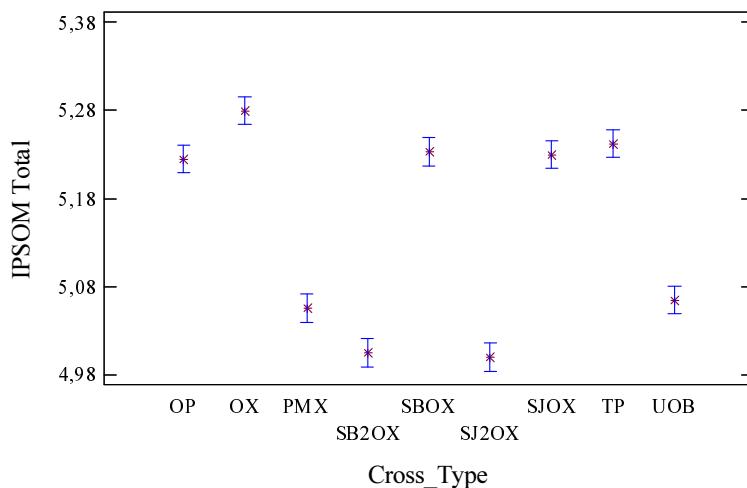


Figura A.107 – Gráfico de medias para el factor `Cross_Type`, experimento `SSD50_P3`.

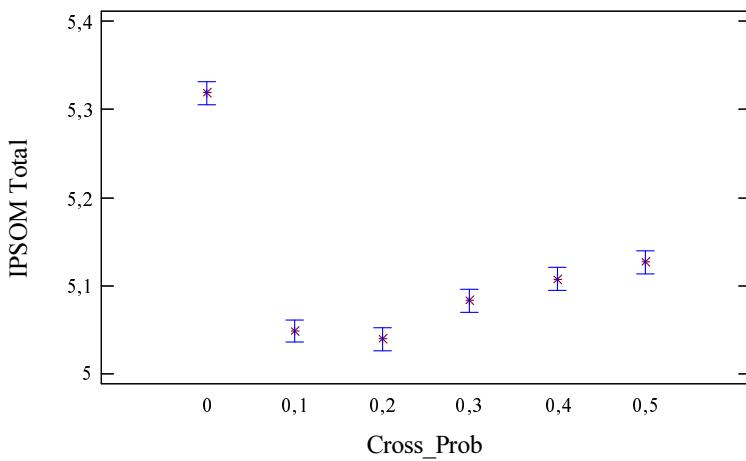


Figura A.108 – Gráfico de medias para el factor Cross_Prob, experimento SSD50_P3.

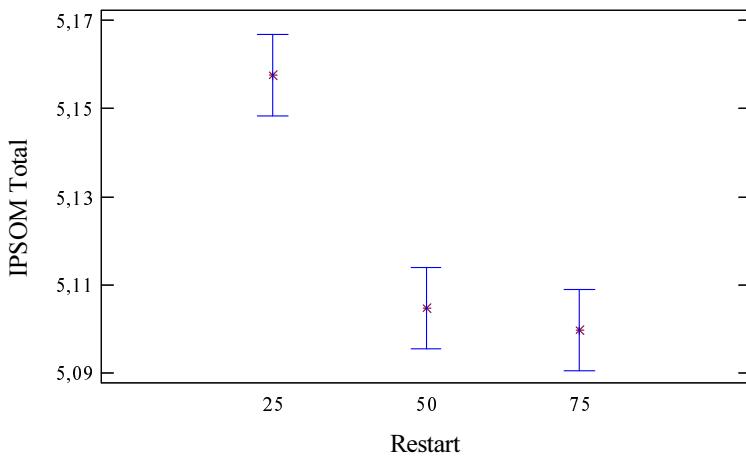


Figura A.109 – Gráfico de medias para el factor Restart, experimento SSD50_P3.

A.3.10. Experimento SSD100_P3

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	119,845	5	23,9691	463,48	0,0000
B:Cross_Type	147,058	8	18,3823	355,45	0,0000
C:Mut_Prob	809,462	4	202,365	3913,05	0,0000
D:Pop_Size	82,9527	3	27,6509	534,67	0,0000
E:Restart	13,1363	2	6,56813	127,00	0,0000
F:Select_Type	5,83863	1	5,83863	112,90	0,0000
INTERACTIONS					
AB	45,2942	40	1,13236	21,90	0,0000
AC	180,625	20	9,03127	174,63	0,0000
AD	1,35287	15	0,0901916	1,74	0,0367
AE	10,1705	10	1,01705	19,67	0,0000
AF	24,4298	5	4,88596	94,48	0,0000
BC	78,3571	32	2,44866	47,35	0,0000
BD	4,49768	24	0,187404	3,62	0,0000
BE	2,22432	16	0,13902	2,69	0,0003
BF	7,43632	8	0,929541	17,97	0,0000
CD	15,1886	12	1,26572	24,47	0,0000
CE	6,86172	8	0,857715	16,59	0,0000
CF	12,8729	4	3,21821	62,23	0,0000
DE	4,7727	6	0,79545	15,38	0,0000
DF	26,3993	3	8,79976	170,16	0,0000
EF	4,96302	2	2,48151	47,98	0,0000
RESIDUAL	323,274	6251	0,0517155		
TOTAL (CORRECTED)	1927,01	6479			

All F-ratios are based on the residual mean square error.

Tabla A.13 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD100_P3.

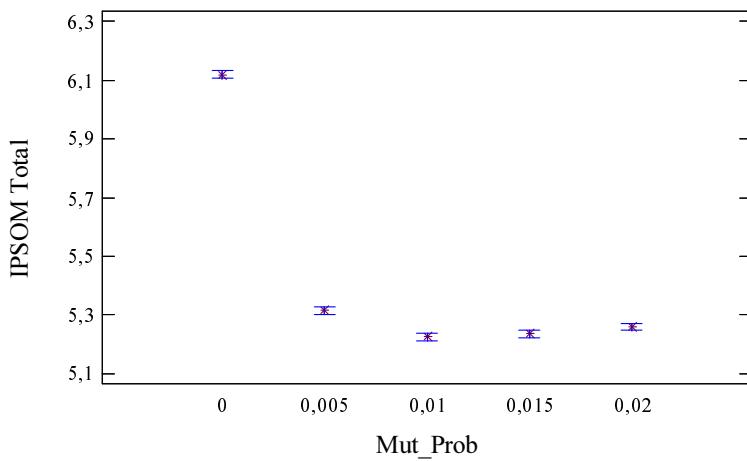


Figura A.110 – Gráfico de medias para el factor Mut_Prob, experimento SSD100_P3.

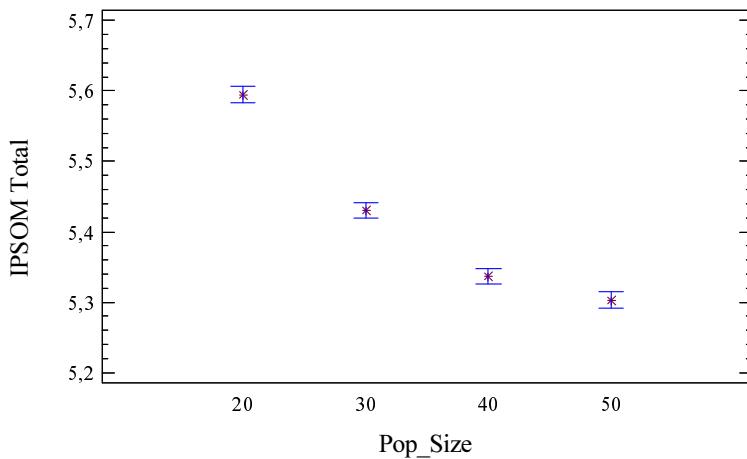


Figura A.111 – Gráfico de medias para el factor Pop_Size, experimento SSD100_P3.

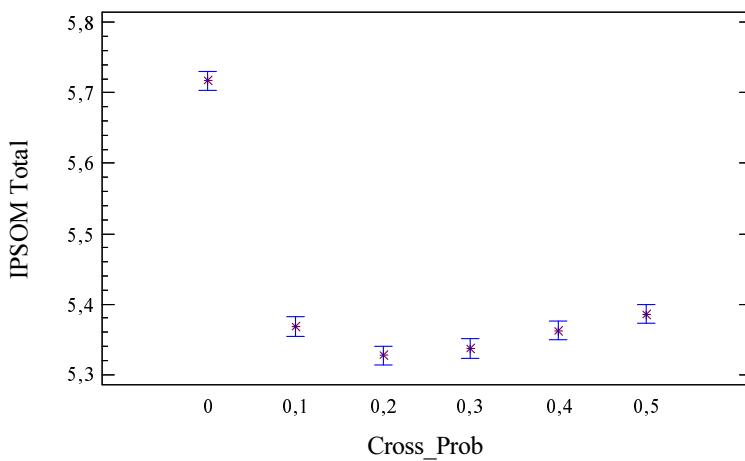


Figura A.112 – Gráfico de medias para el factor Cross_Prob, experimento SSD100_P3.

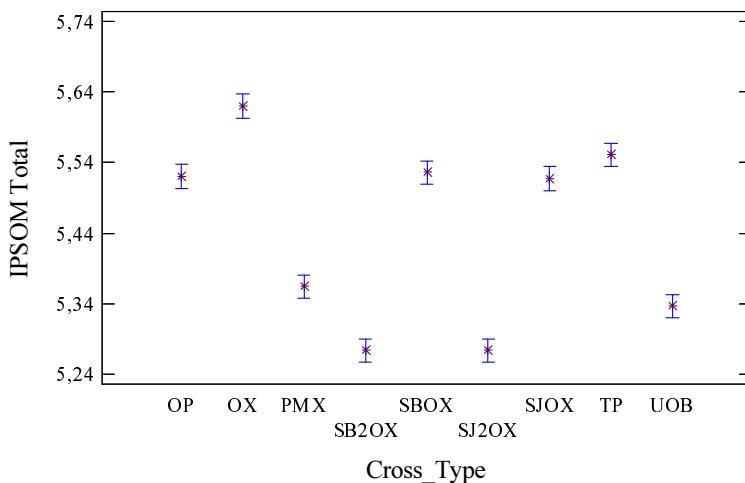


Figura A.113 – Gráfico de medias para el factor Cross_Type, experimento SSD100_P3.

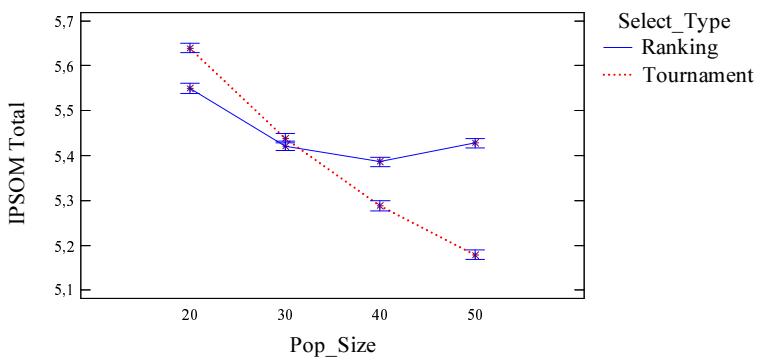


Figura A.114 – Gráfico de medias para la interacción entre los factores Pop_Size y Select_Type, experimento SSD100_P3.

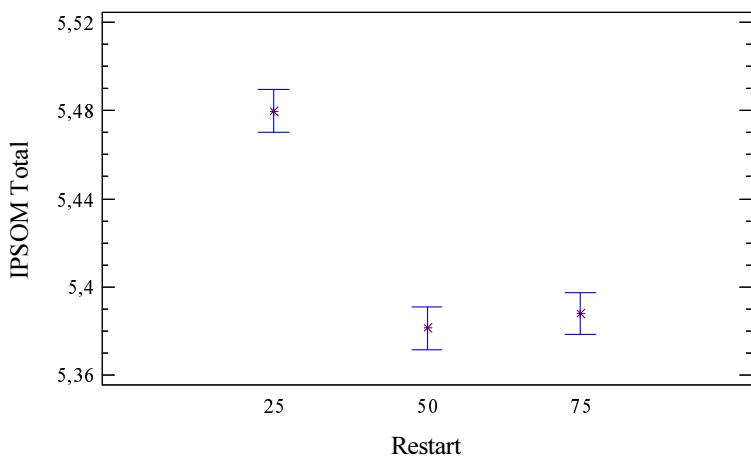


Figura A.115 – Gráfico de medias para el factor Restart, experimento SSD100_P3.

A.3.11. Experimento SSD125_P3

Analysis of Variance for IPSOM Total - Type III Sums of Squares					
Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Cross_Prob	102,452	5	20,4904	322,87	0,0000
B:Cross_Type	236,517	8	29,5646	465,85	0,0000
C:Mut_Prob	984,756	4	246,189	3879,24	0,0000
D:Pop_Size	160,889	3	53,6296	845,05	0,0000
E:Restart	15,0557	2	7,52783	118,62	0,0000
F:Select_Type	24,4753	1	24,4753	385,66	0,0000
INTERACTIONS					
AB	67,7594	40	1,69398	26,69	0,0000
AC	219,011	20	10,9506	172,55	0,0000
AD	1,31002	15	0,0873346	1,38	0,1491
AE	12,5889	10	1,25889	19,84	0,0000
AF	41,3702	5	8,27404	130,38	0,0000
BC	121,763	32	3,80511	59,96	0,0000
BD	5,26966	24	0,219569	3,46	0,0000
BE	2,19972	16	0,137482	2,17	0,0045
BF	10,3532	8	1,29415	20,39	0,0000
CD	22,8655	12	1,90546	30,02	0,0000
CE	6,32233	8	0,790291	12,45	0,0000
CF	18,169	4	4,54226	71,57	0,0000
DE	5,07411	6	0,845685	13,33	0,0000
DF	39,2943	3	13,0981	206,39	0,0000
EF	8,68238	2	4,34119	68,40	0,0000
RESIDUAL	396,708	6251	0,0634632		
TOTAL (CORRECTED)	2502,89	6479			

All F-ratios are based on the residual mean square error.

Tabla A.14 – Tabla ANOVA con los resultados del experimento del algoritmo GA_H propuesto, experimento SSD125_P3.

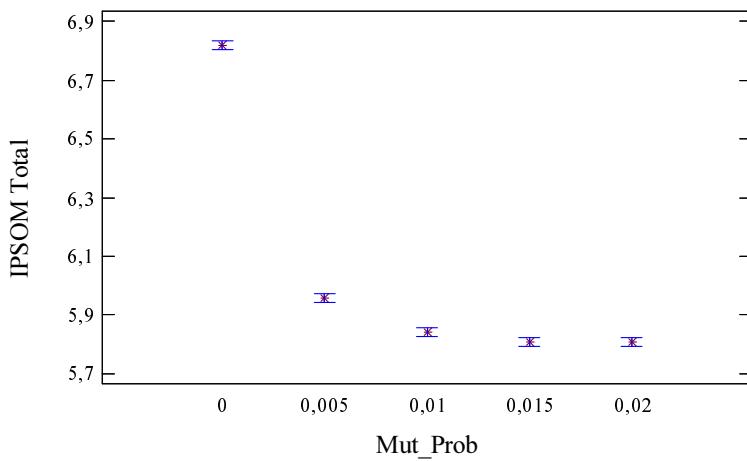


Figura A.116 – Gráfico de medias para el factor Mut_Prob, experimento SSD125_P3.

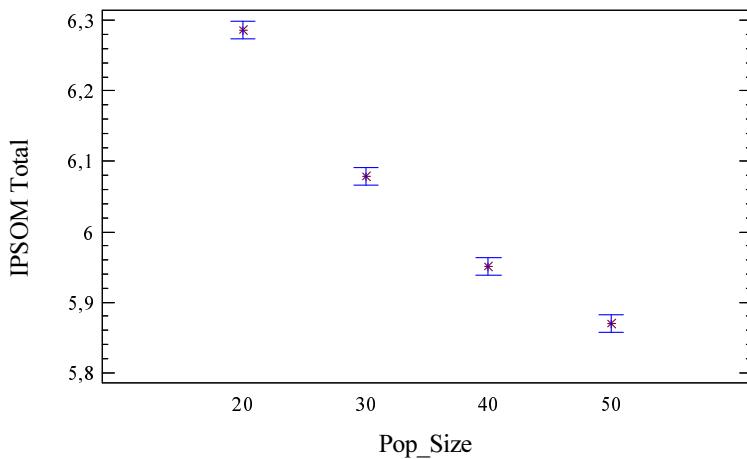


Figura A.117 – Gráfico de medias para el factor Pop_Size, experimento SSD125_P3.

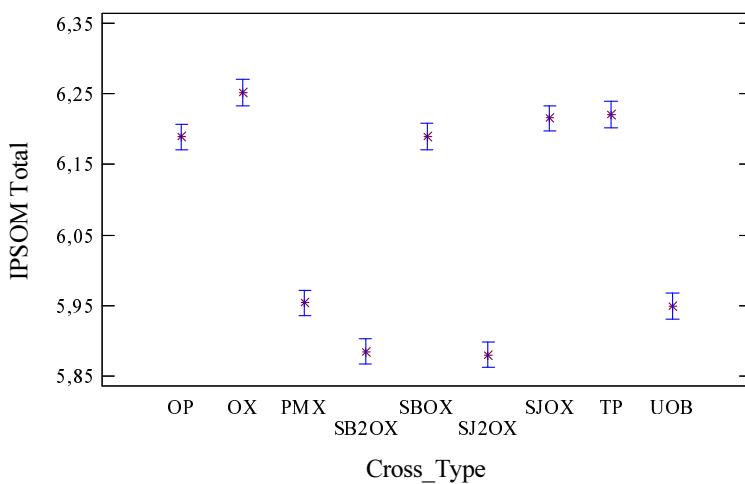


Figura A.118 – Gráfico de medias para el factor `Cross_Type`, experimento SSD125_P3.

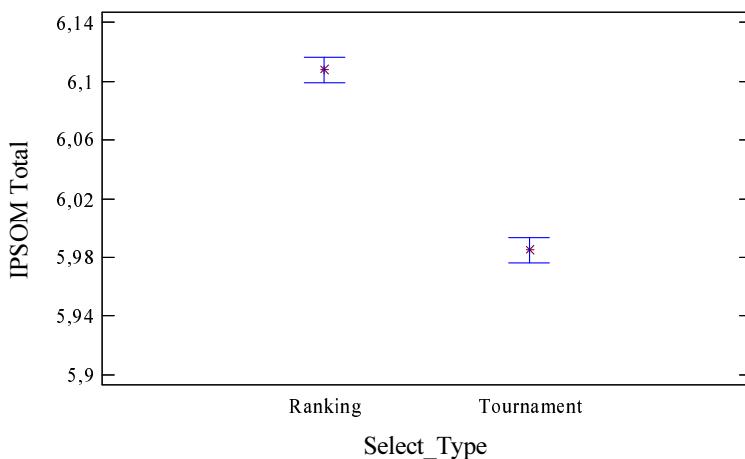


Figura A.119 – Gráfico de medias para el factor `Select_Type`, experimento SSD125_P3.

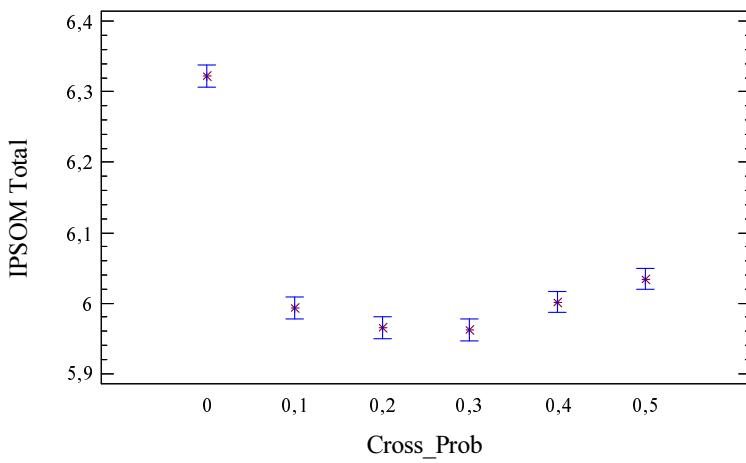


Figura A.120 – Gráfico de medias para el factor Cross_Prob, experimento SSD125_P3.

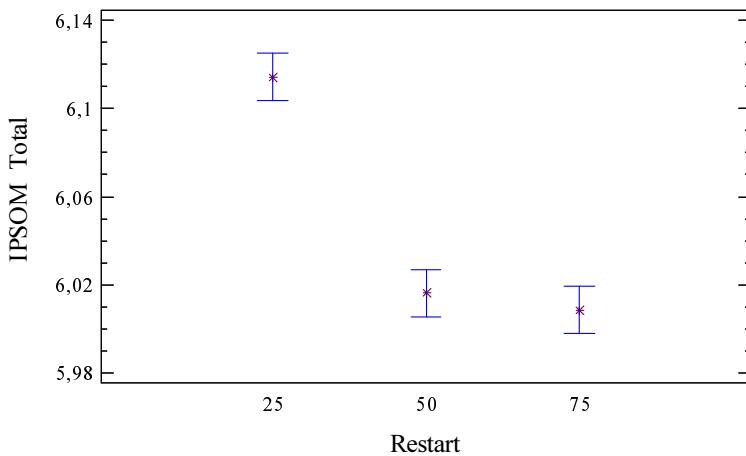
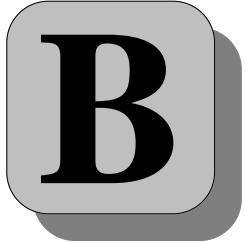


Figura A.121 – Gráfico de medias para el factor Restart, experimento SSD125_P3.

ANEXO



B

BANCOS DE DATOS Y PROBLEMAS EJEMPLO

B.1. Problemas para el taller SDST

En esta sección se detallan las mejores soluciones conocidas para los cuatro grupos de problemas SSD10, SSD50, SSD100 y SSD125 citados en la Sección 5.5.4 del Capítulo 5. Los problemas ejemplo se pueden descargar de <http://www.upv.es/gio> y también se encuentran disponibles en el CD-ROM anexo a esta Tesis Doctoral (ver Anexo C).

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001	20	5	1330	ta002	20	5	1401
ta003	20	5	1161	ta004	20	5	1370
ta005	20	5	1303	ta006	20	5	1269
ta007	20	5	1294	ta008	20	5	1284
ta009	20	5	1313	ta010	20	5	1178
ta011	20	10	1679	ta012	20	10	1751
ta013	20	10	1588	ta014	20	10	1466
ta015	20	10	1510	ta016	20	10	1488
ta017	20	10	1574	ta018	20	10	1632

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta019	20	10	1677	ta020	20	10	1688
ta021	20	20	2391	ta022	20	20	2193
ta023	20	20	2414	ta024	20	20	2315
ta025	20	20	2386	ta026	20	20	2321
ta027	20	20	2360	ta028	20	20	2296
ta029	20	20	2335	ta030	20	20	2267
ta031	50	5	2828	ta032	50	5	2964
ta033	50	5	2754	ta034	50	5	2905
ta035	50	5	2969	ta036	50	5	2954
ta037	50	5	2865	ta038	50	5	2821
ta039	50	5	2684	ta040	50	5	2879
ta041	50	10	3241	ta042	50	10	3119
ta043	50	10	3106	ta044	50	10	3269
ta045	50	10	3226	ta046	50	10	3250
ta047	50	10	3342	ta048	50	10	3248
ta049	50	10	3143	ta050	50	10	3314
ta051	50	20	4144	ta052	50	10	4010
ta053	50	20	3933	ta054	50	10	4030
ta055	50	20	3929	ta056	50	10	4001
ta057	50	20	4033	ta058	50	10	4009
ta059	50	20	4038	ta060	50	10	4089
ta061	100	5	5694	ta062	100	5	5499
ta063	100	5	5429	ta064	100	5	5260
ta065	100	5	5482	ta066	100	5	5328
ta067	100	5	5496	ta068	100	5	5360
ta069	100	5	5690	ta070	100	5	5572
ta071	100	10	6140	ta072	100	10	5727
ta073	100	10	5970	ta074	100	10	6218
ta075	100	10	5876	ta076	100	10	5659
ta077	100	10	5956	ta078	100	10	6009
ta079	100	10	6217	ta080	100	10	6144
ta081	100	20	6814	ta082	100	20	6765
ta083	100	20	6848	ta084	100	20	6791
ta085	100	20	6876	ta086	100	20	6965
ta087	100	20	6830	ta088	100	20	7019
ta089	100	20	6926	ta090	100	20	6986

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta091	200	10	11500	ta092	200	10	11255
ta093	200	10	11607	ta094	200	10	11390
ta095	200	10	11271	ta096	200	10	11027
ta097	200	10	11462	ta098	200	10	11444
ta099	200	10	11176	ta100	200	10	11343
ta101	200	20	12291	ta102	200	20	12357
ta103	200	20	12433	ta104	200	20	12435
ta105	200	20	12306	ta106	200	20	12363
ta107	200	20	12459	ta108	200	20	12440
ta109	200	20	12329	ta110	200	20	12363
ta111	500	20	28651	ta112	500	20	29048
ta113	500	20	28844	ta114	500	20	28879
ta115	500	20	28755	ta116	500	20	28953
ta117	500	20	28691	ta118	500	20	29067
ta119	500	20	28568	ta120	500	20	29029

Tabla B.1 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD10.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001	20	5	1571	ta002	20	5	1580
ta003	20	5	1446	ta004	20	5	1644
ta005	20	5	1526	ta006	20	5	1510
ta007	20	5	1531	ta008	20	5	1554
ta009	20	5	1585	ta010	20	5	1435
ta011	20	10	2009	ta012	20	10	2065
ta013	20	10	1897	ta014	20	10	1794
ta015	20	10	1842	ta016	20	10	1816
ta017	20	10	1858	ta018	20	10	1962
ta019	20	10	1994	ta020	20	10	2013
ta021	20	20	2754	ta022	20	20	2571
ta023	20	20	2748	ta024	20	20	2658
ta025	20	20	2765	ta026	20	20	2686
ta027	20	20	2712	ta028	20	20	2668
ta029	20	20	2707	ta030	20	20	2635

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta031	50	5	3334	ta032	50	5	3514
ta033	50	5	3319	ta034	50	5	3447
ta035	50	5	3463	ta036	50	5	3499
ta037	50	5	3386	ta038	50	5	3346
ta039	50	5	3240	ta040	50	5	3413
ta041	50	10	3987	ta042	50	10	3860
ta043	50	10	3883	ta044	50	10	4043
ta045	50	10	3989	ta046	50	10	3952
ta047	50	10	4042	ta048	50	10	3992
ta049	50	10	3896	ta050	50	10	4020
ta051	50	20	5041	ta052	50	10	4895
ta053	50	20	4837	ta054	50	10	4916
ta055	50	20	4840	ta056	50	10	4901
ta057	50	20	4892	ta058	50	10	4892
ta059	50	20	4905	ta060	50	10	4945
ta061	100	5	6686	ta062	100	5	6555
ta063	100	5	6470	ta064	100	5	6308
ta065	100	5	6564	ta066	100	5	6437
ta067	100	5	6468	ta068	100	5	6361
ta069	100	5	6745	ta070	100	5	6636
ta071	100	10	7562	ta072	100	10	7196
ta073	100	10	7329	ta074	100	10	7665
ta075	100	10	7360	ta076	100	10	7073
ta077	100	10	7289	ta078	100	10	7333
ta079	100	10	7531	ta080	100	10	7511
ta081	100	20	8601	ta082	100	20	8484
ta083	100	20	8540	ta084	100	20	8525
ta085	100	20	8554	ta086	100	20	8606
ta087	100	20	8629	ta088	100	20	8732
ta089	100	20	8612	ta090	100	20	8651
ta091	200	10	14214	ta092	200	10	14159
ta093	200	10	14284	ta094	200	10	14061
ta095	200	10	14029	ta096	200	10	13852
ta097	200	10	14368	ta098	200	10	14266
ta099	200	10	14055	ta100	200	10	14206
ta101	200	20	15677	ta102	200	20	15726

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta103	200	20	15696	ta104	200	20	15795
ta105	200	20	15670	ta106	200	20	15726
ta107	200	20	15798	ta108	200	20	15805
ta109	200	20	15830	ta110	200	20	15793
ta111	500	20	37036	ta112	500	20	37643
ta113	500	20	37281	ta114	500	20	37340
ta115	500	20	37088	ta116	500	20	37389
ta117	500	20	37129	ta118	500	20	37474
ta119	500	20	36895	ta120	500	20	37405

Tabla B.2 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD50.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001	20	5	1891	ta002	20	5	1881
ta003	20	5	1758	ta004	20	5	1973
ta005	20	5	1813	ta006	20	5	1824
ta007	20	5	1855	ta008	20	5	1894
ta009	20	5	1879	ta010	20	5	1732
ta011	20	10	2444	ta012	20	10	2458
ta013	20	10	2303	ta014	20	10	2212
ta015	20	10	2286	ta016	20	10	2231
ta017	20	10	2282	ta018	20	10	2381
ta019	20	10	2376	ta020	20	10	2443
ta021	20	20	3245	ta022	20	20	3049
ta023	20	20	3207	ta024	20	20	3164
ta025	20	20	3242	ta026	20	20	3168
ta027	20	20	3191	ta028	20	20	3169
ta029	20	20	3194	ta030	20	20	3111
ta031	50	5	4025	ta032	50	5	4191
ta033	50	5	4009	ta034	50	5	4116
ta035	50	5	4090	ta036	50	5	4176
ta037	50	5	4085	ta038	50	5	4069
ta039	50	5	3944	ta040	50	5	4131
ta041	50	10	4861	ta042	50	10	4781

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta043	50	10	4809	ta044	50	10	4875
ta045	50	10	4918	ta046	50	10	4950
ta047	50	10	5013	ta048	50	10	4991
ta049	50	10	4859	ta050	50	10	4978
ta051	50	20	6161	ta052	50	10	5986
ta053	50	20	6039	ta054	50	10	6080
ta055	50	20	6039	ta056	50	10	6078
ta057	50	20	6033	ta058	50	10	6060
ta059	50	20	6051	ta060	50	10	6153
ta061	100	5	7990	ta062	100	5	7840
ta063	100	5	7774	ta064	100	5	7607
ta065	100	5	7790	ta066	100	5	7728
ta067	100	5	7839	ta068	100	5	7734
ta069	100	5	8006	ta070	100	5	7923
ta071	100	10	9343	ta072	100	10	9113
ta073	100	10	9223	ta074	100	10	9482
ta075	100	10	9250	ta076	100	10	8887
ta077	100	10	9032	ta078	100	10	9164
ta079	100	10	9283	ta080	100	10	9328
ta081	100	20	10702	ta082	100	20	10716
ta083	100	20	10794	ta084	100	20	10626
ta085	100	20	10767	ta086	100	20	10854
ta087	100	20	10848	ta088	100	20	11002
ta089	100	20	10727	ta090	100	20	10780
ta091	200	10	17690	ta092	200	10	17499
ta093	200	10	17939	ta094	200	10	17417
ta095	200	10	17564	ta096	200	10	17394
ta097	200	10	17683	ta098	200	10	17798
ta099	200	10	17519	ta100	200	10	17690
ta101	200	20	19956	ta102	200	20	20033
ta103	200	20	20156	ta104	200	20	20134
ta105	200	20	19930	ta106	200	20	19968
ta107	200	20	20062	ta108	200	20	20086
ta109	200	20	20021	ta110	200	20	20054
ta111	500	20	47298	ta112	500	20	47799
ta113	500	20	47874	ta114	500	20	47806

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta115	500	20	47756	ta116	500	20	47964
ta117	500	20	47512	ta118	500	20	48053
ta119	500	20	47488	ta120	500	20	47895

Tabla B.3 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD100.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001	20	5	2072	ta002	20	5	2040
ta003	20	5	1933	ta004	20	5	2137
ta005	20	5	1979	ta006	20	5	1979
ta007	20	5	2015	ta008	20	5	2067
ta009	20	5	2005	ta010	20	5	1876
ta011	20	10	2671	ta012	20	10	2661
ta013	20	10	2515	ta014	20	10	2430
ta015	20	10	2502	ta016	20	10	2445
ta017	20	10	2485	ta018	20	10	2593
ta019	20	10	2588	ta020	20	10	2655
ta021	20	20	3499	ta022	20	20	3292
ta023	20	20	3475	ta024	20	20	3437
ta025	20	20	3514	ta026	20	20	3442
ta027	20	20	3452	ta028	20	20	3431
ta029	20	20	3456	ta030	20	20	3383
ta031	50	5	4394	ta032	50	5	4527
ta033	50	5	4301	ta034	50	5	4456
ta035	50	5	4436	ta036	50	5	4512
ta037	50	5	4438	ta038	50	5	4332
ta039	50	5	4262	ta040	50	5	4474
ta041	50	10	5408	ta042	50	10	5301
ta043	50	10	5339	ta044	50	10	5428
ta045	50	10	5368	ta046	50	10	5429
ta047	50	10	5433	ta048	50	10	5373
ta049	50	10	5322	ta050	50	10	5457
ta051	50	20	6732	ta052	50	10	6585
ta053	50	20	6537	ta054	50	10	6674

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta055	50	20	6632	ta056	50	10	6632
ta057	50	20	6676	ta058	50	10	6625
ta059	50	20	6592	ta060	50	10	6757
ta061	100	5	8695	ta062	100	5	8511
ta063	100	5	8378	ta064	100	5	8325
ta065	100	5	8552	ta066	100	5	8309
ta067	100	5	8579	ta068	100	5	8314
ta069	100	5	8681	ta070	100	5	8654
ta071	100	10	10190	ta072	100	10	10014
ta073	100	10	10177	ta074	100	10	10471
ta075	100	10	10037	ta076	100	10	9824
ta077	100	10	10030	ta078	100	10	10160
ta079	100	10	10119	ta080	100	10	10167
ta081	100	20	11890	ta082	100	20	11899
ta083	100	20	11942	ta084	100	20	11717
ta085	100	20	11917	ta086	100	20	12009
ta087	100	20	11998	ta088	100	20	11982
ta089	100	20	11933	ta090	100	20	12048
ta091	200	10	19356	ta092	200	10	19425
ta093	200	10	19682	ta094	200	10	19249
ta095	200	10	19327	ta096	200	10	18944
ta097	200	10	19771	ta098	200	10	19401
ta099	200	10	19215	ta100	200	10	19339
ta101	200	20	22177	ta102	200	20	22443
ta103	200	20	22337	ta104	200	20	22160
ta105	200	20	22146	ta106	200	20	22130
ta107	200	20	22362	ta108	200	20	22416
ta109	200	20	22104	ta110	200	20	22187
ta111	500	20	52832	ta112	500	20	53465
ta113	500	20	53177	ta114	500	20	53021
ta115	500	20	52902	ta116	500	20	53394
ta117	500	20	52814	ta118	500	20	53088
ta119	500	20	52648	ta120	500	20	52901

Tabla B.4 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD125.

B.2. Problemas para el taller de flujo híbrido con tiempos de cambio de partida dependientes de la secuencia

En esta sección se detallan las mejores soluciones conocidas para los 12 grupos de problemas SSD10_P13, SSD50_P13, SSD100_P13, SSD125_P13, SSD10_P2, SSD50_P2, SSD100_P2, SSD125_P2, SSD10_P3, SSD50_P3, SSD100_P3 y SSD125_P3, citados en la Sección 6.5.4.4 del Capítulo 6. Los 1320 problemas y todos los datos se pueden descargar de <http://www.upv.es/gio>. Dado el gran volumen de datos y la cantidad de espacio que estos ocupan (alrededor de 1,34 GigaBytes), no ha sido posible incluir estos problemas de ejemplo en el CD-ROM anexo. Si se desea también se pueden solicitar por correo electrónico rruiz@eio.upv.es.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD10_P13	20	5	1001	ta002_SSD10_P13	20	5	1029
ta003_SSD10_P13	20	5	1163	ta004_SSD10_P13	20	5	1178
ta005_SSD10_P13	20	5	1343	ta006_SSD10_P13	20	5	1314
ta007_SSD10_P13	20	5	1099	ta008_SSD10_P13	20	5	1074
ta009_SSD10_P13	20	5	1153	ta010_SSD10_P13	20	5	1038
ta011_SSD10_P13	20	10	1482	ta012_SSD10_P13	20	10	1439
ta013_SSD10_P13	20	10	1316	ta014_SSD10_P13	20	10	1326
ta015_SSD10_P13	20	10	1389	ta016_SSD10_P13	20	10	1347
ta017_SSD10_P13	20	10	1435	ta018_SSD10_P13	20	10	1426
ta019_SSD10_P13	20	10	1334	ta020_SSD10_P13	20	10	1316
ta021_SSD10_P13	20	20	1468	ta022_SSD10_P13	20	20	1897
ta023_SSD10_P13	20	20	1761	ta024_SSD10_P13	20	20	1890
ta025_SSD10_P13	20	20	1402	ta026_SSD10_P13	20	20	1849
ta027_SSD10_P13	20	20	1765	ta028_SSD10_P13	20	20	1886
ta029_SSD10_P13	20	20	1889	ta030_SSD10_P13	20	20	1724
ta031_SSD10_P13	50	5	2760	ta032_SSD10_P13	50	5	2593
ta033_SSD10_P13	50	5	2716	ta034_SSD10_P13	50	5	2726
ta035_SSD10_P13	50	5	3075	ta036_SSD10_P13	50	5	2772
ta037_SSD10_P13	50	5	2518	ta038_SSD10_P13	50	5	2692
ta039_SSD10_P13	50	5	2758	ta040_SSD10_P13	50	5	2710

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta041_SSD10_P13	50	10	2786	ta042_SSD10_P13	50	10	3002
ta043_SSD10_P13	50	10	3045	ta044_SSD10_P13	50	10	2992
ta045_SSD10_P13	50	10	2932	ta046_SSD10_P13	50	10	2924
ta047_SSD10_P13	50	10	3003	ta048_SSD10_P13	50	10	3002
ta049_SSD10_P13	50	10	1254	ta050_SSD10_P13	50	10	2998
ta051_SSD10_P13	50	20	3188	ta052_SSD10_P13	50	20	3419
ta053_SSD10_P13	50	20	3435	ta054_SSD10_P13	50	20	3201
ta055_SSD10_P13	50	20	3360	ta056_SSD10_P13	50	20	3383
ta057_SSD10_P13	50	20	3587	ta058_SSD10_P13	50	20	3421
ta059_SSD10_P13	50	20	3378	ta060_SSD10_P13	50	20	3444
ta061_SSD10_P13	100	5	2133	ta062_SSD10_P13	100	5	5285
ta063_SSD10_P13	100	5	2079	ta064_SSD10_P13	100	5	1839
ta065_SSD10_P13	100	5	5263	ta066_SSD10_P13	100	5	5323
ta067_SSD10_P13	100	5	5289	ta068_SSD10_P13	100	5	5062
ta069_SSD10_P13	100	5	5645	ta070_SSD10_P13	100	5	4983
ta071_SSD10_P13	100	10	2427	ta072_SSD10_P13	100	10	5741
ta073_SSD10_P13	100	10	5759	ta074_SSD10_P13	100	10	5749
ta075_SSD10_P13	100	10	2145	ta076_SSD10_P13	100	10	5842
ta077_SSD10_P13	100	10	5682	ta078_SSD10_P13	100	10	5728
ta079_SSD10_P13	100	10	5149	ta080_SSD10_P13	100	10	5718
ta081_SSD10_P13	100	20	5830	ta082_SSD10_P13	100	20	6501
ta083_SSD10_P13	100	20	6226	ta084_SSD10_P13	100	20	5811
ta085_SSD10_P13	100	20	6336	ta086_SSD10_P13	100	20	6206
ta087_SSD10_P13	100	20	5624	ta088_SSD10_P13	100	20	5498
ta089_SSD10_P13	100	20	6167	ta090_SSD10_P13	100	20	6040
ta091_SSD10_P13	200	10	11037	ta092_SSD10_P13	200	10	11337
ta093_SSD10_P13	200	10	10934	ta094_SSD10_P13	200	10	10900
ta095_SSD10_P13	200	10	10864	ta096_SSD10_P13	200	10	10519
ta097_SSD10_P13	200	10	10702	ta098_SSD10_P13	200	10	11069
ta099_SSD10_P13	200	10	10317	ta100_SSD10_P13	200	10	10876
ta101_SSD10_P13	200	20	11355	ta102_SSD10_P13	200	20	11612
ta103_SSD10_P13	200	20	11231	ta104_SSD10_P13	200	20	11511
ta105_SSD10_P13	200	20	10821	ta106_SSD10_P13	200	20	11535
ta107_SSD10_P13	200	20	11772	ta108_SSD10_P13	200	20	11301

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta109_SSD10_P13	200	20	11737	ta110_SSD10_P13	200	20	11621

Tabla B.5 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD10_P13.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD50_P13	20	5	1065	ta002_SSD50_P13	20	5	1197
ta003_SSD50_P13	20	5	1298	ta004_SSD50_P13	20	5	1247
ta005_SSD50_P13	20	5	1423	ta006_SSD50_P13	20	5	1392
ta007_SSD50_P13	20	5	1220	ta008_SSD50_P13	20	5	1146
ta009_SSD50_P13	20	5	1318	ta010_SSD50_P13	20	5	1177
ta011_SSD50_P13	20	10	1752	ta012_SSD50_P13	20	10	1567
ta013_SSD50_P13	20	10	1570	ta014_SSD50_P13	20	10	1508
ta015_SSD50_P13	20	10	1628	ta016_SSD50_P13	20	10	1515
ta017_SSD50_P13	20	10	1680	ta018_SSD50_P13	20	10	1654
ta019_SSD50_P13	20	10	1647	ta020_SSD50_P13	20	10	1477
ta021_SSD50_P13	20	20	1691	ta022_SSD50_P13	20	20	2199
ta023_SSD50_P13	20	20	2030	ta024_SSD50_P13	20	20	2097
ta025_SSD50_P13	20	20	1615	ta026_SSD50_P13	20	20	2115
ta027_SSD50_P13	20	20	2082	ta028_SSD50_P13	20	20	2138
ta029_SSD50_P13	20	20	2253	ta030_SSD50_P13	20	20	1992
ta031_SSD50_P13	50	5	3093	ta032_SSD50_P13	50	5	2887
ta033_SSD50_P13	50	5	3237	ta034_SSD50_P13	50	5	3049
ta035_SSD50_P13	50	5	3386	ta036_SSD50_P13	50	5	3316
ta037_SSD50_P13	50	5	2697	ta038_SSD50_P13	50	5	2977
ta039_SSD50_P13	50	5	3004	ta040_SSD50_P13	50	5	3026
ta041_SSD50_P13	50	10	3092	ta042_SSD50_P13	50	10	3570
ta043_SSD50_P13	50	10	3645	ta044_SSD50_P13	50	10	3636
ta045_SSD50_P13	50	10	3482	ta046_SSD50_P13	50	10	3469
ta047_SSD50_P13	50	10	3450	ta048_SSD50_P13	50	10	3535
ta049_SSD50_P13	50	10	1667	ta050_SSD50_P13	50	10	3474
ta051_SSD50_P13	50	20	3747	ta052_SSD50_P13	50	20	4030
ta053_SSD50_P13	50	20	4153	ta054_SSD50_P13	50	20	3895
ta055_SSD50_P13	50	20	4060	ta056_SSD50_P13	50	20	4092

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta057_SSD50_P13	50	20	4307	ta058_SSD50_P13	50	20	4101
ta059_SSD50_P13	50	20	4111	ta060_SSD50_P13	50	20	4164
ta061_SSD50_P13	100	5	2966	ta062_SSD50_P13	100	5	5468
ta063_SSD50_P13	100	5	2782	ta064_SSD50_P13	100	5	2330
ta065_SSD50_P13	100	5	5807	ta066_SSD50_P13	100	5	6163
ta067_SSD50_P13	100	5	5861	ta068_SSD50_P13	100	5	5590
ta069_SSD50_P13	100	5	6427	ta070_SSD50_P13	100	5	5478
ta071_SSD50_P13	100	10	3312	ta072_SSD50_P13	100	10	6934
ta073_SSD50_P13	100	10	6363	ta074_SSD50_P13	100	10	6452
ta075_SSD50_P13	100	10	3029	ta076_SSD50_P13	100	10	6902
ta077_SSD50_P13	100	10	6583	ta078_SSD50_P13	100	10	6323
ta079_SSD50_P13	100	10	5371	ta080_SSD50_P13	100	10	6512
ta081_SSD50_P13	100	20	6917	ta082_SSD50_P13	100	20	7483
ta083_SSD50_P13	100	20	7651	ta084_SSD50_P13	100	20	6842
ta085_SSD50_P13	100	20	7636	ta086_SSD50_P13	100	20	7527
ta087_SSD50_P13	100	20	6534	ta088_SSD50_P13	100	20	6133
ta089_SSD50_P13	100	20	7525	ta090_SSD50_P13	100	20	7297
ta091_SSD50_P13	200	10	12113	ta092_SSD50_P13	200	10	13109
ta093_SSD50_P13	200	10	12689	ta094_SSD50_P13	200	10	12991
ta095_SSD50_P13	200	10	11956	ta096_SSD50_P13	200	10	11476
ta097_SSD50_P13	200	10	12135	ta098_SSD50_P13	200	10	13349
ta099_SSD50_P13	200	10	11376	ta100_SSD50_P13	200	10	12225
ta101_SSD50_P13	200	20	13428	ta102_SSD50_P13	200	20	14618
ta103_SSD50_P13	200	20	13330	ta104_SSD50_P13	200	20	14142
ta105_SSD50_P13	200	20	12376	ta106_SSD50_P13	200	20	13426
ta107_SSD50_P13	200	20	13956	ta108_SSD50_P13	200	20	13582
ta109_SSD50_P13	200	20	14664	ta110_SSD50_P13	200	20	14547

Tabla B.6 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD50_P13.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD100_P13	20	5	1171	ta002_SSD100_P13	20	5	1410
ta003_SSD100_P13	20	5	1502	ta004_SSD100_P13	20	5	1340

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta005_SSD100_P13	20	5	1574	ta006_SSD100_P13	20	5	1557
ta007_SSD100_P13	20	5	1424	ta008_SSD100_P13	20	5	1232
ta009_SSD100_P13	20	5	1519	ta010_SSD100_P13	20	5	1335
ta011_SSD100_P13	20	10	2066	ta012_SSD100_P13	20	10	1818
ta013_SSD100_P13	20	10	1891	ta014_SSD100_P13	20	10	1704
ta015_SSD100_P13	20	10	1987	ta016_SSD100_P13	20	10	1804
ta017_SSD100_P13	20	10	2000	ta018_SSD100_P13	20	10	1967
ta019_SSD100_P13	20	10	1906	ta020_SSD100_P13	20	10	1677
ta021_SSD100_P13	20	20	1960	ta022_SSD100_P13	20	20	2571
ta023_SSD100_P13	20	20	2363	ta024_SSD100_P13	20	20	2471
ta025_SSD100_P13	20	20	1883	ta026_SSD100_P13	20	20	2486
ta027_SSD100_P13	20	20	2472	ta028_SSD100_P13	20	20	2490
ta029_SSD100_P13	20	20	2692	ta030_SSD100_P13	20	20	2353
ta031_SSD100_P13	50	5	3488	ta032_SSD100_P13	50	5	3274
ta033_SSD100_P13	50	5	3887	ta034_SSD100_P13	50	5	3420
ta035_SSD100_P13	50	5	3949	ta036_SSD100_P13	50	5	3925
ta037_SSD100_P13	50	5	2838	ta038_SSD100_P13	50	5	3410
ta039_SSD100_P13	50	5	3411	ta040_SSD100_P13	50	5	3480
ta041_SSD100_P13	50	10	3509	ta042_SSD100_P13	50	10	4334
ta043_SSD100_P13	50	10	4512	ta044_SSD100_P13	50	10	4400
ta045_SSD100_P13	50	10	4180	ta046_SSD100_P13	50	10	4224
ta047_SSD100_P13	50	10	4018	ta048_SSD100_P13	50	10	4248
ta049_SSD100_P13	50	10	2187	ta050_SSD100_P13	50	10	4062
ta051_SSD100_P13	50	20	4470	ta052_SSD100_P13	50	20	4910
ta053_SSD100_P13	50	20	5040	ta054_SSD100_P13	50	20	4726
ta055_SSD100_P13	50	20	4919	ta056_SSD100_P13	50	20	5122
ta057_SSD100_P13	50	20	5297	ta058_SSD100_P13	50	20	5118
ta059_SSD100_P13	50	20	5002	ta060_SSD100_P13	50	20	5049
ta061_SSD100_P13	100	5	3744	ta062_SSD100_P13	100	5	5745
ta063_SSD100_P13	100	5	3710	ta064_SSD100_P13	100	5	2819
ta065_SSD100_P13	100	5	6604	ta066_SSD100_P13	100	5	7071
ta067_SSD100_P13	100	5	6463	ta068_SSD100_P13	100	5	6314
ta069_SSD100_P13	100	5	7417	ta070_SSD100_P13	100	5	6219
ta071_SSD100_P13	100	10	4521	ta072_SSD100_P13	100	10	8518
ta073_SSD100_P13	100	10	7463	ta074_SSD100_P13	100	10	7482

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta075_SSD100_P13	100	10	3840	ta076_SSD100_P13	100	10	8331
ta077_SSD100_P13	100	10	7812	ta078_SSD100_P13	100	10	7075
ta079_SSD100_P13	100	10	5704	ta080_SSD100_P13	100	10	7567
ta081_SSD100_P13	100	20	8373	ta082_SSD100_P13	100	20	9200
ta083_SSD100_P13	100	20	9530	ta084_SSD100_P13	100	20	8375
ta085_SSD100_P13	100	20	9305	ta086_SSD100_P13	100	20	9291
ta087_SSD100_P13	100	20	7706	ta088_SSD100_P13	100	20	7005
ta089_SSD100_P13	100	20	9192	ta090_SSD100_P13	100	20	8973
ta091_SSD100_P13	200	10	13326	ta092_SSD100_P13	200	10	15702
ta093_SSD100_P13	200	10	15038	ta094_SSD100_P13	200	10	15869
ta095_SSD100_P13	200	10	13322	ta096_SSD100_P13	200	10	12697
ta097_SSD100_P13	200	10	14024	ta098_SSD100_P13	200	10	16111
ta099_SSD100_P13	200	10	12668	ta100_SSD100_P13	200	10	14058
ta101_SSD100_P13	200	20	15866	ta102_SSD100_P13	200	20	18262
ta103_SSD100_P13	200	20	16106	ta104_SSD100_P13	200	20	17385
ta105_SSD100_P13	200	20	14527	ta106_SSD100_P13	200	20	15863
ta107_SSD100_P13	200	20	16586	ta108_SSD100_P13	200	20	16452
ta109_SSD100_P13	200	20	18231	ta110_SSD100_P13	200	20	17866

Tabla B.7 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD100_P13.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD125_P13	20	5	1203	ta002_SSD125_P13	20	5	1497
ta003_SSD125_P13	20	5	1604	ta004_SSD125_P13	20	5	1396
ta005_SSD125_P13	20	5	1657	ta006_SSD125_P13	20	5	1635
ta007_SSD125_P13	20	5	1490	ta008_SSD125_P13	20	5	1294
ta009_SSD125_P13	20	5	1630	ta010_SSD125_P13	20	5	1409
ta011_SSD125_P13	20	10	2252	ta012_SSD125_P13	20	10	1983
ta013_SSD125_P13	20	10	2017	ta014_SSD125_P13	20	10	1790
ta015_SSD125_P13	20	10	2145	ta016_SSD125_P13	20	10	1945
ta017_SSD125_P13	20	10	2101	ta018_SSD125_P13	20	10	2128
ta019_SSD125_P13	20	10	2085	ta020_SSD125_P13	20	10	1795
ta021_SSD125_P13	20	20	2137	ta022_SSD125_P13	20	20	2750

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta023_SSD125_P13	20	20	2560	ta024_SSD125_P13	20	20	2652
ta025_SSD125_P13	20	20	2040	ta026_SSD125_P13	20	20	2668
ta027_SSD125_P13	20	20	2628	ta028_SSD125_P13	20	20	2699
ta029_SSD125_P13	20	20	2953	ta030_SSD125_P13	20	20	2521
ta031_SSD125_P13	50	5	3778	ta032_SSD125_P13	50	5	3447
ta033_SSD125_P13	50	5	4272	ta034_SSD125_P13	50	5	3693
ta035_SSD125_P13	50	5	4172	ta036_SSD125_P13	50	5	4312
ta037_SSD125_P13	50	5	2912	ta038_SSD125_P13	50	5	3647
ta039_SSD125_P13	50	5	3655	ta040_SSD125_P13	50	5	3722
ta041_SSD125_P13	50	10	3837	ta042_SSD125_P13	50	10	4612
ta043_SSD125_P13	50	10	5068	ta044_SSD125_P13	50	10	4784
ta045_SSD125_P13	50	10	4554	ta046_SSD125_P13	50	10	4489
ta047_SSD125_P13	50	10	4328	ta048_SSD125_P13	50	10	4627
ta049_SSD125_P13	50	10	2374	ta050_SSD125_P13	50	10	4354
ta051_SSD125_P13	50	20	4746	ta052_SSD125_P13	50	20	5379
ta053_SSD125_P13	50	20	5516	ta054_SSD125_P13	50	20	5248
ta055_SSD125_P13	50	20	5318	ta056_SSD125_P13	50	20	5534
ta057_SSD125_P13	50	20	5722	ta058_SSD125_P13	50	20	5512
ta059_SSD125_P13	50	20	5352	ta060_SSD125_P13	50	20	5594
ta061_SSD125_P13	100	5	4109	ta062_SSD125_P13	100	5	5915
ta063_SSD125_P13	100	5	4038	ta064_SSD125_P13	100	5	3066
ta065_SSD125_P13	100	5	6913	ta066_SSD125_P13	100	5	7632
ta067_SSD125_P13	100	5	6815	ta068_SSD125_P13	100	5	6736
ta069_SSD125_P13	100	5	7850	ta070_SSD125_P13	100	5	6628
ta071_SSD125_P13	100	10	4905	ta072_SSD125_P13	100	10	9418
ta073_SSD125_P13	100	10	7957	ta074_SSD125_P13	100	10	8138
ta075_SSD125_P13	100	10	4378	ta076_SSD125_P13	100	10	9318
ta077_SSD125_P13	100	10	8648	ta078_SSD125_P13	100	10	7445
ta079_SSD125_P13	100	10	5896	ta080_SSD125_P13	100	10	7977
ta081_SSD125_P13	100	20	9156	ta082_SSD125_P13	100	20	9958
ta083_SSD125_P13	100	20	10311	ta084_SSD125_P13	100	20	9081
ta085_SSD125_P13	100	20	10221	ta086_SSD125_P13	100	20	10336
ta087_SSD125_P13	100	20	8230	ta088_SSD125_P13	100	20	7544
ta089_SSD125_P13	100	20	10045	ta090_SSD125_P13	100	20	10005
ta091_SSD125_P13	200	10	14183	ta092_SSD125_P13	200	10	16779

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta093_SSD125_P13	200	10	16468	ta094_SSD125_P13	200	10	17119
ta095_SSD125_P13	200	10	14048	ta096_SSD125_P13	200	10	13419
ta097_SSD125_P13	200	10	15154	ta098_SSD125_P13	200	10	17596
ta099_SSD125_P13	200	10	13388	ta100_SSD125_P13	200	10	14897
ta101_SSD125_P13	200	20	16874	ta102_SSD125_P13	200	20	20018
ta103_SSD125_P13	200	20	17470	ta104_SSD125_P13	200	20	18983
ta105_SSD125_P13	200	20	15402	ta106_SSD125_P13	200	20	17064
ta107_SSD125_P13	200	20	17878	ta108_SSD125_P13	200	20	17653
ta109_SSD125_P13	200	20	20035	ta110_SSD125_P13	200	20	19981

Tabla B.8 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD125_P13.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD10_P2	20	5	523	ta002_SSD10_P2	20	5	569
ta003_SSD10_P2	20	5	488	ta004_SSD10_P2	20	5	619
ta005_SSD10_P2	20	5	574	ta006_SSD10_P2	20	5	565
ta007_SSD10_P2	20	5	542	ta008_SSD10_P2	20	5	570
ta009_SSD10_P2	20	5	532	ta010_SSD10_P2	20	5	518
ta011_SSD10_P2	20	10	825	ta012_SSD10_P2	20	10	827
ta013_SSD10_P2	20	10	803	ta014_SSD10_P2	20	10	733
ta015_SSD10_P2	20	10	727	ta016_SSD10_P2	20	10	758
ta017_SSD10_P2	20	10	778	ta018_SSD10_P2	20	10	748
ta019_SSD10_P2	20	10	744	ta020_SSD10_P2	20	10	842
ta021_SSD10_P2	20	20	1287	ta022_SSD10_P2	20	20	1180
ta023_SSD10_P2	20	20	1248	ta024_SSD10_P2	20	20	1211
ta025_SSD10_P2	20	20	1262	ta026_SSD10_P2	20	20	1232
ta027_SSD10_P2	20	20	1290	ta028_SSD10_P2	20	20	1253
ta029_SSD10_P2	20	20	1270	ta030_SSD10_P2	20	20	1190
ta031_SSD10_P2	50	5	1142	ta032_SSD10_P2	50	5	1145
ta033_SSD10_P2	50	5	1142	ta034_SSD10_P2	50	5	1137
ta035_SSD10_P2	50	5	1264	ta036_SSD10_P2	50	5	1146
ta037_SSD10_P2	50	5	1135	ta038_SSD10_P2	50	5	1077
ta039_SSD10_P2	50	5	1194	ta040_SSD10_P2	50	5	1172

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta041_SSD10_P2	50	10	1393	ta042_SSD10_P2	50	10	1436
ta043_SSD10_P2	50	10	1411	ta044_SSD10_P2	50	10	1452
ta045_SSD10_P2	50	10	1519	ta046_SSD10_P2	50	10	1419
ta047_SSD10_P2	50	10	1526	ta048_SSD10_P2	50	10	1520
ta049_SSD10_P2	50	10	1512	ta050_SSD10_P2	50	10	1448
ta051_SSD10_P2	50	20	2110	ta052_SSD10_P2	50	20	2012
ta053_SSD10_P2	50	20	2005	ta054_SSD10_P2	50	20	2001
ta055_SSD10_P2	50	20	2001	ta056_SSD10_P2	50	20	2012
ta057_SSD10_P2	50	20	2058	ta058_SSD10_P2	50	20	2065
ta059_SSD10_P2	50	20	2045	ta060_SSD10_P2	50	20	2014
ta061_SSD10_P2	100	5	2236	ta062_SSD10_P2	100	5	2118
ta063_SSD10_P2	100	5	2257	ta064_SSD10_P2	100	5	2082
ta065_SSD10_P2	100	5	2219	ta066_SSD10_P2	100	5	2213
ta067_SSD10_P2	100	5	2246	ta068_SSD10_P2	100	5	2100
ta069_SSD10_P2	100	5	2265	ta070_SSD10_P2	100	5	2241
ta071_SSD10_P2	100	10	2623	ta072_SSD10_P2	100	10	2605
ta073_SSD10_P2	100	10	2573	ta074_SSD10_P2	100	10	2680
ta075_SSD10_P2	100	10	2440	ta076_SSD10_P2	100	10	2584
ta077_SSD10_P2	100	10	2658	ta078_SSD10_P2	100	10	2555
ta079_SSD10_P2	100	10	2711	ta080_SSD10_P2	100	10	2572
ta081_SSD10_P2	100	20	3280	ta082_SSD10_P2	100	20	3293
ta083_SSD10_P2	100	20	3287	ta084_SSD10_P2	100	20	3259
ta085_SSD10_P2	100	20	3223	ta086_SSD10_P2	100	20	3200
ta087_SSD10_P2	100	20	3302	ta088_SSD10_P2	100	20	3235
ta089_SSD10_P2	100	20	3198	ta090_SSD10_P2	100	20	3277
ta091_SSD10_P2	200	10	4828	ta092_SSD10_P2	200	10	4857
ta093_SSD10_P2	200	10	4866	ta094_SSD10_P2	200	10	4875
ta095_SSD10_P2	200	10	4870	ta096_SSD10_P2	200	10	4710
ta097_SSD10_P2	200	10	4905	ta098_SSD10_P2	200	10	4899
ta099_SSD10_P2	200	10	4841	ta100_SSD10_P2	200	10	4819
ta101_SSD10_P2	200	20	5611	ta102_SSD10_P2	200	20	5655
ta103_SSD10_P2	200	20	5684	ta104_SSD10_P2	200	20	5590
ta105_SSD10_P2	200	20	5616	ta106_SSD10_P2	200	20	5727
ta107_SSD10_P2	200	20	5633	ta108_SSD10_P2	200	20	5675

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta109_SSD10_P2	200	20	5789	ta110_SSD10_P2	200	20	5669

Tabla B.9 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD10_P2.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD50_P2	20	5	691	ta002_SSD50_P2	20	5	724
ta003_SSD50_P2	20	5	679	ta004_SSD50_P2	20	5	769
ta005_SSD50_P2	20	5	723	ta006_SSD50_P2	20	5	697
ta007_SSD50_P2	20	5	669	ta008_SSD50_P2	20	5	715
ta009_SSD50_P2	20	5	719	ta010_SSD50_P2	20	5	660
ta011_SSD50_P2	20	10	976	ta012_SSD50_P2	20	10	1016
ta013_SSD50_P2	20	10	1024	ta014_SSD50_P2	20	10	904
ta015_SSD50_P2	20	10	942	ta016_SSD50_P2	20	10	927
ta017_SSD50_P2	20	10	975	ta018_SSD50_P2	20	10	945
ta019_SSD50_P2	20	10	984	ta020_SSD50_P2	20	10	1004
ta021_SSD50_P2	20	20	1520	ta022_SSD50_P2	20	20	1399
ta023_SSD50_P2	20	20	1474	ta024_SSD50_P2	20	20	1419
ta025_SSD50_P2	20	20	1484	ta026_SSD50_P2	20	20	1440
ta027_SSD50_P2	20	20	1510	ta028_SSD50_P2	20	20	1467
ta029_SSD50_P2	20	20	1501	ta030_SSD50_P2	20	20	1405
ta031_SSD50_P2	50	5	1539	ta032_SSD50_P2	50	5	1579
ta033_SSD50_P2	50	5	1541	ta034_SSD50_P2	50	5	1554
ta035_SSD50_P2	50	5	1658	ta036_SSD50_P2	50	5	1530
ta037_SSD50_P2	50	5	1560	ta038_SSD50_P2	50	5	1497
ta039_SSD50_P2	50	5	1648	ta040_SSD50_P2	50	5	1593
ta041_SSD50_P2	50	10	1945	ta042_SSD50_P2	50	10	1893
ta043_SSD50_P2	50	10	1899	ta044_SSD50_P2	50	10	1950
ta045_SSD50_P2	50	10	1961	ta046_SSD50_P2	50	10	1917
ta047_SSD50_P2	50	10	2016	ta048_SSD50_P2	50	10	1998
ta049_SSD50_P2	50	10	1977	ta050_SSD50_P2	50	10	1975
ta051_SSD50_P2	50	20	2661	ta052_SSD50_P2	50	20	2550
ta053_SSD50_P2	50	20	2566	ta054_SSD50_P2	50	20	2527
ta055_SSD50_P2	50	20	2574	ta056_SSD50_P2	50	20	2574

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta057_SSD50_P2	50	20	2629	ta058_SSD50_P2	50	20	2633
ta059_SSD50_P2	50	20	2632	ta060_SSD50_P2	50	20	2606
ta061_SSD50_P2	100	5	3164	ta062_SSD50_P2	100	5	3063
ta063_SSD50_P2	100	5	3147	ta064_SSD50_P2	100	5	2937
ta065_SSD50_P2	100	5	3075	ta066_SSD50_P2	100	5	2982
ta067_SSD50_P2	100	5	3036	ta068_SSD50_P2	100	5	2966
ta069_SSD50_P2	100	5	3163	ta070_SSD50_P2	100	5	3108
ta071_SSD50_P2	100	10	3665	ta072_SSD50_P2	100	10	3597
ta073_SSD50_P2	100	10	3597	ta074_SSD50_P2	100	10	3609
ta075_SSD50_P2	100	10	3446	ta076_SSD50_P2	100	10	3605
ta077_SSD50_P2	100	10	3604	ta078_SSD50_P2	100	10	3606
ta079_SSD50_P2	100	10	3630	ta080_SSD50_P2	100	10	3498
ta081_SSD50_P2	100	20	4364	ta082_SSD50_P2	100	20	4409
ta083_SSD50_P2	100	20	4354	ta084_SSD50_P2	100	20	4392
ta085_SSD50_P2	100	20	4329	ta086_SSD50_P2	100	20	4319
ta087_SSD50_P2	100	20	4387	ta088_SSD50_P2	100	20	4339
ta089_SSD50_P2	100	20	4319	ta090_SSD50_P2	100	20	4386
ta091_SSD50_P2	200	10	6756	ta092_SSD50_P2	200	10	6815
ta093_SSD50_P2	200	10	6746	ta094_SSD50_P2	200	10	6773
ta095_SSD50_P2	200	10	6880	ta096_SSD50_P2	200	10	6729
ta097_SSD50_P2	200	10	6746	ta098_SSD50_P2	200	10	6838
ta099_SSD50_P2	200	10	6716	ta100_SSD50_P2	200	10	6833
ta101_SSD50_P2	200	20	7834	ta102_SSD50_P2	200	20	7798
ta103_SSD50_P2	200	20	7784	ta104_SSD50_P2	200	20	7793
ta105_SSD50_P2	200	20	7751	ta106_SSD50_P2	200	20	7878
ta107_SSD50_P2	200	20	7794	ta108_SSD50_P2	200	20	7897
ta109_SSD50_P2	200	20	8013	ta110_SSD50_P2	200	20	7718

Tabla B.10 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD50_P2.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD100_P2	20	5	883	ta002_SSD100_P2	20	5	889
ta003_SSD100_P2	20	5	865	ta004_SSD100_P2	20	5	935

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta005_SSD100_P2	20	5	882	ta006_SSD100_P2	20	5	853
ta007_SSD100_P2	20	5	857	ta008_SSD100_P2	20	5	886
ta009_SSD100_P2	20	5	863	ta010_SSD100_P2	20	5	813
ta011_SSD100_P2	20	10	1208	ta012_SSD100_P2	20	10	1226
ta013_SSD100_P2	20	10	1243	ta014_SSD100_P2	20	10	1102
ta015_SSD100_P2	20	10	1176	ta016_SSD100_P2	20	10	1137
ta017_SSD100_P2	20	10	1228	ta018_SSD100_P2	20	10	1182
ta019_SSD100_P2	20	10	1217	ta020_SSD100_P2	20	10	1204
ta021_SSD100_P2	20	20	1779	ta022_SSD100_P2	20	20	1666
ta023_SSD100_P2	20	20	1731	ta024_SSD100_P2	20	20	1676
ta025_SSD100_P2	20	20	1781	ta026_SSD100_P2	20	20	1699
ta027_SSD100_P2	20	20	1790	ta028_SSD100_P2	20	20	1710
ta029_SSD100_P2	20	20	1740	ta030_SSD100_P2	20	20	1682
ta031_SSD100_P2	50	5	2011	ta032_SSD100_P2	50	5	2010
ta033_SSD100_P2	50	5	1919	ta034_SSD100_P2	50	5	2024
ta035_SSD100_P2	50	5	2102	ta036_SSD100_P2	50	5	1982
ta037_SSD100_P2	50	5	2086	ta038_SSD100_P2	50	5	1931
ta039_SSD100_P2	50	5	2216	ta040_SSD100_P2	50	5	2048
ta041_SSD100_P2	50	10	2438	ta042_SSD100_P2	50	10	2475
ta043_SSD100_P2	50	10	2547	ta044_SSD100_P2	50	10	2498
ta045_SSD100_P2	50	10	2522	ta046_SSD100_P2	50	10	2475
ta047_SSD100_P2	50	10	2586	ta048_SSD100_P2	50	10	2545
ta049_SSD100_P2	50	10	2587	ta050_SSD100_P2	50	10	2480
ta051_SSD100_P2	50	20	3317	ta052_SSD100_P2	50	20	3175
ta053_SSD100_P2	50	20	3209	ta054_SSD100_P2	50	20	3190
ta055_SSD100_P2	50	20	3241	ta056_SSD100_P2	50	20	3224
ta057_SSD100_P2	50	20	3259	ta058_SSD100_P2	50	20	3294
ta059_SSD100_P2	50	20	3296	ta060_SSD100_P2	50	20	3237
ta061_SSD100_P2	100	5	3960	ta062_SSD100_P2	100	5	4011
ta063_SSD100_P2	100	5	4187	ta064_SSD100_P2	100	5	3872
ta065_SSD100_P2	100	5	4121	ta066_SSD100_P2	100	5	4048
ta067_SSD100_P2	100	5	4137	ta068_SSD100_P2	100	5	3801
ta069_SSD100_P2	100	5	4064	ta070_SSD100_P2	100	5	4046
ta071_SSD100_P2	100	10	4812	ta072_SSD100_P2	100	10	4692
ta073_SSD100_P2	100	10	4724	ta074_SSD100_P2	100	10	4842

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta075_SSD100_P2	100	10	4650	ta076_SSD100_P2	100	10	4731
ta077_SSD100_P2	100	10	4635	ta078_SSD100_P2	100	10	4675
ta079_SSD100_P2	100	10	4712	ta080_SSD100_P2	100	10	4709
ta081_SSD100_P2	100	20	5694	ta082_SSD100_P2	100	20	5636
ta083_SSD100_P2	100	20	5558	ta084_SSD100_P2	100	20	5678
ta085_SSD100_P2	100	20	5450	ta086_SSD100_P2	100	20	5579
ta087_SSD100_P2	100	20	5681	ta088_SSD100_P2	100	20	5674
ta089_SSD100_P2	100	20	5614	ta090_SSD100_P2	100	20	5667
ta091_SSD100_P2	200	10	8887	ta092_SSD100_P2	200	10	9017
ta093_SSD100_P2	200	10	8866	ta094_SSD100_P2	200	10	8988
ta095_SSD100_P2	200	10	9011	ta096_SSD100_P2	200	10	9026
ta097_SSD100_P2	200	10	9039	ta098_SSD100_P2	200	10	8803
ta099_SSD100_P2	200	10	8914	ta100_SSD100_P2	200	10	8900
ta101_SSD100_P2	200	20	10226	ta102_SSD100_P2	200	20	10243
ta103_SSD100_P2	200	20	10351	ta104_SSD100_P2	200	20	10170
ta105_SSD100_P2	200	20	10236	ta106_SSD100_P2	200	20	10224
ta107_SSD100_P2	200	20	10308	ta108_SSD100_P2	200	20	10227
ta109_SSD100_P2	200	20	10357	ta110_SSD100_P2	200	20	10182

Tabla B.11 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD100_P2.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD125_P2	20	5	891	ta002_SSD125_P2	20	5	955
ta003_SSD125_P2	20	5	930	ta004_SSD125_P2	20	5	1008
ta005_SSD125_P2	20	5	997	ta006_SSD125_P2	20	5	931
ta007_SSD125_P2	20	5	976	ta008_SSD125_P2	20	5	959
ta009_SSD125_P2	20	5	1002	ta010_SSD125_P2	20	5	872
ta011_SSD125_P2	20	10	1351	ta012_SSD125_P2	20	10	1325
ta013_SSD125_P2	20	10	1290	ta014_SSD125_P2	20	10	1231
ta015_SSD125_P2	20	10	1275	ta016_SSD125_P2	20	10	1239
ta017_SSD125_P2	20	10	1346	ta018_SSD125_P2	20	10	1288
ta019_SSD125_P2	20	10	1288	ta020_SSD125_P2	20	10	1305
ta021_SSD125_P2	20	20	1910	ta022_SSD125_P2	20	20	1780

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta023_SSD125_P2	20	20	1900	ta024_SSD125_P2	20	20	1803
ta025_SSD125_P2	20	20	1927	ta026_SSD125_P2	20	20	1819
ta027_SSD125_P2	20	20	1986	ta028_SSD125_P2	20	20	1853
ta029_SSD125_P2	20	20	1913	ta030_SSD125_P2	20	20	1813
ta031_SSD125_P2	50	5	2325	ta032_SSD125_P2	50	5	2229
ta033_SSD125_P2	50	5	2209	ta034_SSD125_P2	50	5	2192
ta035_SSD125_P2	50	5	2294	ta036_SSD125_P2	50	5	2161
ta037_SSD125_P2	50	5	2327	ta038_SSD125_P2	50	5	2145
ta039_SSD125_P2	50	5	2330	ta040_SSD125_P2	50	5	2237
ta041_SSD125_P2	50	10	2822	ta042_SSD125_P2	50	10	2716
ta043_SSD125_P2	50	10	2807	ta044_SSD125_P2	50	10	2769
ta045_SSD125_P2	50	10	2809	ta046_SSD125_P2	50	10	2750
ta047_SSD125_P2	50	10	2820	ta048_SSD125_P2	50	10	2805
ta049_SSD125_P2	50	10	2764	ta050_SSD125_P2	50	10	2741
ta051_SSD125_P2	50	20	3720	ta052_SSD125_P2	50	20	3493
ta053_SSD125_P2	50	20	3530	ta054_SSD125_P2	50	20	3536
ta055_SSD125_P2	50	20	3590	ta056_SSD125_P2	50	20	3529
ta057_SSD125_P2	50	20	3612	ta058_SSD125_P2	50	20	3586
ta059_SSD125_P2	50	20	3551	ta060_SSD125_P2	50	20	3583
ta061_SSD125_P2	100	5	4603	ta062_SSD125_P2	100	5	4458
ta063_SSD125_P2	100	5	4495	ta064_SSD125_P2	100	5	4249
ta065_SSD125_P2	100	5	4418	ta066_SSD125_P2	100	5	4387
ta067_SSD125_P2	100	5	4485	ta068_SSD125_P2	100	5	4375
ta069_SSD125_P2	100	5	4578	ta070_SSD125_P2	100	5	4535
ta071_SSD125_P2	100	10	5371	ta072_SSD125_P2	100	10	5298
ta073_SSD125_P2	100	10	5208	ta074_SSD125_P2	100	10	5410
ta075_SSD125_P2	100	10	5003	ta076_SSD125_P2	100	10	5223
ta077_SSD125_P2	100	10	5165	ta078_SSD125_P2	100	10	5246
ta079_SSD125_P2	100	10	5216	ta080_SSD125_P2	100	10	5186
ta081_SSD125_P2	100	20	6314	ta082_SSD125_P2	100	20	6307
ta083_SSD125_P2	100	20	6282	ta084_SSD125_P2	100	20	6221
ta085_SSD125_P2	100	20	6330	ta086_SSD125_P2	100	20	6295
ta087_SSD125_P2	100	20	6383	ta088_SSD125_P2	100	20	6319
ta089_SSD125_P2	100	20	6219	ta090_SSD125_P2	100	20	6247
ta091_SSD125_P2	200	10	10070	ta092_SSD125_P2	200	10	10090

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta093_SSD125_P2	200	10	9833	ta094_SSD125_P2	200	10	10009
ta095_SSD125_P2	200	10	9934	ta096_SSD125_P2	200	10	10029
ta097_SSD125_P2	200	10	9908	ta098_SSD125_P2	200	10	10046
ta099_SSD125_P2	200	10	9942	ta100_SSD125_P2	200	10	9901
ta101_SSD125_P2	200	20	11298	ta102_SSD125_P2	200	20	11404
ta103_SSD125_P2	200	20	11436	ta104_SSD125_P2	200	20	11583
ta105_SSD125_P2	200	20	11465	ta106_SSD125_P2	200	20	11341
ta107_SSD125_P2	200	20	11504	ta108_SSD125_P2	200	20	11529
ta109_SSD125_P2	200	20	11508	ta110_SSD125_P2	200	20	11417

Tabla B.12 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD125_P2.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD10_P3	20	5	341	ta002_SSD10_P3	20	5	351
ta003_SSD10_P3	20	5	336	ta004_SSD10_P3	20	5	352
ta005_SSD10_P3	20	5	323	ta006_SSD10_P3	20	5	348
ta007_SSD10_P3	20	5	349	ta008_SSD10_P3	20	5	332
ta009_SSD10_P3	20	5	301	ta010_SSD10_P3	20	5	322
ta011_SSD10_P3	20	10	557	ta012_SSD10_P3	20	10	541
ta013_SSD10_P3	20	10	511	ta014_SSD10_P3	20	10	461
ta015_SSD10_P3	20	10	513	ta016_SSD10_P3	20	10	509
ta017_SSD10_P3	20	10	542	ta018_SSD10_P3	20	10	513
ta019_SSD10_P3	20	10	508	ta020_SSD10_P3	20	10	533
ta021_SSD10_P3	20	20	933	ta022_SSD10_P3	20	20	786
ta023_SSD10_P3	20	20	863	ta024_SSD10_P3	20	20	782
ta025_SSD10_P3	20	20	838	ta026_SSD10_P3	20	20	859
ta027_SSD10_P3	20	20	888	ta028_SSD10_P3	20	20	869
ta029_SSD10_P3	20	20	871	ta030_SSD10_P3	20	20	878
ta031_SSD10_P3	50	5	678	ta032_SSD10_P3	50	5	678
ta033_SSD10_P3	50	5	673	ta034_SSD10_P3	50	5	680
ta035_SSD10_P3	50	5	661	ta036_SSD10_P3	50	5	700
ta037_SSD10_P3	50	5	642	ta038_SSD10_P3	50	5	656
ta039_SSD10_P3	50	5	737	ta040_SSD10_P3	50	5	684

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta041_SSD10_P3	50	10	875	ta042_SSD10_P3	50	10	867
ta043_SSD10_P3	50	10	939	ta044_SSD10_P3	50	10	837
ta045_SSD10_P3	50	10	941	ta046_SSD10_P3	50	10	940
ta047_SSD10_P3	50	10	942	ta048_SSD10_P3	50	10	947
ta049_SSD10_P3	50	10	899	ta050_SSD10_P3	50	10	926
ta051_SSD10_P3	50	20	1345	ta052_SSD10_P3	50	20	2168
ta053_SSD10_P3	50	20	1321	ta054_SSD10_P3	50	20	1324
ta055_SSD10_P3	50	20	1290	ta056_SSD10_P3	50	20	1325
ta057_SSD10_P3	50	20	1342	ta058_SSD10_P3	50	20	1284
ta059_SSD10_P3	50	20	1297	ta060_SSD10_P3	50	20	1316
ta061_SSD10_P3	100	5	1289	ta062_SSD10_P3	100	5	1297
ta063_SSD10_P3	100	5	1319	ta064_SSD10_P3	100	5	1263
ta065_SSD10_P3	100	5	1269	ta066_SSD10_P3	100	5	1260
ta067_SSD10_P3	100	5	1284	ta068_SSD10_P3	100	5	1236
ta069_SSD10_P3	100	5	1314	ta070_SSD10_P3	100	5	1252
ta071_SSD10_P3	100	10	1573	ta072_SSD10_P3	100	10	1532
ta073_SSD10_P3	100	10	1616	ta074_SSD10_P3	100	10	1555
ta075_SSD10_P3	100	10	1451	ta076_SSD10_P3	100	10	1554
ta077_SSD10_P3	100	10	1596	ta078_SSD10_P3	100	10	1534
ta079_SSD10_P3	100	10	1548	ta080_SSD10_P3	100	10	1519
ta081_SSD10_P3	100	20	2027	ta082_SSD10_P3	100	20	2044
ta083_SSD10_P3	100	20	2053	ta084_SSD10_P3	100	20	1977
ta085_SSD10_P3	100	20	1990	ta086_SSD10_P3	100	20	1992
ta087_SSD10_P3	100	20	2053	ta088_SSD10_P3	100	20	2032
ta089_SSD10_P3	100	20	2077	ta090_SSD10_P3	100	20	2072
ta091_SSD10_P3	200	10	2734	ta092_SSD10_P3	200	10	2749
ta093_SSD10_P3	200	10	2770	ta094_SSD10_P3	200	10	2790
ta095_SSD10_P3	200	10	2774	ta096_SSD10_P3	200	10	2772
ta097_SSD10_P3	200	10	2781	ta098_SSD10_P3	200	10	2791
ta099_SSD10_P3	200	10	2724	ta100_SSD10_P3	200	10	2850
ta101_SSD10_P3	200	20	3483	ta102_SSD10_P3	200	20	3361
ta103_SSD10_P3	200	20	3460	ta104_SSD10_P3	200	20	3410
ta105_SSD10_P3	200	20	3459	ta106_SSD10_P3	200	20	3461
ta107_SSD10_P3	200	20	3481	ta108_SSD10_P3	200	20	3442

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta109_SSD10_P3	200	20	3408	ta110_SSD10_P3	200	20	3457

Tabla B.13 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD10_P3.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD50_P3	20	5	446	ta002_SSD50_P3	20	5	459
ta003_SSD50_P3	20	5	440	ta004_SSD50_P3	20	5	461
ta005_SSD50_P3	20	5	431	ta006_SSD50_P3	20	5	440
ta007_SSD50_P3	20	5	470	ta008_SSD50_P3	20	5	429
ta009_SSD50_P3	20	5	434	ta010_SSD50_P3	20	5	412
ta011_SSD50_P3	20	10	673	ta012_SSD50_P3	20	10	652
ta013_SSD50_P3	20	10	640	ta014_SSD50_P3	20	10	580
ta015_SSD50_P3	20	10	648	ta016_SSD50_P3	20	10	637
ta017_SSD50_P3	20	10	675	ta018_SSD50_P3	20	10	621
ta019_SSD50_P3	20	10	665	ta020_SSD50_P3	20	10	646
ta021_SSD50_P3	20	20	1070	ta022_SSD50_P3	20	20	928
ta023_SSD50_P3	20	20	1018	ta024_SSD50_P3	20	20	926
ta025_SSD50_P3	20	20	1004	ta026_SSD50_P3	20	20	999
ta027_SSD50_P3	20	20	1042	ta028_SSD50_P3	20	20	1020
ta029_SSD50_P3	20	20	1020	ta030_SSD50_P3	20	20	1012
ta031_SSD50_P3	50	5	982	ta032_SSD50_P3	50	5	955
ta033_SSD50_P3	50	5	944	ta034_SSD50_P3	50	5	977
ta035_SSD50_P3	50	5	991	ta036_SSD50_P3	50	5	951
ta037_SSD50_P3	50	5	945	ta038_SSD50_P3	50	5	932
ta039_SSD50_P3	50	5	1006	ta040_SSD50_P3	50	5	965
ta041_SSD50_P3	50	10	1215	ta042_SSD50_P3	50	10	1215
ta043_SSD50_P3	50	10	1257	ta044_SSD50_P3	50	10	1174
ta045_SSD50_P3	50	10	1235	ta046_SSD50_P3	50	10	1288
ta047_SSD50_P3	50	10	1353	ta048_SSD50_P3	50	10	1284
ta049_SSD50_P3	50	10	1243	ta050_SSD50_P3	50	10	1261
ta051_SSD50_P3	50	20	1746	ta052_SSD50_P3	50	20	1658
ta053_SSD50_P3	50	20	1715	ta054_SSD50_P3	50	20	1718
ta055_SSD50_P3	50	20	1729	ta056_SSD50_P3	50	20	1713

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta057_SSD50_P3	50	20	1747	ta058_SSD50_P3	50	20	1678
ta059_SSD50_P3	50	20	1714	ta060_SSD50_P3	50	20	1704
ta061_SSD50_P3	100	5	1926	ta062_SSD50_P3	100	5	1903
ta063_SSD50_P3	100	5	1953	ta064_SSD50_P3	100	5	1891
ta065_SSD50_P3	100	5	1838	ta066_SSD50_P3	100	5	1860
ta067_SSD50_P3	100	5	1888	ta068_SSD50_P3	100	5	1856
ta069_SSD50_P3	100	5	1923	ta070_SSD50_P3	100	5	1790
ta071_SSD50_P3	100	10	2336	ta072_SSD50_P3	100	10	2184
ta073_SSD50_P3	100	10	2278	ta074_SSD50_P3	100	10	2214
ta075_SSD50_P3	100	10	2182	ta076_SSD50_P3	100	10	2253
ta077_SSD50_P3	100	10	2272	ta078_SSD50_P3	100	10	2205
ta079_SSD50_P3	100	10	2262	ta080_SSD50_P3	100	10	2222
ta081_SSD50_P3	100	20	2812	ta082_SSD50_P3	100	20	2784
ta083_SSD50_P3	100	20	2832	ta084_SSD50_P3	100	20	2736
ta085_SSD50_P3	100	20	2772	ta086_SSD50_P3	100	20	2789
ta087_SSD50_P3	100	20	2810	ta088_SSD50_P3	100	20	2845
ta089_SSD50_P3	100	20	2916	ta090_SSD50_P3	100	20	2853
ta091_SSD50_P3	200	10	4135	ta092_SSD50_P3	200	10	4148
ta093_SSD50_P3	200	10	4165	ta094_SSD50_P3	200	10	4164
ta095_SSD50_P3	200	10	4180	ta096_SSD50_P3	200	10	4078
ta097_SSD50_P3	200	10	4170	ta098_SSD50_P3	200	10	4135
ta099_SSD50_P3	200	10	4103	ta100_SSD50_P3	200	10	4209
ta101_SSD50_P3	200	20	4918	ta102_SSD50_P3	200	20	4854
ta103_SSD50_P3	200	20	4887	ta104_SSD50_P3	200	20	4895
ta105_SSD50_P3	200	20	4879	ta106_SSD50_P3	200	20	4937
ta107_SSD50_P3	200	20	4944	ta108_SSD50_P3	200	20	4926
ta109_SSD50_P3	200	20	4926	ta110_SSD50_P3	200	20	4874

Tabla B.14 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD50_P3.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD100_P3	20	5	543	ta002_SSD100_P3	20	5	564
ta003_SSD100_P3	20	5	539	ta004_SSD100_P3	20	5	571

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta005_SSD100_P3	20	5	569	ta006_SSD100_P3	20	5	538
ta007_SSD100_P3	20	5	603	ta008_SSD100_P3	20	5	531
ta009_SSD100_P3	20	5	567	ta010_SSD100_P3	20	5	509
ta011_SSD100_P3	20	10	836	ta012_SSD100_P3	20	10	798
ta013_SSD100_P3	20	10	793	ta014_SSD100_P3	20	10	717
ta015_SSD100_P3	20	10	819	ta016_SSD100_P3	20	10	769
ta017_SSD100_P3	20	10	861	ta018_SSD100_P3	20	10	760
ta019_SSD100_P3	20	10	794	ta020_SSD100_P3	20	10	794
ta021_SSD100_P3	20	20	1239	ta022_SSD100_P3	20	20	1113
ta023_SSD100_P3	20	20	1207	ta024_SSD100_P3	20	20	1112
ta025_SSD100_P3	20	20	1181	ta026_SSD100_P3	20	20	1184
ta027_SSD100_P3	20	20	1234	ta028_SSD100_P3	20	20	1204
ta029_SSD100_P3	20	20	1200	ta030_SSD100_P3	20	20	1191
ta031_SSD100_P3	50	5	1296	ta032_SSD100_P3	50	5	1256
ta033_SSD100_P3	50	5	1282	ta034_SSD100_P3	50	5	1275
ta035_SSD100_P3	50	5	1283	ta036_SSD100_P3	50	5	1248
ta037_SSD100_P3	50	5	1254	ta038_SSD100_P3	50	5	1248
ta039_SSD100_P3	50	5	1322	ta040_SSD100_P3	50	5	1248
ta041_SSD100_P3	50	10	1552	ta042_SSD100_P3	50	10	1555
ta043_SSD100_P3	50	10	1616	ta044_SSD100_P3	50	10	1570
ta045_SSD100_P3	50	10	1636	ta046_SSD100_P3	50	10	1658
ta047_SSD100_P3	50	10	1685	ta048_SSD100_P3	50	10	1614
ta049_SSD100_P3	50	10	1639	ta050_SSD100_P3	50	10	1612
ta051_SSD100_P3	50	20	2236	ta052_SSD100_P3	50	20	2106
ta053_SSD100_P3	50	20	2170	ta054_SSD100_P3	50	20	2145
ta055_SSD100_P3	50	20	2166	ta056_SSD100_P3	50	20	2172
ta057_SSD100_P3	50	20	2251	ta058_SSD100_P3	50	20	2100
ta059_SSD100_P3	50	20	2158	ta060_SSD100_P3	50	20	2159
ta061_SSD100_P3	100	5	2526	ta062_SSD100_P3	100	5	2518
ta063_SSD100_P3	100	5	2529	ta064_SSD100_P3	100	5	2531
ta065_SSD100_P3	100	5	2459	ta066_SSD100_P3	100	5	2494
ta067_SSD100_P3	100	5	2545	ta068_SSD100_P3	100	5	2388
ta069_SSD100_P3	100	5	2492	ta070_SSD100_P3	100	5	2499
ta071_SSD100_P3	100	10	3027	ta072_SSD100_P3	100	10	2945
ta073_SSD100_P3	100	10	3002	ta074_SSD100_P3	100	10	2981

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta075_SSD100_P3	100	10	2883	ta076_SSD100_P3	100	10	2993
ta077_SSD100_P3	100	10	3024	ta078_SSD100_P3	100	10	3005
ta079_SSD100_P3	100	10	3008	ta080_SSD100_P3	100	10	2965
ta081_SSD100_P3	100	20	3697	ta082_SSD100_P3	100	20	3706
ta083_SSD100_P3	100	20	3737	ta084_SSD100_P3	100	20	3660
ta085_SSD100_P3	100	20	3615	ta086_SSD100_P3	100	20	3666
ta087_SSD100_P3	100	20	3702	ta088_SSD100_P3	100	20	3703
ta089_SSD100_P3	100	20	3757	ta090_SSD100_P3	100	20	3701
ta091_SSD100_P3	200	10	5552	ta092_SSD100_P3	200	10	5599
ta093_SSD100_P3	200	10	5583	ta094_SSD100_P3	200	10	5507
ta095_SSD100_P3	200	10	5730	ta096_SSD100_P3	200	10	5466
ta097_SSD100_P3	200	10	5609	ta098_SSD100_P3	200	10	5526
ta099_SSD100_P3	200	10	5550	ta100_SSD100_P3	200	10	5580
ta101_SSD100_P3	200	20	6522	ta102_SSD100_P3	200	20	6465
ta103_SSD100_P3	200	20	6492	ta104_SSD100_P3	200	20	6504
ta105_SSD100_P3	200	20	6596	ta106_SSD100_P3	200	20	6473
ta107_SSD100_P3	200	20	6609	ta108_SSD100_P3	200	20	6526
ta109_SSD100_P3	200	20	6589	ta110_SSD100_P3	200	20	6531

Tabla B.15 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD100_P3.

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta001_SSD125_P3	20	5	600	ta002_SSD125_P3	20	5	609
ta003_SSD125_P3	20	5	617	ta004_SSD125_P3	20	5	610
ta005_SSD125_P3	20	5	605	ta006_SSD125_P3	20	5	588
ta007_SSD125_P3	20	5	647	ta008_SSD125_P3	20	5	586
ta009_SSD125_P3	20	5	592	ta010_SSD125_P3	20	5	555
ta011_SSD125_P3	20	10	900	ta012_SSD125_P3	20	10	858
ta013_SSD125_P3	20	10	880	ta014_SSD125_P3	20	10	782
ta015_SSD125_P3	20	10	866	ta016_SSD125_P3	20	10	827
ta017_SSD125_P3	20	10	911	ta018_SSD125_P3	20	10	831
ta019_SSD125_P3	20	10	880	ta020_SSD125_P3	20	10	861
ta021_SSD125_P3	20	20	1309	ta022_SSD125_P3	20	20	1206

Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta023_SSD125_P3	20	20	1320	ta024_SSD125_P3	20	20	1209
ta025_SSD125_P3	20	20	1314	ta026_SSD125_P3	20	20	1264
ta027_SSD125_P3	20	20	1345	ta028_SSD125_P3	20	20	1293
ta029_SSD125_P3	20	20	1310	ta030_SSD125_P3	20	20	1269
ta031_SSD125_P3	50	5	1426	ta032_SSD125_P3	50	5	1362
ta033_SSD125_P3	50	5	1370	ta034_SSD125_P3	50	5	1368
ta035_SSD125_P3	50	5	1409	ta036_SSD125_P3	50	5	1372
ta037_SSD125_P3	50	5	1399	ta038_SSD125_P3	50	5	1387
ta039_SSD125_P3	50	5	1443	ta040_SSD125_P3	50	5	1370
ta041_SSD125_P3	50	10	1830	ta042_SSD125_P3	50	10	1732
ta043_SSD125_P3	50	10	1732	ta044_SSD125_P3	50	10	1732
ta045_SSD125_P3	50	10	1772	ta046_SSD125_P3	50	10	1801
ta047_SSD125_P3	50	10	1858	ta048_SSD125_P3	50	10	1778
ta049_SSD125_P3	50	10	1832	ta050_SSD125_P3	50	10	1760
ta051_SSD125_P3	50	20	2424	ta052_SSD125_P3	50	20	2331
ta053_SSD125_P3	50	20	2344	ta054_SSD125_P3	50	20	2352
ta055_SSD125_P3	50	20	2469	ta056_SSD125_P3	50	20	2373
ta057_SSD125_P3	50	20	2445	ta058_SSD125_P3	50	20	2307
ta059_SSD125_P3	50	20	2398	ta060_SSD125_P3	50	20	2370
ta061_SSD125_P3	100	5	2824	ta062_SSD125_P3	100	5	2778
ta063_SSD125_P3	100	5	2781	ta064_SSD125_P3	100	5	2780
ta065_SSD125_P3	100	5	2802	ta066_SSD125_P3	100	5	2721
ta067_SSD125_P3	100	5	2845	ta068_SSD125_P3	100	5	2694
ta069_SSD125_P3	100	5	2885	ta070_SSD125_P3	100	5	2649
ta071_SSD125_P3	100	10	3318	ta072_SSD125_P3	100	10	3198
ta073_SSD125_P3	100	10	3325	ta074_SSD125_P3	100	10	3324
ta075_SSD125_P3	100	10	3235	ta076_SSD125_P3	100	10	3307
ta077_SSD125_P3	100	10	3292	ta078_SSD125_P3	100	10	3266
ta079_SSD125_P3	100	10	3387	ta080_SSD125_P3	100	10	3322
ta081_SSD125_P3	100	20	4102	ta082_SSD125_P3	100	20	4137
ta083_SSD125_P3	100	20	4154	ta084_SSD125_P3	100	20	4081
ta085_SSD125_P3	100	20	4064	ta086_SSD125_P3	100	20	4035
ta087_SSD125_P3	100	20	4098	ta088_SSD125_P3	100	20	4106
ta089_SSD125_P3	100	20	4190	ta090_SSD125_P3	100	20	4115
ta091_SSD125_P3	200	10	6300	ta092_SSD125_P3	200	10	6266

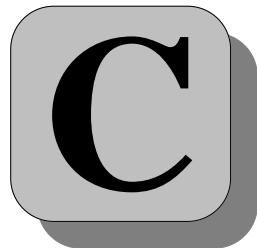
Problema	<i>n</i>	<i>m</i>	solución	Problema	<i>n</i>	<i>m</i>	solución
ta093_SSD125_P3	200	10	6310	ta094_SSD125_P3	200	10	6356
ta095_SSD125_P3	200	10	6291	ta096_SSD125_P3	200	10	6172
ta097_SSD125_P3	200	10	6395	ta098_SSD125_P3	200	10	6134
ta099_SSD125_P3	200	10	6247	ta100_SSD125_P3	200	10	6306
ta101_SSD125_P3	200	20	7243	ta102_SSD125_P3	200	20	7232
ta103_SSD125_P3	200	20	7265	ta104_SSD125_P3	200	20	7233
ta105_SSD125_P3	200	20	7197	ta106_SSD125_P3	200	20	7306
ta107_SSD125_P3	200	20	7380	ta108_SSD125_P3	200	20	7188
ta109_SSD125_P3	200	20	7267	ta110_SSD125_P3	200	20	7238

Tabla B.16 – Número de trabajos (*n*), máquinas (*m*), y mejores soluciones para el grupo de problemas SSD125_P3.

B.3. Problemas reales de planificación de la producción

En esta sección se presentan los 10 problemas de planificación de la producción reales obtenidos de la empresa Cerypsa Cerámicas S.A. Estos problemas se pueden clasificar como $FH2, ((R3^{(i)})_{i=1}^{(2)})/S_{sd}, M_j/C_{max}$, dado que tienen dos etapas y tres máquinas no relacionadas por etapa. Para representar las restricciones de uso de máquinas tenemos que cuando un $p_{i,j}$ es igual a -1 significa que esa máquina *l* de la etapa *i* no puede procesar el trabajo *j*. En los ejemplos se han omitido las referencias a los productos. Los tiempos de proceso van en función del tamaño de lote y de la velocidad de las máquinas con respecto al producto en cuestión. Todos los tiempos de proceso, así como los tiempos de cambio están expresados en minutos. Estos problemas se pueden encontrar en el directorio “Problemas reales” del CD-ROM anexo (ver Anexo C).

ANEXO



PROTOTIPO DE SOFTWARE: CÓDIGO FUENTE

A lo largo de la presente Tesis Doctoral se han realizado evaluaciones comparativas de varios métodos de programación de la producción, así como nuevos algoritmos genéticos para varios problemas distintos. Todo este trabajo, como se ha comentado, se ha implementado en Borland Delphi[©] versión 6.0. dentro de un programa que llamamos ProdPlanner.

Este prototipo de software, es capaz de resolver los siguientes problemas de tipo taller de flujo mediante la aplicación de distintos métodos heurísticos y metaheurísticos:

- $F/prmu/C_{max}$ o taller de flujo de permutación.
- $F/S_{sd}, prmu/C_{max}$ o taller de flujo de permutación con tiempos de cambio de partida dependientes de la secuencia.
- $FHm, ((PM^{(i)})_{i=1}^{(m)})//C_{max}$ o taller de flujo híbrido con máquinas idénticas en cada etapa.
- $FHm, ((PM^{(i)})_{i=1}^{(m)})/S_{sd}/C_{max}$ o taller de flujo híbrido con máquinas

idénticas en cada etapa y tiempos de cambio de partida dependientes de la secuencia.

- $FHm, ((RM^{(i)})_{i=1}^{(m)}) // C_{max}$ o taller de flujo híbrido con máquinas no relacionadas en cada etapa.
- $FHm, ((RM^{(i)})_{i=1}^{(m)}) / S_{sd} / C_{max}$ o taller de flujo híbrido con máquinas no relacionadas en cada etapa y tiempos de cambio de partida dependientes de la secuencia.
- $FHm, ((RM^{(i)})_{i=1}^{(m)}) / S_{sd}, M_j / C_{max}$ o taller de flujo híbrido con máquinas no relacionadas en cada etapa, tiempos de cambio de partida dependientes de la secuencia y elegibilidad de máquinas.

El software implementa un total de 40 métodos entre heurísticas, metaheurísticas y otros algoritmos. Es capaz de leer los ficheros de datos con el formato que se ha especificado en los Capítulos 3, 5 y 6.

El programa incorpora una funcionalidad básica de manejo de problemas y de secuencias (ver Figura C.1) y es posible indicar los criterios de parada (Figura C.2). Se ha implementado un completo evaluador de métodos que permite seleccionar los métodos que se quieren aplicar al conjunto de problemas de ejemplo cargados en modo de proceso por lote (Figura C.3), los resultados se muestran en hojas de cálculo Excel con varias fichas, una para el objetivo de optimización y otra para el tiempo de CPU utilizado. Aunque el programa se encuentra en fase de prototipo, es posible obtener un completo diagrama de Gantt con la programación de todas las tareas (Figura C.4). El programa se ha modularizado de manera que añadir nuevos métodos o considerar variaciones de otros problemas de tipo taller de flujo, así como considerar otros criterios de optimización resulta muy sencillo.

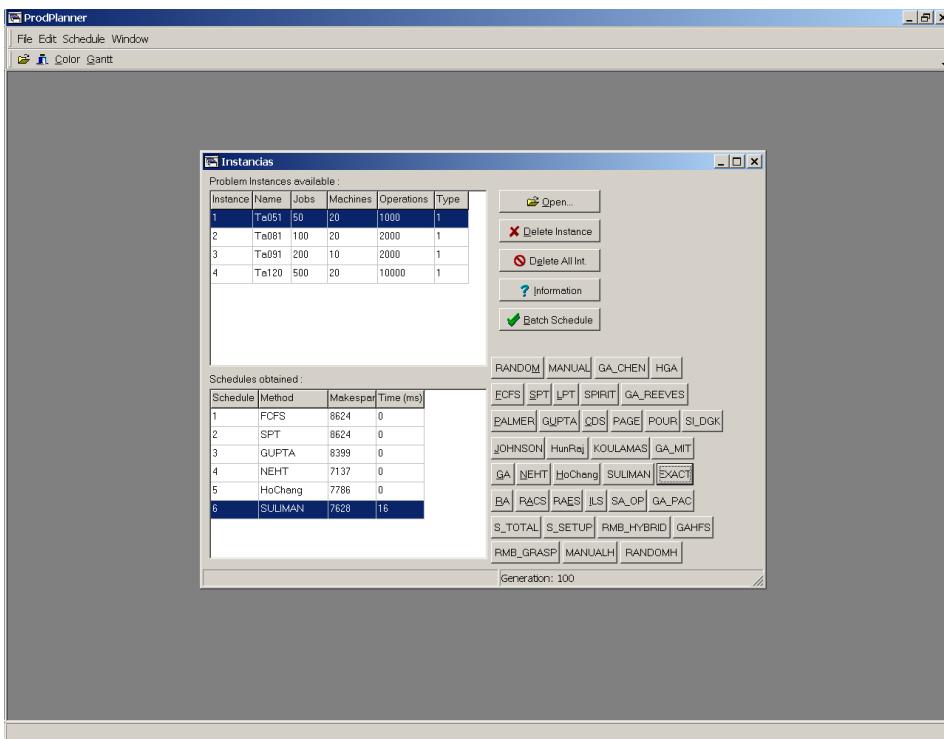


Figura C.1 – Herramienta de programación de la producción ProdPlanner. Pantalla de gestión de problemas ejemplo y secuencias.

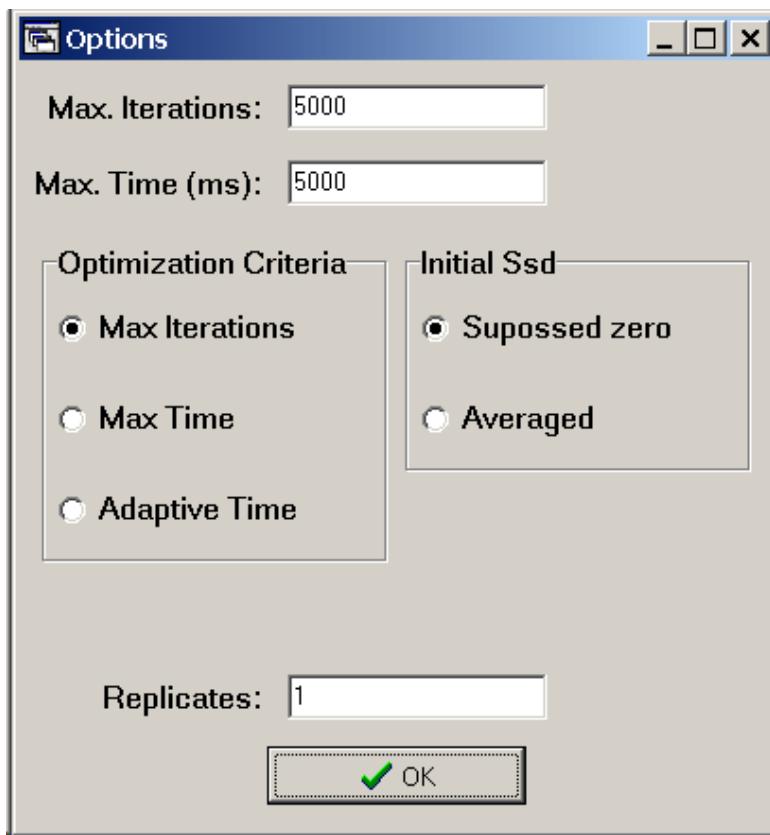


Figura C.2 – Herramienta de programación de la producción ProdPlanner. Pantalla de configuración y selección de criterio de parada.

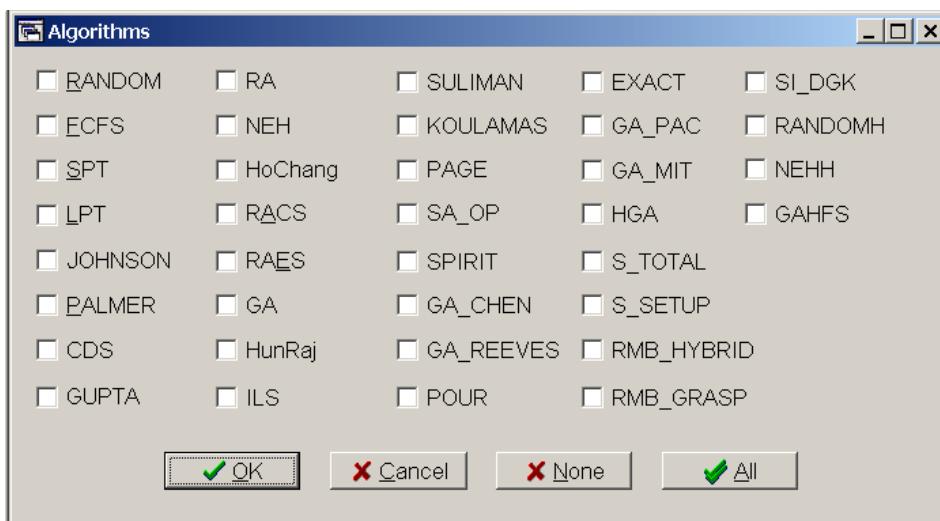


Figura C.3 – Herramienta de programación de la producción ProdPlanner. Pantalla de selección de métodos de resolución.

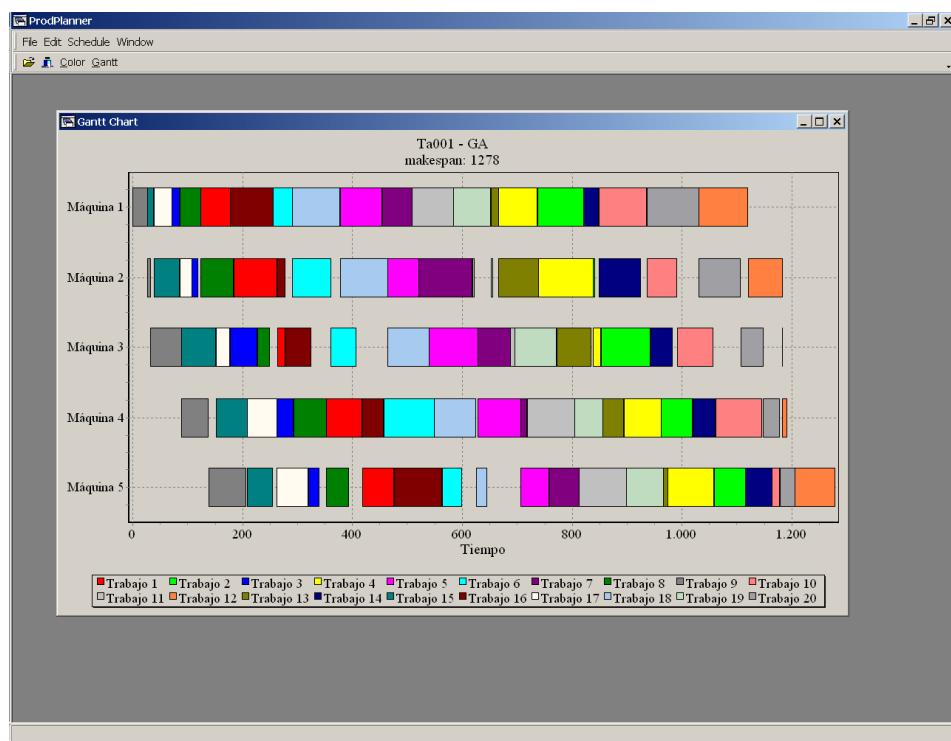


Figura C.4 – Herramienta de programación de la producción ProdPlanner. Diagrama de Gantt con la programación de las tareas para un ejemplo.

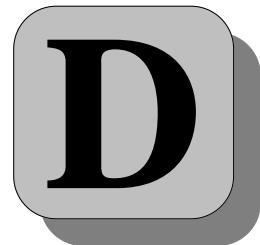
El código fuente completo es demasiado extenso como para incluirlo aquí, dado que ocuparía demasiado espacio. Como solución, hemos adjuntado un CD-ROM con las partes fundamentales de ProdPlanner. Se han listado los distintos ficheros que forman parte del programa en archivos PDF que se pueden consultar con la versión 5.0 del programa Acrobat Reader disponible gratuitamente en <http://www.adobe.com>. Concretamente, el contenido del CD es el siguiente:

Archivo	Número de páginas	Líneas de código	Comentarios
Estructuras.pdf	5	223	Definición de estructuras de datos para el programa
FInstancias.pdf	4	250	Control de los problemas de ejemplo y secuencias resultado
Funciones.pdf	13	816	Funciones de cálculo del C_{max} y funciones auxiliares (QuickSort)
Gantt.pdf	10	604	Control y dibujo de los diagramas de Gantt
Genetic.pdf	42	2467	Definición de los algoritmos GA, HGA, GA_{sd} , HGA_{sd} y GA_H , así como GA-Chen, y GAPAC
Genetic_MIT.pdf	5	245	Definición del algoritmo GAMIT
Genetic_Reeves.pdf	5	270	Definición del algoritmo GAReev
Heuristics.pdf	57	3477	Especificación y definición de todos los métodos heurísticos y metaheurísticos para los problemas $F/prmu/C_{max}$ y $F/S_{sd}, prmu/C_{max}$, así como adaptaciones
Info.pdf	3	163	Funciones y pantalla de información sobre problema de ejemplo
Main.pdf	20	1264	Pantalla inicial, cuerpo principal de la aplicación
Prodplanner.exe			Ejecutable ProdPlanner
Select.pdf	3	190	Pantalla de selección de heurísticas y metaheurísticas para ejecución en modo por lotes

Tabla C.1 – Contenido del CD-ROM anexo.

También se han incluido en el CD el conjunto de pruebas de Taillard (1993) dentro del directorio “Problemas ejemplo Taillard”. Las mejores soluciones para estos 120 problemas pueden consultarse en la Tabla 3.8 del Capítulo 3. En el directorio “Problemas ejemplo OR-Library” se han recogido los grupos de problemas de ejemplo accesibles en la OR-Library (<http://mscmga.ms>).

ic.ac.uk/jeb/orlib/flowshopinfo.html). El directorio “Problemas ejemplo artículos varios” contiene problemas de ejemplo que aparecen en varios artículos de métodos que se han implementado en ProdPlanner y que se han descrito en los Capítulos 3, 4 y 5. De igual manera, en el fichero “Problemas de ejemplo SSD10, SSD50, SSD100 y SSD125.zip” se han comprimido, en formato ZIP los 480 problemas de ejemplo que constituyen los cuatro grupos de problemas SSD10, SSD50, SSD100 y SSD125 que se presentaron en el Capítulo 5 y cuyas mejores soluciones se han detallado en la Sección B.1 del Anexo B. Por último, el directorio “Problemas reales” contiene los 10 problemas reales utilizados en el Capítulo 6, que se han obtenido de la empresa Cerypsa Cerámicas S.A. También se incluyen las secuencias manuales realizadas por los responsables de la planificación de la producción y que se cargan con el método MANUALH del software.



ANEXO

TIPOS DE BALDOSAS CERÁMICAS

D.1. Azulejo

Se denomina azulejo a las baldosas cerámicas con una alta porosidad, normalmente prensadas en seco y que están fabricadas por monococción o por biccoción. El uso más común, debido en parte a sus características técnicas, es para revestimiento de interiores. Junto con los pavimentos de gres, el azulejo representa la mayor parte de la producción española con una oferta muy amplia, dadas su amplitud de medidas y/o acabados.

Las materias primas utilizadas en la fabricación del azulejo son muy variadas. Como soporte se puede utilizar mayólica o loza fina, de color blanco, grisáceo, crema, marfil, pardo o rojizo, sin que este color afecte al producto final. El esmalte de la cara vista puede ser blanco, de un solo color, marmoleado, moteado o puede llevar diversos decorados, motivos y acabados. Las formas predominantes son las cuadradas y rectangulares.



Figura D.1 – Ejemplo de azulejo para revestimiento.

D.2. Pavimento de gres

El pavimento de gres es un tipo de baldosa cerámica donde la absorción de agua es intermedia o baja, el prensado se realiza en seco y la cocción normalmente se hace por el método de monococción. El pavimento de gres también se conoce como pavimento gresificado, pavimento cerámico esmaltado o simplemente pavimento cerámico. El uso habitual es el de pavimento de interior, sea en residencias o en locales comerciales, dado que reúne las características de resistencia a la abrasión y a la helada, de ahí que se utilice también para revestir fachadas y para pavimentar suelos expuestos a la intemperie. Como se ha comentado, junto con el azulejo, representa la mayor parte de la producción española, existiendo también una oferta muy amplia.

El pavimento de gres, como su nombre indica, está fabricado a partir de gres (baja absorción de agua) o gresificado (absorción de agua media-baja), el color suele ir desde el blanco hasta el pardo, el grano es fino y poco apreciable a simple vista.

Al igual que con el azulejo, existe mucha variedad tanto a la hora del esmaltado como en las formas.



Figura D.2 – Ejemplo de pavimento de gres.

D.3. Gres porcelánico

Las baldosas cerámicas con una muy baja absorción de agua, prensadas en seco y no esmaltadas se conocen como gres porcelánico, al no ser esmaltadas tan sólo pasan por un único proceso de cocción. Las baldosas de gres pueden pasar por un proceso de pulido tras la cocción, para darle brillo. Es posible que el gres porcelánico tenga relieves con fines decorativos o antideslizantes. Se trata de un producto relativamente reciente, de gran aceptación y con una demanda creciente.



Figura D.3 – Ejemplo de pavimento de gres porcelánico.

D.4. Baldosín catalán

El baldosín catalán es un tipo de baldosa cerámica tradicional que se moldea por extrusión y que generalmente no está esmaltada (sometida a una única cocción) y que tiene como principal característica su elevada absorción de agua. También es conocido como semi-gres no esmaltado.

El color del baldosín no esmaltado es el propio rojo de la arcilla cocida, aunque puede estar esmaltado con cubiertas vidriadas y adoptar colores variados (baldosas vidriadas). El uso es tradicional en la costa mediterránea para el solado de terrazas, balcones, porches y cocinas de estilo rural. El consumo y la producción en España tienden a decrecer.

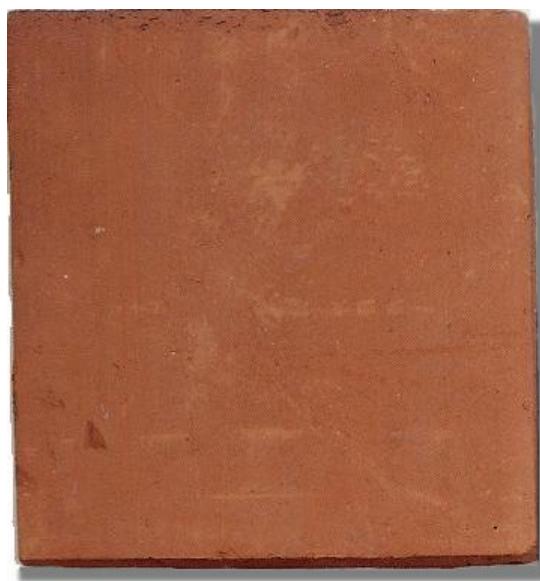


Figura D.4 – Ejemplo de baldosín catalán.

D.5. Gres rústico

El gres rústico comprende las baldosas cerámicas con baja absorción de agua, moldeadas por extrusión y generalmente no esmaltadas. El grosor de las piezas es muy variable según las medidas y tipos, pero usualmente son mayores a los del pavimento de gres. El uso es muy variado, desde revestimiento de fachadas a suelos industriales.

La producción de gres rústico en España es relativamente pequeña, aunque existen una gran variedad de tipos que comentamos brevemente:

- Baldosas “quarry tiles”, baldosas extrudidas tradicionales, en algunos casos son esmaltadas.
- Baldosas “Spalplatten”, que son baldosas dobles separables, moldeadas por extrusión y separadas mediante un golpe seco después de la cocción, tienen unas ranuras características en la parte posterior.

- Baldosas “de gres salado”, sobre las que se añade sal común durante el proceso de cocción, de gran efecto decorativo pero poca resistencia a la abrasión.

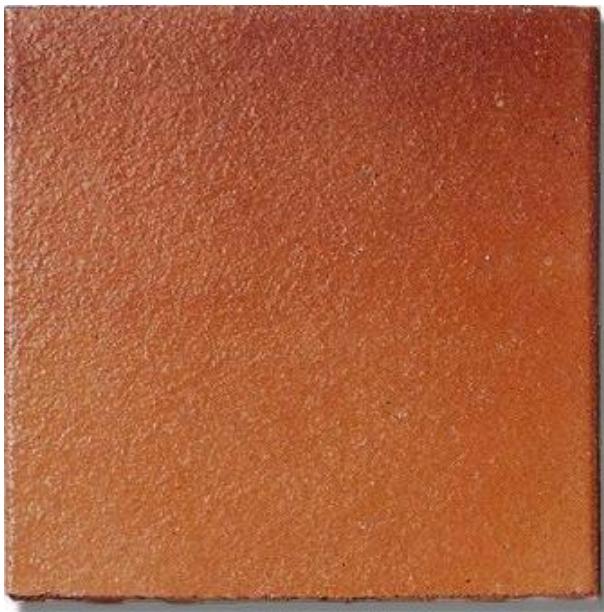


Figura D.5 – Ejemplo de gres rústico.

D.6. Barro cocido

Bajo la denominación de barro cocido se encuentran una gran variedad de productos con características diferentes, fabricados en pequeños lotes y con medios artesanales. Es importante no confundir el barro cocido con el pavimento y baldosa de gres de producción industrial y acabado rústico. Existe una gran dispersión de formas y medidas, aunque todas coinciden en su aspecto rústico, muchas veces intencionado.

Debido a sus características, el uso del barro cocido queda casi exclusivamente acotado a edificaciones o estancias rústicas, donde a veces es necesario aplicar tratamientos de impermeabilización posteriores a la colocación de las piezas.



Figura D.6 – Ejemplo de barro cocido (aplicación).