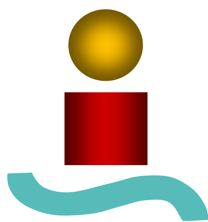


Grupo Organización Industrial

Secuenciación de trabajos en flujo uniforme mediante algoritmos de computación paralela: Diseño, implementación y ajuste

Tesis Doctoral

Autor:

José Miguel León Blanco

Directores:

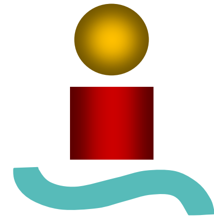
José Manuel Framiñán Torres

Pedro Luis González Rodríguez

Sevilla, octubre de 2009

Universidad de Sevilla

Escuela Superior de Ingenieros



Departamento de Organización Industrial y Gestión de Empresas

Secuenciación de trabajos en flujo uniforme mediante algoritmos de computación paralela: Diseño, implementación y ajuste

Tesis Doctoral

José Miguel León Blanco

Directores:

Dr. José Manuel Framiñán Torres

Dr. Pedro Luis González Rodríguez

Sevilla, 2009

Agradecimientos

Me gustaría agradecer el apoyo recibido de compañeros y profesores del grupo de Investigación Organización Industrial, de la Escuela Superior de Ingenieros de la Universidad de Sevilla, especialmente de los profesores José Manuel Framiñán y Pedro Luis González por sus sugerencias sobre el trabajo, su dirección y sus constantes palabras de ánimo.

Al personal del centro de cálculo, especialmente a Corsino Álvarez, por su constante disponibilidad ante las dificultades que han ido surgiendo a lo largo de este trabajo.

A María José por los fines de semana sacrificados y por su constante apoyo y comprensión y a Juan por las horas de juego perdidas.

Gracias a todos.

José Miguel León Blanco

Índice

1 INTRODUCCIÓN Y OBJETO DE LA TESIS 1

1.1 Introducción 1

1.2 Objeto del trabajo 4

1.3 Organización del documento 5

2 PROGRAMACIÓN EN PARALELO. ESTADO DEL ARTE 7

2.1 Introducción 7

2.2 Definiciones básicas 7

2.2.1 Hilos, procesos, tareas y sincronización 7

2.2.2 Computación secuencial, paralela y distribuida 9

2.2.3 Escalabilidad 13

2.3 Ventajas de la computación paralela 13

2.3.1 Reducción del tiempo de ejecución 13

2.3.2 Otras medidas de la reducción en el tiempo de ejecución 17

2.3.3 Mejoras en la robustez 19

2.4 Inconvenientes de la computación paralela 20

2.5 Plataformas paralelas de computación 22

2.5.1 Arquitecturas hardware 23

2.5.2 Software de base 32

2.5.3 Herramientas para evaluación de las prestaciones de algoritmos paralelos 47

2.6 Implementación de algoritmos paralelos. Mecanismos de paralelización de algoritmos 49

2.6.1 Estrategia de exploración del espacio de soluciones 50

2.6.2 División del trabajo 50

2.6.3 Comunicación entre procesos 53

2.6.4 Granularidad 54

2.6.5 Revisión de tipos de implementación en paralelo 54

2.6.6 Procedimientos para el desarrollo de aplicaciones paralelas 56

2.7 Referencias sobre programación en paralelo 58

2.8 Conclusiones 61

3 COMPUTACIÓN PARALELA APLICADA A PROBLEMAS DE SECUENCIACIÓN EN FLUJO UNIFORME 63

3.1 Introducción 63

3.2 La secuenciación de trabajos en entornos de flujo uniforme 64

3.3 Métodos de resolución del problema objeto de estudio mediante computación paralela. Estado del arte 66

3.3.1 Métodos Exactos: Branch&Bound 68

3.3.2 Métodos aproximados 76

3.3.3 Métodos basados en poblaciones 93

3.3.4 Scatter Search y Path Relinking 97

3.3.5 Optimización mediante colonias de hormigas (ACO) 99

3.4 Resumen de prestaciones de algoritmos paralelos en cuanto a eficiencia y aceleración 101

3.5 Resumen de aplicaciones en paralelo al problema $F_m|prmu|C_{max}$ 109

3.6 Conclusiones 113

4 CLM-PARALELO (CLMP) 117

4.1 Introducción 117

4.2 Descripción de la metaheurística CLM 118

4.3 Algoritmo paralelo de grano grueso: CLMpg 124

4.3.1 Proceso Jefe 125

4.3.2 Proceso trabajador 132

4.3.3 Experimentación 138

4.4 Algoritmo paralelo de grano fino CLMpf 146

4.4.1 Proceso jefe 147

4.4.2 Proceso Trabajador 148

4.4.3 Diseño experimental 151

4.4.4 Resultados: ARPD 159

4.4.5 Resultados: Tiempo de ejecución 167

4.5 Conclusiones 175

5 AJUSTE DE PARÁMETROS EN ALGORITMOS PARALELOS 177

5.1 Introducción 177

5.2 Estado de la cuestión 178

5.2.1 Metodologías OFAT 180

5.2.2 Metodologías basadas en la optimización global 181

5.2.3 Metodologías basadas en diseño estadístico de experimentos 182

5.2.4 Metodologías de experimentación para ajuste de parámetros en algoritmos paralelos 184

5.3 Metodología para el ajuste de parámetros en algoritmos paralelos de búsqueda local para maximizar la eficiencia 185

5.3.1 Introducción 185

5.3.2 Descripción de la metodología 185

5.4 Diseño experimental para la mejora de la eficiencia 187

5.4.1 Descripción de los factores 187

5.4.2 Número de niveles 188

5.4.3 Selección del tipo de diseño experimental 189

5.4.4 Datos 190

5.4.5 Respuestas 190

5.4.6 Número de replicados 190

5.4.7 Resultados 190

5.4.8 Análisis de los resultados de eficiencia 193

5.4.9 Optimización 195

5.5 Estudio estadístico sobre la eficiencia del algoritmo CLMpf ... 200

5.5.1 Tablas de resultados medios de eficiencia ... 201

5.5.2 Gráficas de las medias de la eficiencia ... 204

5.5.3 Análisis estadístico ... 207

5.5.4 Influencia del número de máquinas ... 210

5.5.5 Selección del mejor conjunto de parámetros ... 211

5.6 Estudio de la robustez del algoritmo ... 212

5.6.1 Introducción ... 212

5.6.2 Enfoque 1: Robustez para un número determinado de procesos y diferentes tamaños de problema 214

5.6.3 Enfoque 2: Robustez para un problema determinado y diferente número de procesadores ... 217

5.7 Conclusiones ... 220

6 ALGORITMO PARALELO HÍBRIDO PBDS ... 223

6.1 Introducción ... 223

6.2 Estado del arte ... 224

6.2.1 Combinaciones de métodos aproximados ... 225

6.2.2 Combinaciones de métodos exactos y aproximados ... 234

6.2.3 Conclusiones sobre algoritmos híbridos que incluyen métodos exactos y aproximados ... 241

6.3 Descripción del algoritmo secuencial BDS ... 242

6.3.1 Estructura general ... 243

6.3.2 Método exacto ... 244

6.3.3 Método aproximado ... 244

6.4 Descripción del algoritmo paralelo híbrido pBDS ... 246

6.4.1 Comienzo de la exploración ... 246

6.4.2 Intercambios de información entre procesos ... 249

6.5 Experimentación ... 251

6.5.1 Parámetros de exploración ... 251

6.5.2 Parámetros del algoritmo paralelo ... 253

6.5.3 Resultados experimentales ... 253

6.6 Conclusiones ... 258

7 CONCLUSIONES Y LÍNEAS FUTURAS DE INVESTIGACIÓN ... 261

7.1 Contribuciones ... 263

7.1.1 Aportaciones de la tesis ... 263

7.1.2 Trabajos derivados de esta tesis ... 264

7.2 Futuras líneas de trabajo ... 265

8 REFERENCIAS ... 269

9 ANEXOS ... 301

9.1 Anexo 1: Valores de makespan ... 301

9.2 Anexo 2: Patologías en las observaciones ... 302

9.3 Anexo 3: Tablas para la eficiencia y relación señal/ruido en el algoritmo CLMpf ... 303

9.3.1 Estudio según el número de procesadores ... 303

9.3.2 Estudio según el tamaño de problema ... 304

9.4 Anexo 4: Gráficas de medias marginales para la relación señal/ruido para distintos tamaños de problema para CLMpf ... 306

9.4.1 20x20 ... 306

9.4.2 50x5 ... 306

9.4.3 50x10 ... 307

9.4.4 50x20 ... 307

9.4.5 100x5 ... 308

9.4.6 100x10 ... 308

9.4.7 100x20 ... 309

9.4.8 200x10 ... 309

9.4.9 200x20 ... 310

9.5 Anexo 5: Contrastes y tratamientos empleados en la experimentación con CLMpf ... 311

9.6 Anexo 6: Comparación de los resultados de eficiencia de los algoritmos CLMpg y CLMpf 312

9.7 Anexo 7: Resultados de la experimentación con el algoritmo pBDS..313

9.7.1 Serie 20x5..313

9.7.2 Serie 20x10..314

9.7.3 Serie 20x20..315

9.7.4 Serie 50x5..316

9.7.5 Serie 50x10..317

9.7.6 Serie 50x20..318

Índice de tablas

Tabla 1: Combinaciones de programación paralela y distribuida con tipos de ordenador (Hughes y Hughes, 2003) 12

Tabla 2: Herramientas para evaluación del rendimiento de aplicaciones en paralelo 48

Tabla 3: Tipologías de algoritmos paralelos 55

Tabla 4: Referencia sobre conceptos básicos en algoritmos paralelos 58

Tabla 5: Resumen de referencias sobre librerías de paso de mensajes entre procesos 58

Tabla 6: Resumen de referencias sobre entornos de desarrollo 59

Tabla 7: Resumen de referencias sobre procedimientos de paralelización 59

Tabla 8: Resumen de referencias sobre división del trabajo entre procesos 60

Tabla 9: Resumen de referencias sobre evaluación de las prestaciones de algoritmos paralelos 60

Tabla 10: Algoritmos tipo B&B en paralelo aplicados a problemas de permutación (ILP – *Integer Linear Programming*, 0-1 KP – *Knapsack Problem*) 75

Tabla 11: Resultados (ARPD) del algoritmo SA paralelo de Wodecki y Bozejko (2002) y la heurística NEH frente a la batería de Taillard (1993) 81

Tabla 12: Referencias en Vallada (2006) 95

Tabla 13: Eficiencia del algoritmo paralelo propuesto por Aiex *et al.* (2002) 102

Tabla 14: Aceleración y eficiencia del algoritmo paralelo propuesto por Taillard (1994) 102

Tabla 15: Aceleración y eficiencia obtenidas por el algoritmo paralelo propuesto por Malek *et al.* (1989) 103

Tabla 16: Resumen de las prestaciones de otros algoritmos paralelos 105

Tabla 17: Aceleración conseguida por el algoritmo paralelo propuesto por Brüngger *et al.* (1999) 106

Tabla 18: Eficiencia de algunos de los algoritmos estudiados 108

Tabla 19: Referencias sobre aplicaciones en paralelo para el problema $F_m/prmu/C_{max}$ 110

Tabla 20: Abreviaturas empleadas en la Tabla 19 111

Tabla 21: Resumen de similitudes de CLM con otras metaheurísticas en función de las decisiones sobre los parámetros 123

Tabla 22: Parámetros del algoritmo paralelo 140

Tabla 23: ARPD promediados según la dimensión del problema con un solo procesador 141

Tabla 24: Tiempo promediado según la dimensión del problema y el número de procesos 143

Tabla 25: Eficiencia del algoritmo paralelo según tamaño de problema y número de procesos .. 144

Tabla 26: Eficiencia incremental generalizada del algoritmo paralelo para problemas con 100 trabajos y 5 o 10 máquinas según la variación en el número de procesos 145

Tabla 27: Eficiencia incremental generalizada del algoritmo paralelo para problemas con 100 trabajos y 20 máquinas o 200 trabajos y 10 máquinas según la variación en el número de procesos 145

Tabla 28: Eficiencia incremental generalizada del algoritmo paralelo para problemas con 200 trabajos y 20 máquinas o 500 trabajos y 20 máquinas según la variación en el número de procesos 145

Tabla 29: Parámetros de la búsqueda local 155

Tabla 30: Matriz de Diseño experimental L18 de Taguchi 158

Tabla 31: Resultados de los experimentos 159

Tabla 32: ANOVA de una vía para promedios de la respuesta ARPD, nivel significación: 0.1 160

Tabla 33: ANOVA de un grado de libertad para ARPD 160

Tabla 34: Resumen de ANOVA para ARPD 165

Tabla 35: ANOVA de una vía para el tiempo con nivel alpha = 0.1 168

Tabla 36: ANOVA de un grado de libertad para el tiempo de ejecución del algoritmo 168

Tabla 37: Resumen del análisis de varianza para el tiempo. 172

Tabla 38: Niveles óptimos de los parámetros 173

Tabla 39: Valores de compromiso ARPD/tiempo para iteraciones, número de procesos y K 173

Tabla 40: Efectos principales para iteraciones, número de procesos y K en cuanto a ARPD y tiempo 174

Tabla 41: Parámetros de la búsqueda local 187

Tabla 42: Valores de los parámetros para tiempo de ejecución y calidad de soluciones 188

Tabla 43: Parámetros de la búsqueda local 189

Tabla 44: Conjunto de tratamientos para eficiencia 189

Tabla 45: Calidad (ARPD) de las soluciones obtenidas en un procesador con 10 iteraciones globales 191

Tabla 46: Tiempo de ejecución del algoritmo en un solo procesador con 10 iteraciones globales 191

Tabla 47: Resultados de los experimentos 193

Tabla 48: ANOVA de 1 vía para promedios de la eficiencia, nivel Alfa: 0.1 193

Tabla 49: ANOVA de un grado de libertad para eficiencia 194

Tabla 50: Resumen de ANOVA para ARPD 198

Tabla 51: Factores y niveles 201

Tabla 52: Media de la eficiencia según trabajos (n), máquinas (m) y procesadores (np) 202

Tabla 53: Resultados medios de eficiencia agrupados según n y m 203

Tabla 54: Promedios de eficiencia agrupados según cada uno de los tratamientos 204

Tabla 55: Subgrupos según el factor número de procesadores 209

Tabla 56: Subgrupos según el factor número de trabajos 210

Tabla 57: Subgrupos según el factor número de máquinas 210

Tabla 58: Subconjuntos homogéneos según el factor combinación de parámetros 211

Tabla 59: Conjunto de valores que ofrece mejor resultado en cuanto a eficiencia 212

Tabla 60: Matriz experimental de Taguchi 214

Tabla 61: Relación señal/ruido para cada número de procesadores, siendo el tamaño de problema el factor de ruido 214

Tabla 62: Valores de los parámetros que maximizan la relación señal/ruido para cada número de procesadores 216

Tabla 63: Relación señal/ruido según tamaño de problema para 12 procesadores 216

Tabla 64: Relación señal/ruido para cada tamaño de problema, siendo el número de procesadores el factor de ruido 217

Tabla 65: Valores de los parámetros que maximizan la relación señal/ruido para cada tamaño de problema 218

Tabla 66: Combinación de parámetros para maximizar la relación señal/ruido para la serie 100x20 219

Tabla 67: Relación señal/ruido según número de procesadores para la serie 100x20 219

Tabla 68: Algoritmos secuenciales que combinan métodos aproximados aplicados a problemas de permutación (GA: *Genetic Algorithms*, LS: *Local Search*, SA *Simulated Annealing*, ACO: *Ant Colony Optimization, PSO: Particle Swarm Optimization*) 228

Tabla 69: Algoritmos paralelos que combinan métodos aproximados aplicados a problemas de permutación (TS: *Tabu Search*, GA/EA: *Genetic Algorithms/Evolutionary Algorithms*) 229

Tabla 70: Tiempos promedio, aceleración y eficiencia conseguidas por el algoritmo paralelo propuesto por Matsumura *et al.* (2000) 231

Tabla 71: Algoritmos híbridos secuenciales que combinan métodos exactos y aproximados 234

Tabla 72: Algoritmos paralelos que combinan métodos exactos y aproximados aplicados a problemas de permutación 237

Tabla 73: Resultados en tiempo de despliegue y de ejecución de Bendjoudi *et al.* (2008) y Bendjoudi *et al.* (2007), entre paréntesis 240

Tabla 74: Eficiencia del algoritmo pBDS 255

Tabla 75: Eficiencia incremental generalizada del algoritmo pBDS 255

Tabla 76: Comparación de la eficiencia de los algoritmos CLMpg, CLMpf y pBDS 256

Tabla 77: Batería de Taillard (1993). Valores mínimos de makespan empleados en este documento 301

Tabla 78: Composición de los contrastes 311

Tabla 79: Conjunto de tratamientos 311

Tabla 80: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 20x5 313

Tabla 81: Eficiencia incremental generalizada para la serie 20x5 313

Tabla 82: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 20x10 314

Tabla 83: Eficiencia incremental generalizada para la serie 20x10 314

Tabla 84: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 20x20 315

Tabla 85: Eficiencia incremental generalizada para la serie 20x20 315

Tabla 86: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 20x20 316

Tabla 87: Eficiencia incremental generalizada para la serie 50x5 316

Tabla 88: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 50x10 317

Tabla 89: Eficiencia incremental generalizada para la serie 50x10 317

Tabla 90: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 50x20 318

Tabla 91: Eficiencia incremental generalizada para la serie 50x20 318

Índice de figuras

Figura 1: Ejemplo de la división de un programa en procesos y en hilos 8

Figura 2: Ejemplo de programación concurrente 10

Figura 3: Eficiencia máxima de un algoritmo paralelo según la ley de Amdahl 22

Figura 4: Estructura general de un sistema informático para computación paralela 23

Figura 5: Jerarquías de memoria (adaptado de Dongarra *et al.,* 2003) 24

Figura 6: Sistemas multiprocesador de memoria compartida según Barr y Hickman (1993) 25

Figura 7: Sistema multiprocesador con sistema operativo multiproceso según Brawer (1986) 26

Figura 8: Esquema de PRAM (Gibbons y Ritter, 1988) 28

Figura 9: Esquema del clúster empleado en esta investigación 30

Figura 10: Esquema de clúster según Buyya (1999) 34

Figura 11: Tipos de hilos de ejecución en paralelo 34

Figura 12: Modelo de computación y modelo de la arquitectura de PVM (Sunderam *et al.*, 1994) 38

Figura 13: Clasificación de algoritmos de búsqueda en paralelo según Crainic *et al.* (1997) 49

Figura 14: Ejemplo de las diferentes alternativas de Flynn (1966) según Kindervater y Lenstra (1989) 51

Figura 15: Grafo dirigido para el cálculo de makespan 65

Figura 16: Métodos de resolución en paralelo aplicados al problema $F_m/prmu/C_{max}$ 67

Figura 17: Tipos de implementación de metaheurísticas según Talbi (2002). 68

Figura 18: Clasificación de tipos de implementación en paralelo de metaheurísticas según Talbi. ..68

Figura 19: Pseudocódigo general de Branch and Bound (Adaptado de Gillet, 1976) 70

Figura 20: Pseudocódigo de algoritmo secuencial de recocido simulado 78

Figura 21: Versión básica de búsqueda tabú (Glover, 1989) 83

Figura 22: Algoritmo paralelo adaptativo de búsqueda tabú (Talbi *et al.*, 1998) 84

Figura 23: Pseudocódigo de VNS (adaptado de García López *et al.,* 2002) 91

Figura 24: Pseudocódigo de *scatter search* (adaptado de Adenso-Díaz *et al.*, 2006)........................98

Figura 25: Pseudocódigo (Dorigo y Di Caro, 1999) de optimización mediante colonias de hormigas 100

Figura 26: Aceleración, eficiencia y calidad de las soluciones en el algoritmo de Sanvicente y Frausto, (2002).. 106

Figura 27: Eficiencia de algunos de los algoritmos estudiados 108

Figura 28: Pseudocódigo de CLM (Ghosh y Sierksma, 2002) 119

Figura 29: Esquema de funcionamiento interno del algoritmo CLM...................................... 120

Figura 30: Pseudocódigo del proceso jefe 126

Figura 31: Estructura de las soluciones almacenadas en la memoria 128

Figura 32: Esquema de inclusión de soluciones en el *pool* central y modificación de la movilidad 129

Figura 33: Esquema de funcionamiento en paralelo.. 132

Figura 34: Pseudocódigo del proceso trabajador 134

Figura 35: Diagrama de secuencia del algoritmo paralelo de grano grueso 135

Figura 36: Pseudocódigo de la función Actualizar umbral 136

Figura 37: Desplazamiento hacia adelante.. 138

Figura 38: Intercambio aleatorio.. 138

Figura 39: Pseudocódigo del proceso jefe o master 147

Figura 40: Pseudocódigo de la función Generar nueva solución y política del proceso jefe 147

Figura 41: Pseudocódigo del proceso trabajador o esclavo 149

Figura 42: Diagrama de secuencia de la comunicación entre procesos 150

Figura 43: Gráfico de efectos principales para α 162

Figura 44: Gráfico de efectos principales para β 162

Figura 45: Gráfico de efectos principales para *pool*.. 163

Figura 46: Gráfico de efectos principales para las iteraciones en los trabajadores.............. 163

Figura 47: Gráfico de efectos principales para las iteraciones en el proceso jefe 164

Figura 48: Gráfico de efectos principales para el número de procesos 164

Figura 49: Gráfico de efectos principales para K 164

Figura 50: Gráfico de efectos principales para B 165

Figura 51: Gráfico de efectos principales para α 169

Figura 52: Gráfico de efectos principales para β 169

Figura 53: Gráfico de efectos principales para iteraciones en trabajadores 170

Figura 54: Gráfico de efectos principales para iteraciones en proceso jefe 170

Figura 55: Gráfico de efectos principales para K 170

Figura 56: Gráfico de efectos principales para número total de procesos 171

Figura 57: Gráfico de efectos principales para B 171

Figura 58: Gráfico de efectos principales para el tamaño de la memoria central 171

Figura 59: Gráfico de calidad y tiempo de ejecución del algoritmo secuencial 192

Figura 60: Gráfico de efectos principales para K 195

Figura 61: Gráfico de efectos principales sobre la eficiencia para α 196

Figura 62: Gráfico de efectos principales sobre la eficiencia para *pool* 196

Figura 63: Gráfico de efectos principales sobre la eficiencia para β 197

Figura 64: Gráfico de efectos principales sobre la eficiencia para el número de iteraciones en los trabajadores (it_s) 197

Figura 65: Gráfico de efectos principales para B 198

Figura 66: Diagrama de cajas para la eficiencia según el número de procesadores (nivel de confianza del 95%) 205

Figura 67: Valor medio de eficiencia según el número de procesadores 205

Figura 68: Valores medios de eficiencia agrupados por número de trabajos para cada número de procesadores 206

Figura 69: Valores medios de eficiencia agrupados por número de máquinas para cada número de procesadores 207

Figura 70: 3 procesadores 215

Figura 71: 6 procesadores 215

Figura 72: 9 procesadores 215

Figura 73: 12 procesadores 215

Figura 74: Relación señal/ruido para la eficiencia del algoritmo cuando se varía el número de procesadores para problemas 20x20 218

Figura 75: Clasificación propuesta de algoritmos híbridos 225

Figura 76: Clasificación de heurísticas paralelas de Raidl (2006) 226

Figura 77: Combinaciones de metaheurísticas de Raidl (2006) 227

Figura 78: Diagrama de secuencia del intercambio de mensajes en HPMH de Matsumura *et al.* (2000) 230

Figura 79: Pseudocódigo del algoritmo exacto de Framiñán y Pastor (2008) 245

Figura 80: Diagrama de secuencia del algoritmo pBDS propuesto en este capítulo de la tesis 247

Figura 81: Comparación de la eficiencia de los algoritmos CLMpg y pBDS (20x5 y 20x10) 256

Figura 82: Comparación de la eficiencia de los algoritmos CLMpg, CLMpf y pBDS (20x20) 257

Figura 83: Comparación de la eficiencia de los algoritmos CLMpg, CLMpf y pBDS (50x5 y 50x10) 257

Figura 84: Comparación de la eficiencia de los algoritmos CLMpg, CLMpf y pBDS (50x20) 258

1 Introducción y objeto de la tesis

1.1 Introducción

En numerosas plantas de producción y montaje se emplean configuraciones o disposiciones de máquinas, de forma que los trabajos a procesar realizan una serie de operaciones en el mismo orden. Este tipo de entornos de producción se conocen como entornos de flujo uniforme, talleres de flujo o entornos tipo *flowshop*. La secuenciación o programación de los trabajos en estos entornos tiene una enorme importancia, ya que existen notables diferencias en lo que respecta al uso eficiente de los recursos (Pinedo, 2002). Para valorar la eficiencia de determinada secuencia o programa de producción, se suelen emplear diferentes medidas. Una de las más conocidas es el *makespan* o tiempo de terminación del último trabajo (Conway *et al.*, 1967). Su importancia radica en que equivale a conseguir la máxima utilización de las máquinas del taller de flujo objeto de estudio. El problema de secuenciación de trabajos en entornos de flujo uniforme con el objetivo de minimizar el *makespan* es un problema NP-completo para un número de máquinas mayor de dos (Garey *et al.*, 1976), por lo que la mayor parte de las contribuciones en este campo se refieren a la implementación de métodos aproximados o heurísticas que permiten obtener soluciones cercanas al óptimo en tiempos de computación razonables. En concreto, la aplicación a este problema de diversas heurísticas de propósito general o metaheurísticas, como la búsqueda tabú o los algoritmos genéticos entre otras, ha proporcionado muy buenos resultados en cuanto a calidad de las soluciones en tiempos de computación reducidos. En Ruiz y Maroto (2005) o en Framiñán *et al.* (2004) se muestra un resumen de las principales heurísticas aplicadas a este problema.

Una vía de investigación actual que se abre dentro del empleo de metaheurísticas para el problema de secuenciación antes mencionado es la paralelización de estos algoritmos. Ésta consiste en aprovechar las capacidades de computación de varios ordenadores conectados entre sí a través de un medio físico. El desarrollo de un algoritmo de este tipo implica (si se produce cooperación entre los procesos que lo componen) mayor eficacia en su ejecución debido al conocimiento que adquiere un proceso sobre la búsqueda de soluciones realizada por el resto de procesos. Este conocimiento incluye la calidad de las soluciones encontradas por otros procesos, las características del espacio de soluciones, el estancamiento en mínimos locales o la política de exploración que están empleando (como por ejemplo, los parámetros de la búsqueda, criterios de selección de rutas de exploración o criterios de fin de la exploración).

Todo este intercambio de información conlleva una mejora en la calidad de las soluciones que se obtienen en comparación con la calidad que se obtendría en ausencia de comunicación. Este intercambio puede llevar, en algunos casos a que la mejora del tiempo de computación del algoritmo paralelo sobre el secuencial sea mayor que proporcional al número de procesadores empleados, lo que se conoce como aceleración superlineal. Sin embargo, hay que tener en cuenta que la aceleración de un algoritmo paralelo no es ilimitada (Foster y Riseman, 1972). Además, es posible conseguir mejoras en la robustez del algoritmo, entendiendo por tal su capacidad para resolver diferentes instancias del mismo problema, ya que la resolución en paralelo permite una mejor exploración del espacio de soluciones para un mismo tiempo de ejecución, disminuyendo la sensibilidad del algoritmo a variaciones en las características del problema.

Por otra parte, puesto que las metaheurísticas son procedimientos generales para la resolución aproximada de problemas, su funcionamiento óptimo requiere su ajuste para cada problema particular. Una de las posibles técnicas que se puede aplicar para dicho ajuste es el diseño de experimentos (DOE-*Design of Experiments*) (Montgomery, 2002) o incluso la aplicación de conceptos de robustez en el diseño (ver Taguchi *et al.,* 1989) al ajuste de estos procedimientos.

Por otra parte, aunque la búsqueda de soluciones de calidad en problemas NP-completos suele ser sinónimo de aplicación de metaheurísticas, se ha observado que la combinación de varios procedimientos distintos en la búsqueda de estas soluciones puede ofrecer resultados mejores que cada uno de los procedimientos por separado. Estos procedimientos híbridos pueden combinar tanto diferentes metaheurísticas como metaheurísticas con procedimientos exactos. De este modo, se obtendrían no solo soluciones de mejor calidad sino también algoritmos más robustos que cada uno de los procedimientos por separado (Talbi, 2002 o Basseur *et al.*, 2004).

El desarrollo de algoritmos paralelos se encuentra con dificultades añadidas a las del desarrollo de algoritmos secuenciales ya que éstos deben diseñarse de forma que aprovechen al máximo la potencia de cálculo añadida que supone disponer de varios procesadores trabajando de forma simultánea. Para evaluar el comportamiento de un algoritmo paralelo, se suelen emplear medidores relacionados con el ratio entre el tiempo que emplea una versión secuencial del algoritmo y el tiempo que necesita la versión paralela para alcanzar soluciones de la misma calidad que las encontradas por la versión secuencial. Uno de estos medidores es la aceleración o *speedup*. Si la aceleración del algoritmo es lineal, significa que el algoritmo paralelo, empleando n procesadores, tarda t/n en encontrar una solución de la misma calidad que la que encontró el algoritmo secuencial en un tiempo t. Otra de las medidas del aprovechamiento de la potencia de cálculo en algoritmos paralelos es la eficiencia. La eficiencia del algoritmo paralelo es el cociente entre la aceleración y el número de procesadores empleado. Un valor de eficiencia del 100% significa un aprovechamiento completo de todos los procesadores empleados para la resolución, lo que se denomina aceleración lineal.

Esta tesis está dedicada a abordar el diseño y la implementación de algoritmos paralelos para el problema de secuenciación de trabajos en entornos de flujo uniforme con el objetivo de minimizar el *makespan*. Asimismo, se abordará el problema del ajuste de parámetros de forma que se consiga un algoritmo paralelo con un funcionamiento óptimo, no solo en cuanto a calidad de soluciones y tiempo de ejecución sino en cuanto a robustez y aprovechamiento del sistema. Además, se estu-

diarán las mejoras que aporta la combinación de dos métodos diferentes, uno exacto y otro aproximado, para el problema objeto de estudio.

Como resultado de este trabajo, se proponen varios algoritmos paralelos para la aproximación de soluciones del problema de secuenciación mencionado, así como una metodología para el ajuste de parámetros de algoritmos paralelos con el objetivo de encontrar una combinación de parámetros adecuada en un tiempo razonable. También se presenta un algoritmo híbrido que combina los algoritmos paralelos desarrollados en este trabajo con un método de ramificación y acotación (*branch & bound*).

1.2 Objeto del trabajo

El objetivo general de este trabajo es el diseño, evaluación e implementación de algoritmos paralelos eficientes para el problema de minimización del tiempo máximo de terminación en un entorno de flujo uniforme.

Para implementar estos algoritmos paralelos, es preciso disponer en primer lugar, de un sistema informático apropiado. En este trabajo, se investigará la configuración más adecuada del sistema informático que nos permita obtener conclusiones lo más generales posibles sobre el desarrollo de algoritmos paralelos para el problema objeto de estudio. Así mismo, se implementarán algoritmos flexibles, que nos permitan extraer conclusiones extrapolables a otros algoritmos. Para ello, se partirá del algoritmo C*omplete Local search with Memory* (CLM), de Ghosh y Sierksma (2002), que permite, variando sus parámetros característicos, asimilarlo a un algoritmo *greedy* puro, a un recocido simulado o a distintas variantes de búsqueda local.

Se ha comprobado que la eficiencia de un algoritmo paralelo disminuye conforme aumenta el número de procesadores, debido tanto al tiempo de espera en la comunicación entre procesos como a las operaciones que debe realizar el algoritmo de forma secuencial. Por ello, en nuestro trabajo se prestará especial atención a la influencia del número de procesadores en la eficiencia del algoritmo paralelo. De este modo se pretende analizar cómo se degrada dicha eficiencia conforme se emplea en sistemas con cada vez mayor número de procesadores.

Para alcanzar el objetivo general comentado anteriormente, se plantean los siguientes objetivos específicos:

1. Analizar y evaluar las arquitecturas paralelas disponibles. Para ello, se hará una revisión tanto de la bibliografía sobre computación en paralelo como de sistemas informáticos paralelos comerciales y de libre distribución. Sobre esta revisión, se determinará la elección de una plataforma física y de un sistema para el desarrollo y experimentación de algoritmos paralelos. Este objetivo se aborda en el capítulo 2.

2. Revisar el estado del arte sobre algoritmos paralelos para el problema objeto de estudio así como de problemas similares a fin de extraer y clasificar las principales experiencias en este ámbito. Este objetivo se aborda en el capítulo 3.

3. Diseñar e implementar algoritmos paralelos basados en metaheurísticas para el problema objeto de estudio. Para ello se implementarán varias versiones paralelas del algoritmo CLM y se evaluará la eficiencia de las mismas. Este objetivo se aborda en el capítulo 4.

4. Proponer una metodología para la optimización de los parámetros de funcionamiento del algoritmo anterior. Metodología que será general y aplicable a otras metaheurísticas paralelas y otros problemas de secuenciación. Este objetivo se aborda en el capítulo 5.

5. Diseñar e implementar un algoritmo paralelo híbrido que combine un método exacto y un método aproximado. Se evaluará la eficiencia de este algoritmo con objeto de comprobar si sus prestaciones son superiores o no al empleo únicamente de algoritmos aproximados a la hora de abordar problemas NP-completos. Este objetivo se trata en el capítulo 6.

1.3 Organización del documento

En el capítulo 2 de este trabajo se describen las posibilidades existentes en cuanto a la implementación mediante ordenador de algoritmos en paralelo, clasificándolos según el número de pasadas del algoritmo, el empleo de una o varias funciones en cada uno de los procesos, la sincronización entre los mismos o el balance entre co-

municación y computación, entre otros aspectos. También se describen las plataformas existentes para la implementación de algoritmos paralelos. De entre estas posibilidades, se seleccionará la plataforma física y el sistema más adecuado para el desarrollo y la experimentación de algoritmos paralelos.

En el capítulo 3 se describe el problema de secuenciación en flujo uniforme y posteriormente se realiza una revisión de las implementaciones en paralelo de algoritmos aplicados a la resolución del problema objeto de estudio así como a aquellos que guardan cierta similitud.

En el capítulo 4 se realizan dos implementaciones en paralelo de la metaheurística CLM. Se estudia además la influencia de los parámetros del algoritmo paralelo sobre el tiempo de ejecución y la calidad de las soluciones obtenidas.

En el capítulo 5 se presenta una metodología para el ajuste de los parámetros del algoritmo paralelo propuesto en el capítulo anterior, una de las tareas más complejas en la implementación de este tipo de técnicas. Para ello, se presenta una metodología de ajuste de parámetros inspirada en el diseño experimental clásico basado en métodos Taguchi.

En el capítulo 6 se presenta un algoritmo paralelo híbrido que combina un método exacto y un método aproximado. Se realiza una revisión bibliográfica sobre la aplicación de este tipo de algoritmos al problema de secuenciación de trabajos en flujo uniforme con el objetivo de minimizar el *makespan* y problemas similares. El conocimiento adquirido en dicha revisión se emplea para desarrollar un algoritmo paralelo híbrido cuyas prestaciones se comparan con las obtenidas en los algoritmos desarrollados en capítulos anteriores.

Finalmente, en el capítulo 7 se muestran los principales resultados y aportaciones de este trabajo, así como las futuras líneas de investigación que se derivan del mismo.

2 Programación en paralelo. Estado del arte

2.1 Introducción

Este capítulo resume las principales alternativas disponibles para desarrollar aplicaciones que hacen uso de la potencia de cálculo de varios sistemas informáticos trabajando de forma simultánea y cómo se afrontan las dificultades que plantea el desarrollo de estas aplicaciones en tales sistemas.

En primer lugar, se definen una serie de conceptos básicos de la computación paralela como son los relacionados con la computación concurrente en sus dos variantes: paralela y distribuida. Otros conceptos básicos que se discuten son los de proceso, hilo y sincronización. A continuación se revisan las posibilidades que ofrece la utilización de procesadores en paralelo, tanto en forma de *clúster* o granjas de ordenadores, como en forma de recursos de computación de distinta capacidad. Se enumeran las características diferenciales entre la utilización de estos recursos de forma simultánea o de forma asimétrica. El capítulo finaliza con una discusión sobre las alternativas anteriores y la elección de un entorno de trabajo para el desarrollo de algoritmos paralelos en los futuros capítulos.

2.2 Definiciones básicas

2.2.1 Hilos, procesos, tareas y sincronización

Las tareas que precisa realizar un programa o aplicación informática pueden dividirse para su ejecución concurrente mediante el empleo de hilos o mediante procesos.

Se denomina proceso a la unidad de trabajo creada por el sistema operativo, teniendo asignado por el mismo un espacio separado de memoria del ordenador (ver Hughes y Hughes, 2003). Un proceso puede crear otros procesos, que se denominan procesos hijos del anterior.

La diferencia fundamental entre hilos y procesos es que un proceso dispone de un espacio de memoria separado mientras que un hilo no. De esta forma, los hilos pueden trabajar directamente con las variables creadas por el proceso padre mientras que los procesos hijos no pueden hacerlo. Así, suele ser más compleja la programación paralela mediante procesos que mediante hilos. Al mismo tiempo, esta complejidad permite al programador tener mayor control sobre la ejecución del programa. Por último, comentar que el hecho de que los hilos de un proceso compartan su espacio de memoria y recursos puede dar lugar a problemas cuando más de uno de los hilos trata de acceder a las mismas variables y no se diseña correctamente la sincronización entre los hilos.

Figura 1: Ejemplo de la división de un programa en procesos y en hilos

Hilo o *thread* es una sección de código ejecutable de un proceso que puede ser planificada (ver Hughes y Hughes, 2003). Cada proceso tiene un hilo principal, pero

puede tener múltiples hilos (ver Figura 1). Estos son los procesos multihilo o *multi-threaded*. En la Figura 1, el programa se ha dividido en dos procesos. El proceso 1 contiene un hilo principal y un hilo secundario. El proceso 2 únicamente tiene un hilo. A su vez, el proceso 1 ha creado 3 procesos hijos, mientras que el proceso 2 ha creado 2 procesos hijos. Cada uno de estos procesos se ejecuta en un espacio de memoria separado del resto, incluyendo su proceso padre.

Tanto los procesos como los hilos se ejecutan de forma síncrona cuando uno o varios deben esperar alguna señal o mensaje de otro u otros. Cuando cada proceso sigue su ejecución de forma que no es interrumpido por la ejecución de otros, se dice que son procesos o hilos asíncronos.

En bastantes situaciones se requiere que los procesos que componen una aplicación paralela se comuniquen entre sí. Para evitar que un proceso comience a leer los datos antes de que el otro proceso los envíe o para que no se traten de leer más o menos datos que los que se envían, se hace necesaria la sincronización. La segunda utilidad de la sincronización es conseguir la atomicidad de las operaciones, esto es, evitar que procesos reescriban las operaciones realizadas por otros, borrando por tanto operaciones ya realizadas (Quinn, 1987).

2.2.2 Computación secuencial, paralela y distribuida

En oposición a un programa paralelo, se suele emplear el concepto de programa secuencial. En un programa secuencial, el trabajo que debe realizar dicho programa puede dividirse en tareas que debe ejecutar el equipo informático una a continuación de la otra hasta concluirlas. Es el caso de ordenadores con un solo procesador, en el que funciona un único programa cuyas tareas deben ejecutarse una detrás de otra.

Una forma de reducir el tiempo necesario para terminar estas tareas es realizarlas de forma simultánea en un espacio de tiempo. Este es el concepto de programación concurrente (ver Hughes y Hughes, 2003). Estas tareas no tienen por qué realizarse de forma simultánea (en el mismo instante) pero sí concluirse en un espacio de tiempo determinado. Un esquema se muestra en la Figura 2. De este modo, en el tiempo F se han concluido las dos tareas.

Dentro de la programación concurrente, si el programa reparte cada tarea a cada uno de los procesadores de uno o varios ordenadores, estaremos hablando de programación paralela. En este caso, las tareas sí se realizan de forma simultánea en cada instante de tiempo, ya que cada una se ejecuta en un procesador diferente. La computación paralela consiste en la ejecución de programas en un solo ordenador que dispone de varios procesadores (Crichlow, 1988). Este ordenador puede ser un solo equipo físico o virtual formado por varios ordenadores comunicándose entre si.

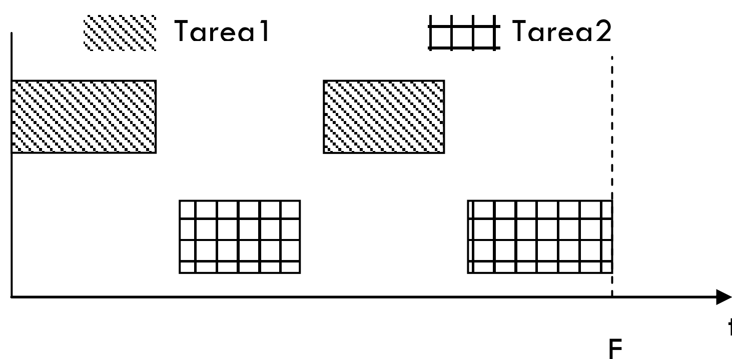


Figura 2: Ejemplo de programación concurrente

También, dentro de la computación concurrente, el programa puede dividirse en un conjunto de programas que se ejecutan cada uno en un ordenador diferente, conectados todos ellos mediante una red de comunicación. En este caso, se habla de computación distribuida. Según Crichlow (1988), la computación distribuida consiste en la ejecución simultánea de programas o procesos en ordenadores separados conectados entre sí por una red de comunicación.

Así, la diferencia fundamental entre programación paralela y programación distribuida está en la forma en la que se reparten las tareas. En la programación paralela, cada tarea se asigna a un procesador, mientras que en la programación distribuida se asigna un proceso a cada tarea. Por este motivo, se puede dar el caso de que una aplicación distribuida pueda funcionar en paralelo cuando cada proceso se ejecuta en un procesador diferente. Así, por ejemplo las librerías de paso de mensajes MPI – *Message Passing Interface* – (Gropp y Lusk, 1992) proporcionan softwa-

re para implementación de aplicaciones distribuidas que pueden funcionar en paralelo. También podemos encontrar aplicaciones paralelas que funcionan como distribuidas cuando cada uno de los procesadores que se emplea en el sistema pertenece a un ordenador diferente. Por otro lado, las librerías de computación paralela PVM (*Parallel Virtual Machine* – Geist *et al.*, 1994) proporcionan software para implementación de aplicaciones paralelas que pueden funcionar en sistemas distribuidos. Ambas cuestiones se tratan en detalle en el apartado 2.5.

Como ejemplos de computación paralela y distribuida se pueden citar los siguientes:

- El superordenador Mare Nostrum (Barcelona Supercomputing Center, 2007), con 10.240 procesadores, capaz de realizar 94.12 Teraflops, empleado en la ejecución de pocas tareas muy complejas en el menor tiempo posible. Es un ejemplo de computación paralela.
- Los ordenadores personales de última generación también son un ejemplo de computación paralela. Estos ordenadores incluyen procesadores de doble, triple y cuádruple núcleo (Intel 2007, Amd 2007), que permitirían la ejecución de dos, tres y cuatro instrucciones de forma simultánea.
- El proyecto SETI@home (SETI, 2007) incluye miles de ordenadores heterogéneos conectados a través de Internet, ejecutándose en cada uno una aplicación sobre una pequeña parte de los datos del problema. Este es un ejemplo de computación distribuida.

En general, con un programa paralelo, se puede sacar el mejor partido posible a los sistemas multiprocesador para resolver ciertos problemas más rápidamente o simplemente para abordar problemas que, de otro modo, serían intratables (ver Kuhn y Padua, 1981). Dentro de estos problemas entran los relacionados con el clima, la investigación biológica y genética, la dinámica de fluidos, el cálculo de estructuras, y más cercanos a la ingeniería de organización, los relacionados con la investigación operativa.

Por otro lado, la programación distribuida hará un mejor uso de sistemas dispersos conectados mediante redes de comunicaciones, como pueden ser las granjas de ordenadores, intranets o incluso Internet. Dentro de estos sistemas entran desde los

sistemas cliente-servidor, los sistemas maestro-esclavo hasta los sistemas entre iguales o *Peer-to-Peer* tal como se comenta en el apartado 2.6.5.

	Un ordenador	**Varios ordenadores**
Programación paralela	Debe constar de varios procesadores. Se consigue dividiendo el programa en varios hilos o procesos. Éstos se pueden ejecutar en diferentes procesadores. Para coordinar las tareas, son necesarias técnicas de comunicación entre procesos.	Se implementa con librerías que consiguen imitar, con varios ordenadores, el comportamiento de un solo equipo con varios procesadores compartiendo la memoria.
Programación distribuida	No se necesitan varios procesadores. Se puede dividir el programa en varios hilos o procesos. Para coordinar las tareas, son necesarias técnicas de comunicación entre procesos.	Se consigue con enlaces entre los equipos y dividiendo los programas en varios fragmentos que puedan ejecutarse de forma independiente. Se puede usar comunicación que se asocia normalmente con la programación paralela.

Tabla 1: Combinaciones de programación paralela y distribuida con tipos de ordenador (Hughes y Hughes, 2003)

Mediante la computación distribuida, se resuelven algunas de las tareas más comunes en las redes de ordenadores, como son los servicios de red, sistemas tolerantes a fallos o redundantes y los sistemas para equilibrado de la carga de trabajo. Puede emplearse tanto un solo ordenador multiprocesador como varios ordenadores, para la programación paralela como para la distribuida (ver Tabla 1). En este sentido, ambos son conceptos que pueden emplearse dentro de una misma aplicación informática, lo cual suele ser la situación más común en la actualidad.

En la experimentación computacional del presente trabajo de investigación, dispondremos de un conjunto de equipos que funcionan de forma simultánea conectados

mediante una red local. Es, por tanto, un sistema distribuido con varios procesadores, uno por cada equipo físico. Sobre este sistema distribuido, desarrollaremos una aplicación y se repartirá el trabajo de computación en diferentes procesos que funcionan en paralelo, uno en cada uno de los procesadores disponibles (ver Capítulo 4).

2.2.3 Escalabilidad

La escalabilidad, en sentido estricto, significa que las prestaciones del sistema paralelo, medidas por la calidad de las soluciones obtenidas o por el tiempo empleado en lograr una determinada calidad de soluciones, se incrementan de forma lineal con el número de procesadores P:

$$Prestaciones(P) = O(P) \quad (1)$$

En un sentido más amplio, también se puede considerar que un sistema paralelo es escalable aunque las prestaciones no aumenten linealmente, si caen menos que un factor logarítmico:

$$Prestaciones(P) = O\left(\frac{P}{(lnP)^c}\right) \quad (2)$$

2.3 *Ventajas de la computación paralela*

2.3.1 Reducción del tiempo de ejecución

Los algoritmos paralelos consiguen dos ventajas fundamentales sobre los mismos algoritmos ejecutados de forma secuencial. Por un lado, la reducción del tiempo de ejecución y por otro, la bondad de las soluciones obtenidas (Alba, 2005). Para comparar la calidad de un algoritmo paralelo frente a otro, se suele recurrir a comparar el tiempo de ejecución para obtener soluciones de la misma calidad. Las dos medidas más empleadas basadas en esta comparación son la aceleración y la eficiencia, que se discuten en los siguientes apartados.

2.3.1.1 Aceleración

Lo que se trata de obtener va más allá de una simple reducción del tiempo de ejecución. Se busca conseguirlo teniendo en cuenta la limitación en la capacidad de

cálculo del sistema informático encargado de la resolución del problema. La premisa suele ser una mejor relación coste/prestaciones de los sistemas compuestos por varios procesadores de bajo coste frente a los sistemas secuenciales más potentes.

Ya se ha mencionado que la aceleración o *speedup* compara el tiempo de ejecución del algoritmo paralelo con el de un algoritmo secuencial. La aceleración $S(p)$ se calcula como el cociente entre el tiempo $t(1)$ necesario para obtener una solución mediante un algoritmo secuencial en un solo procesador y $t(p)$ el necesario para obtener una solución de la misma calidad que la anterior, empleando ahora p procesadores. Suele emplearse la expresión (3) (Ver por ejemplo Alba, 2005):

$$S(p) = \frac{t(1)}{t(p)} \tag{3}$$

Si $S(p) < p$ se denomina aceleración sublineal.

Si $S(p) = p$ se denomina aceleración lineal.

Si $S(p) > p$ se denomina aceleración superlineal.

La aceleración lineal implica que por cada procesador adicional con el que se cuenta, el tiempo de ejecución del algoritmo paralelo disminuye proporcionalmente. Por tanto, en el caso de procesos independientes, se estará aprovechando toda la potencia de cálculo disponible. Para llegar a aceleración superlineal, será necesario que los procesos intercambien información entre sí, de forma que se produzca una disminución mayor sobre el tiempo de ejecución de cada procesador.

Para obtener esta medida, se emplean algoritmos con múltiples procesos independientes, a los que se comunica un valor de la función objetivo alcanzado con un solo proceso resolviendo el problema. Una vez comenzada la ejecución, los procesos se detienen cuando uno de ellos alcanza el valor propuesto para la función objetivo.

También se pueden emplear los conceptos de aceleración relativa y aceleración absoluta. Se habla de aceleración relativa cuando la comparación de los tiempos de ejecución se realiza entre las versiones secuencial y paralela del mismo algorit-

mo. Se habla de aceleración absoluta cuando esta comparación se realiza entre el algoritmo paralelo y la mejor versión secuencial del mismo algoritmo existente en la literatura aplicado al mismo problema.

Como puede verse, la medida de la aceleración no es un asunto trivial. En Roucairol (1996), se recalca la dificultad de comparar los resultados del algoritmo paralelo con los del mejor algoritmo secuencial o bien con uno exactamente igual al algoritmo paralelo con un solo proceso, ya que debido a la comparación con algoritmos diferentes, pueden darse casos en los que la aceleración conseguida con la implementación en paralelo resulte superior a la lineal, mientras que en otros, los resultados sean mucho peores. Esto refleja la falta de homogeneidad en las comparaciones que aparecen en la literatura.

En el caso de emplear procesos independientes, Verhoeven y Aarts (1995) proponen el siguiente teorema:

> Sea $Q_p(t)$ la probabilidad de no haber encontrado una solución de una calidad dada en un tiempo t con p pasadas independientes.
>
> Si $Q_1(t) = e^{-\lambda t}$ con $\lambda \in R^+$, entonces $Q_p(t)=Q_1(pt)$.

Son numerosos los autores que han demostrado la distribución exponencial de la probabilidad de encontrar soluciones de una calidad dada para problemas de optimización combinatoria en general. Por ejemplo, para el problema de asignación cuadrática, Taillard (1991) obtuvo de forma empírica dicha distribución para encontrar soluciones de igual calidad a las mejores conocidas. Más tarde, se ha demostrado la posibilidad de alcanzar una aceleración lineal cuando se resuelven en paralelo problemas de optimización combinatoria mediante búsqueda local y procesos independientes. Verhoeven y Aarts (1995) citan a otros autores que lo han demostrado de forma empírica. La aceleración es aún mayor cuando los procesos interactúan entre sí intercambiando información sobre las soluciones obtenidas.

Por tanto, es posible desarrollar algoritmos paralelos que alcancen una aceleración lineal si la probabilidad de no haber encontrado una solución de una calidad dada en un tiempo t sigue una distribución exponencial. Esto es así empleando procesos independientes, luego (al menos de forma ideal) sería posible conseguir algoritmos

paralelos con aceleración superlineal si los procesos intercambiasen información entre sí.

La posibilidad de aceleración superior a la lineal también ha sido comentada en Quinn (1987). Al disponer de más procesadores, un sistema paralelo tiene una probabilidad mayor de encontrar antes la solución que un solo ordenador con un procesador repitiendo el algoritmo tantas veces como procesadores del sistema paralelo y por tanto sería posible conseguir aceleración superlineal. Esto es particularmente interesante en casos no deterministas.

En un estudio presentado por Pardalos *et al.* (1995a) se ofrecen tres razones para una aceleración mayor que la lineal en el caso de versiones paralelas de algoritmos genéticos. En primer lugar, debido a la selección de los individuos en un subconjunto de la vecindad global, llamada selección local. En segundo lugar, por la evolución de las poblaciones por separado, llamada evolución asíncrona. Y, en tercer lugar, porque las prestaciones de un procesador no influyen en el comportamiento de otro, lo que se denomina fiabilidad en las prestaciones de computación.

En conclusión, es posible diseñar un algoritmo paralelo que consiga una aceleración superlineal y por tanto, reducir el tiempo de ejecución para encontrar soluciones de la misma calidad que el mismo algoritmo ejecutado de forma secuencial de forma más que proporcional al número de procesadores empleado. La comparación con otros algoritmos secuenciales que empleen la misma metaheurística para encontrar soluciones al mismo problema es un tema más complejo.

2.3.1.2 Eficiencia

La eficiencia de un algoritmo paralelo mide el nivel de utilización de cada procesador. Cung *et al.* (2001) definen la eficiencia ρ de un algoritmo paralelo como el ratio entre la aceleración en función del número de procesadores p y este mismo número según la ecuación (4):

$$\rho(p) = \frac{S(p)}{p} \tag{4}$$

Si $\rho(p) < 1$, se tiene el caso de aceleración sublineal y eficiencia menor del 100%.

Si $\rho(p) = 1$, se tiene el caso de aceleración lineal y eficiencia del 100%.

Si $\rho(p) > 1$, se tiene el caso de aceleración superlineal y eficiencia mayor del 100%.

De esta forma, cuando este valor esté próximo a la unidad, significa que la utilización de cada procesador está cerca del 100%, con lo que se estará aprovechando al máximo la disponibilidad de procesadores adicionales en la resolución del problema cuando se emplean procesos independientes.

Se ha comprobado que la computación en paralelo para encontrar soluciones mediante metaheurísticas puede alcanzar, con bastante aproximación, valores de eficiencia cercanos a la unidad en el caso de múltiples procesos independientes. Esto se corresponde con un comportamiento lineal de la aceleración con el número de procesadores.

En definitiva, tanto aceleración como eficiencia se han empleado indistintamente para medir las prestaciones de los algoritmos paralelos. Ambas medidas tienen el mismo significado: el aprovechamiento de la capacidad de cómputo del conjunto de procesadores sin indicar cuál de ellos está más ocupado (Alba, 2005). Conocido el número de procesadores, es inmediato obtener una a partir de la otra. Por ejemplo, con 10 procesadores, una eficiencia del 50% equivale a una aceleración de 5. En nuestro trabajo, emplearemos la eficiencia para medir las prestaciones de nuestro algoritmo, por ofrecer una visión más inmediata del aprovechamiento de los procesadores empleados así como la eficiencia incremental generalizada, que se describe a continuación.

2.3.2 Otras medidas de la reducción en el tiempo de ejecución

En este apartado, se definen una serie de medidas de la reducción del tiempo de ejecución mediante procesadores en paralelo menos habituales que las comentadas en el apartado anterior.

- Eficiencia incremental generalizada

Para obtener una medida más ajustada de cómo se está empleando la potencia de cálculo disponible proporcionada por los procesadores que se van añadiendo, se

puede emplear la eficiencia incremental generalizada (ver Barr y Hickman, 1993). Ésta calcula la mejora en el tiempo de ejecución cuando se añaden nuevos procesadores al clúster. La expresión es la siguiente:

$$gie(p, q) = \frac{p \cdot t(p)}{q \cdot t(q)} \quad (5)$$

Donde $gie(p, q)$ es la eficiencia incremental generalizada del algoritmo paralelo comparando los tiempos de ejecución $t(p)$ con p procesadores y $t(q)$ con q procesadores siendo $p < q$.

- Trabajo requerido del algoritmo paralelo

Spencer (1997) mide el aprovechamiento de la potencia de cálculo de los procesadores disponibles mediante una expresión diferente de la aceleración y de la eficiencia. Lo llama trabajo requerido del algoritmo paralelo, que es el producto procesador-tiempo. El mejor algoritmo paralelo, en cuanto al trabajo requerido, según Galil (1984), es aquel que necesita un trabajo proporcional al tiempo empleado por el mejor algoritmo secuencial. Si el trabajo que realiza un algoritmo paralelo ejecutado mediante n procesadores es $\mathrm{O}(t(1)\log^k n)$, donde $t(1)$ es el tiempo empleado por el algoritmo secuencial y k una constante, se dice que el algoritmo paralelo es eficiente.

- Eficiencia paralela

La eficiencia paralela es el porcentaje del tiempo de ejecución durante el que los procesadores están calculando en vez de comunicando. El sistema paralelo (ordenadores + red de comunicaciones + programa paralelo) será tanto mejor cuanto más tiempo emplee en cálculos y menos en la comunicación. Los programas paralelos constan de una parte secuencial, con sólo aquellas funciones y procedimientos que deben ejecutarse secuencialmente y otra paralela que puede acelerarse empleando muchos procesadores. T_s es el tiempo empleado en la parte secuencial y T_{cpu} el tiempo empleado en la parte paralela en un solo procesador, el tiempo de ejecución en un procesador es $T(1) = T_s + T_{cpu}$. Con n procesadores, hay tiempo de comu-

nicación, pero a la vez se reduce el tiempo en cada procesador: $T(n) = T_s + \frac{T_{cpu}}{n} + T_{comm}(n)$. Como la eficiencia es el ratio entre el tiempo con un procesador y n por el tiempo con n procesadores:

$$\epsilon = \frac{T(1)}{nT(n)} = \frac{1+\frac{T_S}{T_{cpu}}}{1+\frac{nT_S}{T_{cpu}}+\frac{nT_{comm}(n)}{T_{cpu}}} \cong \frac{1}{1+f(n,m)\times\frac{\tau_{comm}}{\tau_{cpu}}} \quad (6)$$

Según Schmidt-Voigt (1994), la aproximación es válida si el tiempo secuencial es lo suficientemente pequeño. τ_{comm} es el tiempo necesario para transferir una palabra entre procesadores y τ_{cpu} el tiempo típico para ejecutar una operación en coma flotante. $f(n, m)$ es una función que depende del algoritmo y la dimensión del problema medida por m y el número de procesadores, n.

2.3.3 Mejoras en la robustez

En este contexto, se define la robustez del algoritmo paralelo como la capacidad de encontrar soluciones de calidad ante diferentes instancias del mismo problema, ya que su resolución mediante metaheurísticas es muy sensible a cada problema concreto. La computación en paralelo permite (al menos en teoría) explorar de forma mucho más eficaz el espacio de soluciones en un tiempo similar al empleado por el algoritmo secuencial, con lo que se obtiene una mejora en la calidad de las soluciones obtenidas con respecto a la que se obtendría únicamente con el proceso secuencial incluso empleando el mismo tiempo total. Un algoritmo paralelo basado en búsqueda local podrá encontrar soluciones de más calidad que el algoritmo secuencial para distintas instancias del mismo problema sin necesidad de reajustar los parámetros de la búsqueda.

Existen numerosas aplicaciones en las que se demuestra que la resolución en paralelo de problemas mediante metaheurísticas no sólo mejora los tiempos de resolución, sino que mejora la robustez de las soluciones obtenidas (ver Cung *et al.*, 2001, Crainic y Toulouse, 1998 y Crainic *et al.* 2002). Cung *et al.* (2001) hablan de la robustez que aporta emplear distintas combinaciones de los parámetros de búsqueda en

cada procesador. En este sentido, Crainic y Gendreau (2002) comentan que para conseguir la necesaria robustez del algoritmo paralelo no es suficiente con que cada proceso comience la exploración a partir de una solución diferente, sino que es necesario el empleo de distintos parámetros en cada proceso. En su caso, se corroboró observando el número de veces que se alcanzaba el óptimo ante diferentes instancias del problema estudiado. En otro trabajo (ver Crainic *et al.*, 2002) se emplea un procedimiento más elaborado como indicación de la robustez de una combinación de parámetros del algoritmo: Se hizo funcionar el algoritmo con varias combinaciones de parámetros sobre diez instancias del problema estudiado. Se otorgó tres puntos al algoritmo que encontró el mayor número de mejores soluciones, dos al que encontró más veces la segunda solución y uno al que encontró más veces la tercera mejor solución del problema a lo largo de los diez problemas. Se tomó como más robusto el algoritmo con la combinación de parámetros que obtuvo mejor puntuación sumando las puntuaciones obtenidas con cada instancia del problema.

Por tanto, las soluciones de partida y los parámetros de ejecución tienen una gran importancia en la calidad de las soluciones obtenidas y ha de ser tenido en cuenta en el desarrollo de algoritmos paralelos.

En nuestra investigación, se implementará un algoritmo paralelo en el que cada proceso, no sólo parte de una solución diferente, sino que emplea parámetros diferentes en su ejecución.

2.4 Inconvenientes de la computación paralela

A pesar de las ventajas comentadas anteriormente, sobre la computación paralela se han presentado diferentes objeciones. Se resumen, a continuación, cinco de las más conocidas citadas por Quinn (1987):

- La ley de Grosch (1953), que indica que la velocidad de los superordenadores es proporcional al cuadrado de su coste. En estas condiciones, sería mejor emplear el dinero en un ordenador más potente que en dos menos potentes y conectarlos.

- La conjetura de Minsky (Minsky y Papert, 1971), comenta que la máxima aceleración que puede alcanzarse con un número grande de procesadores es proporcional al logaritmo del número de procesadores.
- La ley de Moore (1965), indica que la potencia de los ordenadores se va multiplicando aproximadamente por dos cada dos años sin incremento de su coste por lo que los esfuerzos en desarrollar algoritmos paralelos, serían compensados por la existencia de ordenadores más potentes en un corto espacio de tiempo.
- La ley de Amdahl (1967) advierte que la existencia de operaciones secuenciales limita la aceleración que puede alcanzarse con un algoritmo paralelo según la expresión (7).

$$S(p) \leq \frac{1}{f+\frac{1-f}{p}} \qquad (7)$$

Donde $S(p)$ es la aceleración alcanzable con p procesos al ejecutar un algoritmo paralelo cuya fracción serie o proporción de operaciones secuenciales es f. Este es uno de los mayores argumentos en contra del paralelismo, y sirve de hecho para ver cuándo un algoritmo puede beneficiarse o no de su ejecución en paralelo. Esta expresión se ha estudiado de forma sistemática en el campo de la computación paralela y, en muchos casos, ha sido actualizada e incluso refutada, pero la idea principal sigue siendo la misma: el algoritmo paralelo tiene una fracción secuencial que impide que la aceleración sea superior a un valor proporcional a esa fracción. Si se representa la eficiencia en lugar de la aceleración en función del componente secuencial y del número de procesadores, se obtiene el gráfico de la Figura 3.

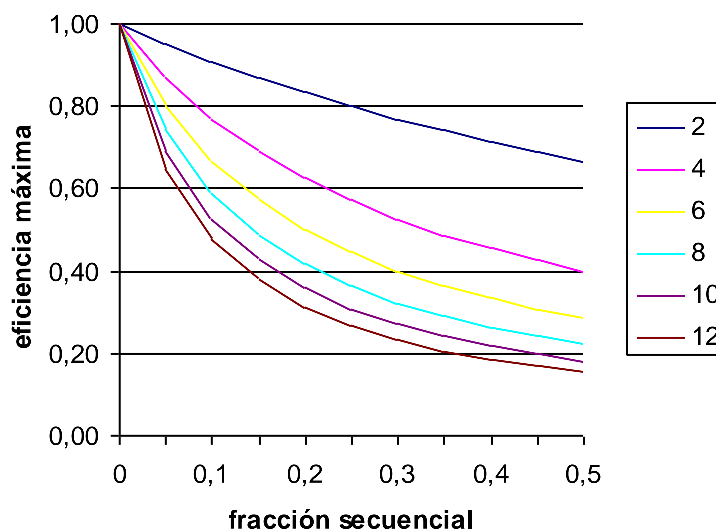


Figura 3: Eficiencia máxima de un algoritmo paralelo según la ley de Amdahl

Conforme aumenta el número de procesadores, el algoritmo paralelo se ve afectado de forma más negativa por la fracción secuencial del algoritmo. Entonces, si la fracción secuencial del algoritmo es importante, se necesitará un clúster formado por unos pocos ordenadores más potentes que hagan rápido este trabajo. Como se cita en Quinn (1987), esta es la aproximación tipo "senda de los elefantes". En otros casos, con una fracción secuencial menos importante, es mejor seguir la aproximación tipo "ejército de hormigas", con sistemas compuestos por muchos procesadores de menor potencia, ya que la aceleración se degradará muy poco conforme aumenta el número de procesadores empleados.

En definitiva, todas estas objeciones deben servir para poner de manifiesto la necesidad de un diseño cuidadoso de las aplicaciones en paralelo que aproveche realmente la mayor potencia de estos sistemas.

2.5 Plataformas paralelas de computación

Para la ejecución de algoritmos paralelos es necesario disponer de una plataforma o conjunto de sistemas físicos o hardware y del software adecuado. La estructura típica de un sistema para computación paralela se puede organizar, como cualquier

sistema informático, en un conjunto de capas, cada una de las cuales tiene a su vez una estructura más o menos compleja (ver Figura 4). En lo que sigue, a este sistema lo denominaremos plataforma paralela de computación.

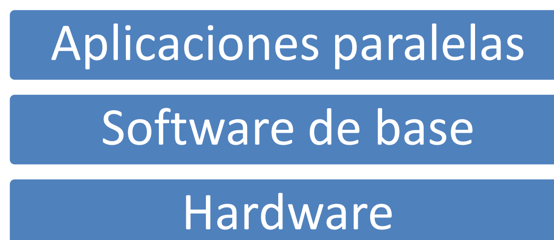


Figura 4: Estructura general de un sistema informático para computación paralela

A continuación, describiremos en detalle cada uno de los elementos anteriores. Tomaremos como referencia las clasificaciones de plataformas paralelas propuestas por Dongarra *et al*. (2003), y por McBrian (1994).

2.5.1 Arquitecturas hardware

A continuación se mencionan algunos de los sistemas físicos más usuales para la ejecución de tareas en paralelo. Dentro de los sistemas considerados, se tendrán en cuenta tanto equipos físicos que contienen más de un procesador o procesadores con más de una CPU, como propuestas de sistemas paralelos abstractos.

No se van a considerar dentro de esta clasificación los sistemas monoprocesador, si bien sobre ellos también se pueden implementar aplicaciones paralelas con alguno de los sistemas operativos multiproceso actuales. Tampoco se incluyen en esta clasificación a los sistemas distribuidos, como los *grids* de ordenadores heterogéneos.

La clasificación más empleada suele ser la de dividir los equipos según el acceso de los procesadores a la memoria. Sobre el acceso a memoria, se suelen emplear para medir sus prestaciones los conceptos de latencia y ancho de banda. Latencia es el espacio de tiempo necesario para que un procesador obtenga de la memoria el dato que desea. Ancho de banda es el número de bytes por unidad de tiempo que se pueden traspasar desde la memoria al procesador a la máxima velocidad.

Estas medidas suelen complicarse por la existencia de jerarquías o niveles de memoria. Así, es normal encontrar en sistemas monoprocesador estructuras como la mostrada en la Figura 5. En ellas, mientras que el acceso a la memoria caché de primer nivel lleva un retraso de 2 a 5 ciclos de reloj, el acceso a la caché de segundo nivel se retrasa de 10 a 20 ciclos de reloj y el acceso a la memoria del sistema de 100 o más ciclos.

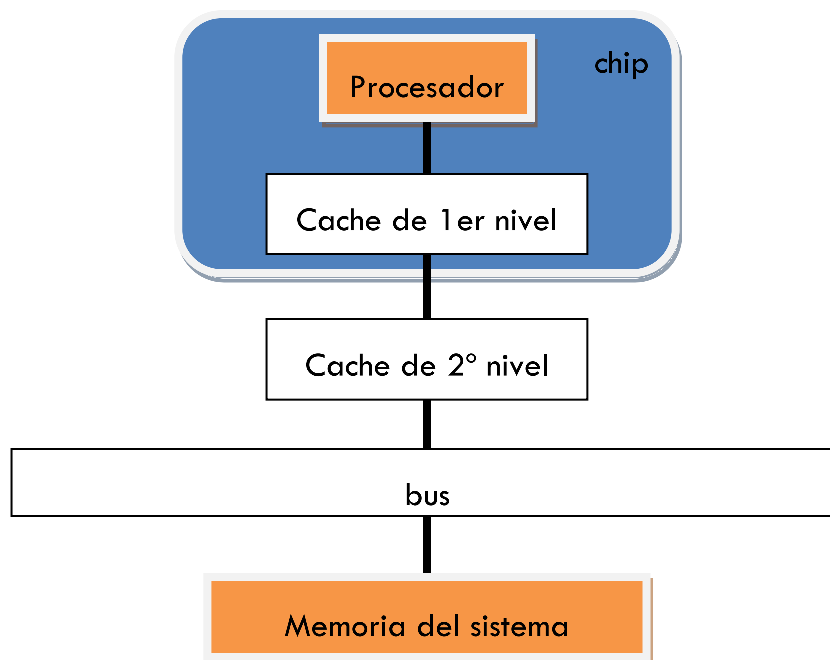


Figura 5: Jerarquías de memoria (adaptado de Dongarra *et al.*, 2003)

A continuación, se comentan con más detalle los tipos de arquitecturas hardware sobre las que se suelen implementar aplicaciones en paralelo. Tres de ellas son arquitecturas físicas: sistemas multiprocesador con memoria compartida, sistemas con memoria distribuida y sistemas masivamente paralelos. El cuarto tipo son los modelos abstractos, sobre los que se han desarrollado aplicaciones que tratan de estudiar a priori cuáles son las características que puede tener un algoritmo paralelo real.

2.5.1.1 Sistemas multiprocesador con memoria compartida

Los sistemas multiprocesador son ordenadores que disponen de varios procesadores que pueden programarse de forma independiente. Se caracterizan por disponer de una memoria compartida, centralizada o común (Quinn, 1987). En este tipo de sistemas, cuando un procesador escribe en la memoria, los datos quedan inmediatamente accesibles al resto de procesadores.

El hecho de que todos los procesadores puedan acceder a la memoria hace que estos sistemas sean poco escalables, y que el acceso a memoria se convierta en un cuello de botella cuando se incrementa el número de procesadores. En Aggarwal *et al.* (1989) se citan como ejemplos de sistemas con memoria compartida el BBN *Butterfly* (Rettberg y Thomas, 1986) y el IBM RP3 (Pfister *et al.*, 1985), donde el acceso a la memoria ocupa del orden decenas de ciclos de instrucción, mientras que los sistemas con memoria distribuida tienen una latencia de cientos o miles de ciclos de instrucción (Karp, 1987).

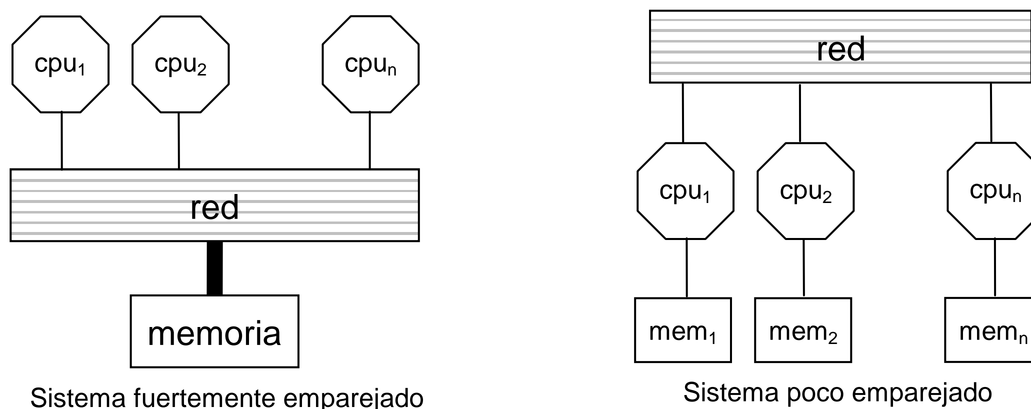


Figura 6: Sistemas multiprocesador de memoria compartida según Barr y Hickman (1993)

Barr y Hickman (1993) separan los sistemas multiprocesador con memoria compartida en fuertemente emparejados (*tightly coupled*) y sistemas débilmente emparejados (*loosely coupled*). Los sistemas fuertemente emparejados son aquellos en los que el tiempo necesario para acceder a cualquier zona de la memoria es el mismo para todos los procesadores, mientras que en los sistemas débilmente emparejados, el tiempo de acceso a la memoria depende de la cercanía del procesador a la zona

concreta de la memoria (ver Figura 6). Los sistemas fuertemente emparejados no pueden ser empleados más que en sistemas multiprocesador con memoria compartida. Un ejemplo de ello son los sistemas con arquitectura UMA (*Uniform Memory Access*). Los sistemas poco emparejados están más cerca de los grupos de ordenadores o sistemas multiordenador considerados en el apartado 2.5.1.2.

Dos ejemplos de sistemas poco emparejados son las arquitecturas NUMA (*Non-Uniform Memory Access*, Bolosky *et al.*, 1991), CC-NUMA (*Cache-Coherent Non-Uniform Memory Access*, Chapin *et al.*, 1995) y COMA (*Cache Only Memory Architecture*, Hagersten *et al.*, 1992). La arquitectura NUMA trata de evitar el cuello de botella que se produce en el bus de memoria de sistemas con muchos procesadores, cuando tratan de acceder a la memoria. La arquitectura NUMA coloca una parte de memoria cerca de cada procesador (típicamente se coloca memoria, procesador y quizá sistema de entrada/salida en una tarjeta y el sistema lo conforman varias tarjetas).

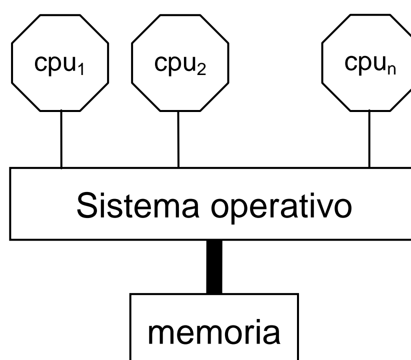


Figura 7: Sistema multiprocesador con sistema operativo multiproceso según Brawer (1986)

En Brawer (1986) se menciona un modelo abstracto de sistema paralelo fuertemente emparejado, aunque en ese caso se sustituía la red por el sistema operativo (ver Figura 7).

2.5.1.2 Sistemas multiordenador con memoria distribuida

Otra posibilidad es emplear multiordenadores, grupos o clústeres de ordenadores, bien de bajo costo o bien soluciones proporcionadas por los principales fabricantes

de hardware. Los multiordenadores o clústeres, se diferencian de los multiprocesadores en que cada procesador de un clúster dispone de una memoria particular y toda la comunicación entre procesos debe hacerse mediante mensajes (Quinn, 1987). Son, por tanto, sistemas con memoria distribuida.

Al hablar de clústeres de ordenadores se suelen distinguir, según el uso que se vaya a hacer de los mismos en clústeres de alta disponibilidad, clústeres para equilibrado de carga de trabajo y clústeres para procesamiento de altas prestaciones mediante intercambio de mensajes (IBM DeveloperWorks, IBM, 2007).

Se pueden conseguir equipos de altas prestaciones con un menor desembolso económico empleando ordenadores menores unidos mediante una red de comunicaciones (Benett *et al.*, 1999). El paradigma de este tipo de sistemas de bajo costo son los equipos basados en componentes comerciales o *Commercial Off-The-Shelf* - COTS. A su vez, los clústeres pueden estar formados por ordenadores con más de un procesador cada uno.

2.5.1.3 Arquitecturas masivamente paralelas

Otras posibilidades son las arquitecturas masivamente paralelas empleadas por Taillard (1994) o por Barr y Hickman (1993) o las redes de *transputers*, empleadas por Taillard (1990).

Los *transputers* (*transistor computer*) son elementos de computación que constan de una unidad de proceso, una pequeña cantidad de memoria y elementos de conexión que le permite conectarse simultáneamente a varias unidades del mismo tipo. De este modo, se dispone de un elemento de bajo coste con el que se pueden construir sistemas informáticos masivamente paralelos. Aunque la primera referencia a un *transputer* se encuentra en el libro de Stewart y Sulley (1966), los primeros *transputers* comerciales fueron producidos en 1983 por la empresa INMOS. Sus prestaciones eran superiores a las de los ordenadores con microprocesadores convencionales de su época (Barron *et al.,* 1983).

Otro sistema masivamente paralelo, similar al anterior, es el nCUBE, compuesto por procesadores nCUBE 2S. Para más detalles sobre la arquitectura del procesador nCUBE S2 se puede consultar el trabajo de Schmidt-Voigt (1994).

El principal inconveniente de los *transputers* y otros sistemas masivamente paralelos es su relación prestaciones precio más desfavorable que otros sistemas empleando procesadores actuales.

2.5.1.4 Modelos paralelos abstractos

Según Juurlink y Wijshoff (1996), los modelos paralelos abstractos son los siguientes:

- PRAM: Parallel Random Access Memory (Fortune y Willie, 1978)

PRAM es el modelo abstracto de computador paralelo más comentado en la literatura. El PRAM es un modelo abstracto para el desarrollo inicial de aplicaciones en paralelo que evita considerar particularidades del hardware sobre el que se va a implementar el programa y posteriormente puede simularse en un sistema real – ver Figura 8, (Gibbons y Ritter, 1988). Es una forma simple de entender cómo funciona un ordenador paralelo (Hughes y Hughes, 2004). Sobre este ordenador virtual se pueden emplear algoritmos de cuatro tipos según la forma en que acceden a la memoria compartida:

  - EREW – Exclusive Read, Exclusive Write
  - CREW – Concurrent Read, Exclusive Write
  - ERCW – Exclusive Read, Concurrent Write
  - CRCW – Concurrent Read, Concurrent Write

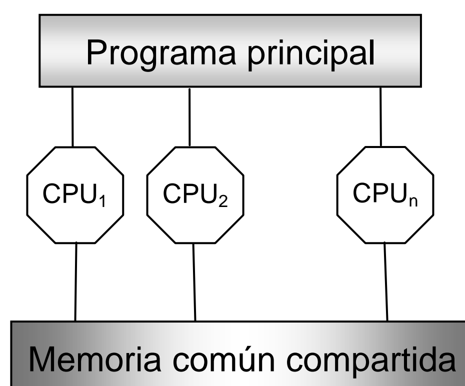


Figura 8: Esquema de PRAM (Gibbons y Ritter, 1988)

Se trata de un modelo abstracto de ordenador multiprocesador con memoria compartida en el que los procesadores funcionan de forma síncrona. Su defecto es que no considera el tiempo de la comunicación entre procesos.

- BSP: Bulk Synchronous Parallel model (Valiant, 1990).

Valiant (1990), observando que la mayoría de los desarrollos informáticos siguen orientados a la computación secuencial, propone el BSP como modelo de computación paralela puente entre la computación secuencial y la computación paralela, de forma que se puedan aprovechar desarrollos secuenciales como base para desarrollos paralelos.

- MP-BPRAM: Message Passing Block PRAM (Aggarwal *et al.,* 1989).

En Aggarwal *et al.* (1989) se propone un modelo de ordenador paralelo basado en el modelo PRAM que permite las transferencias de bloques de información completos, de forma que se reducen las pérdidas de tiempo debidas a la latencia de la comunicación.

- E-BSP: Extended BSP (Juurlink y Wijshoff, 1995).

Es un modelo que supera alguna de las limitaciones del BSP permitiendo que la comunicación se lleve a cabo aún cuando la cantidad enviada de datos no coincida con la cantidad recibida. Además emplea el concepto de memoria local, añadiendo retrasos cuando se trata de acceder a datos que no se encuentren en la memoria de cada procesador, sino que deben recogerse de la memoria de otro procesador.

- LogP (Culler *et al.,* 1993)

LogP toma su nombre de las características del sistema paralelo que modela: el retraso de la comunicación o latencia (L), la sobrecarga que provoca la comunicación en el sistema (o – *overhead*), el ancho de banda de la comunicación (g) y el número de procesadores (P). Los autores opinan que estos son los parámetros que permiten adaptar un algoritmo paralelo a diferentes tipos de sistema informático, y por tanto deben tenerse en cuenta desde las etapas de diseño del algoritmo. De este modo, afirman que su modelo se acerca más a la realidad que el modelo PRAM al tener en cuenta los tiempos empleados en la comunicación y permite averi-

guar mejor dónde pueden estar los cuellos de botella del algoritmo paralelo que se está diseñando antes de implementarlo en un sistema paralelo real.

2.5.1.5 Conclusiones y solución adoptada

De las arquitecturas hardware descritas, destacan los sistemas con memoria distribuida por su relación más favorable entre prestaciones y precio. Con mayor coste por procesador y mayor fiabilidad por la integración de sus componentes están los sistemas poco emparejados comerciales como, por ejemplo, los sistemas tipo *blade*. Los sistemas tipo *blade* de los fabricantes Sun e IBM se descartaron para este trabajo por el enorme desembolso que suponían. El resto de propuestas tienen menos aplicabilidad en empresas de tamaño pequeño o mediano bien por su coste superior a las propuestas anteriores, o por su escasa difusión, como las redes de *transputers*. Por todo ello, nos hemos decantado por una arquitectura hardware basada en un clúster de ordenadores monoprocesador.

Así, hemos configurado un sistema paralelo mediante una red Gigabit Ethernet con 12 PCs Dell Poweredge 750. Cada uno de estos PCs dispone de un procesador Pentium IV funcionando a 3.2 MHz, 1 GB de memoria RAM y dos tarjetas de comunicación tipo Gigabit Ethernet, según el esquema que se muestra en la Figura 9.

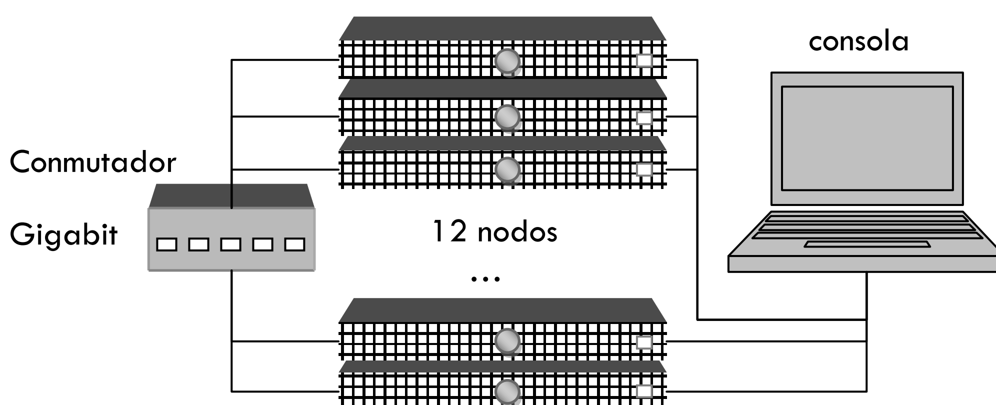


Figura 9: Esquema del clúster empleado en esta investigación

El sistema operativo empleado en los equipos es Linux, basado en la distribución Debian 3.0. Se ha configurado el núcleo del sistema operativo de forma que se han eliminado los servicios y controladores innecesarios además de dar mejor soporte a los dispositivos disponibles, pasando de la versión 2.4 original de la distribución Debian 3.0 a la versión 2.6 del kernel. Así, se minimizan las interferencias que sobre el funcionamiento del mismo podrían tener otros procesos en funcionamiento y además se consigue optimizar el rendimiento del sistema operativo para la arquitectura disponible. Esta adaptación ha significado un trabajo extra para la preparación de los experimentos, ya que, en el momento en el que se comenzó la instalación del sistema, no existía una distribución de Linux completamente adaptada a los equipos que se emplean en este clúster. Se ha debido reconfigurar el kernel para el procesador tipo *Intel Pentium IV*, para el disco duro tipo *SATA*, para las tarjetas de comunicaciones tipo *Gigabit Ethernet* y para poder emplear dispositivos de almacenamiento externo que permitieran el mantenimiento del sistema.

De entre los sistemas formados por servidores, tras el estudio de ofertas de Sun, Intel y Dell, se seleccionó esta última por ofrecer el mayor número de procesadores dentro del presupuesto disponible. La oferta de Sun incluía equipos con doble procesador a un coste por cada ordenador muy superior al de dos ordenadores monoprocesador de los otros fabricantes. En este sentido, se ha cumplido la experiencia de que la relación coste/potencia de los equipos multiprocesador es peor que el de ordenadores PC o *Workstation* funcionando en paralelo. Los argumentos en contra de esta decisión suelen ser la menor fiabilidad de los equipos pequeños en paralelo o de ser sistemas no especialmente preparados para este cometido. Esto no ha influido en la decisión, ya que los equipos adquiridos pertenecen a fabricantes de experiencia contrastada. Sobre la red de comunicaciones empleada, el equipo de interconexión también pertenece a uno de estos fabricantes y sus prestaciones son suficientes comparadas con el bus interno de los equipos, no suponiendo un cuello de botella en la comunicación. En el caso de los equipos empleados, la velocidad de comunicaciones del bus PCI Express del sistema de cada uno de los ordenadores es del orden de la velocidad de la red de comunicaciones Gigabit Ethernet.

2.5.2 Software de base

En cuanto a soluciones software, se encuentran diversas opciones que pueden hacer uso de las arquitecturas hardware descritas en el apartado 2.5.1. Partiendo del sistema operativo, existe la posibilidad de configurar cada nodo de un clúster con su sistema operativo independiente del resto o emplear un sistema operativo que contemple a todos los nodos del sistema. La mayoría de sistemas operativos actuales, incluso los de ámbito personal, son capaces de aprovechar la existencia de más de un procesador en cada ordenador o incluso de aprovechar microprocesadores con más de una unidad central de proceso o CPU.

Ya se han comentado las posibilidades de dividir las tareas que debe realizar un programa en procesos o en hilos. Empleando lenguajes de propósito general, existen funciones que permiten dividir la ejecución en hilos o en procesos. Para facilitar la tarea de la división de la ejecución en procesos, se pueden emplear librerías añadidas a los lenguajes de propósito general, como las disponibles en HPF (*High Performance Fortran*) comentada en el apartado 2.5.2.6 o las que se emplean para pasar mensajes entre procesos como PVM (*Parallel Virtual Machine*) o MPI (*Message Passing Interface*), ver apartado 2.5.2.6. Tanto PVM como MPI incluyen funciones para iniciar y detener procesos, para crearlos y destruirlos o para agruparlos en conjuntos de procesos que se comunican entre si.

En los siguientes apartados se comentarán las posibilidades, partiendo del nivel más cercano a la arquitectura física del sistema hasta el nivel más cercano a las aplicaciones:

- Sistemas operativos
- Herramientas middleware
- Hilos
- Llamadas a procedimientos remotos
- Sockets
- Interfaces de programación paralela
- Lenguajes de programación paralela

2.5.2.1 Sistemas operativos

Dejando de lado los sistemas multiprocesador y los sistemas masivamente paralelos, en los que se emplea un solo sistema operativo, en el empleo de ordenadores de bajo coste para configurar sistemas paralelos, existen dos tendencias principales. La primera consiste en que cada nodo ejecuta su propio sistema operativo, mientras que en la segunda, sólo se instala el sistema operativo en uno de los nodos de forma que el resto de nodos se inicia cargando el sistema operativo desde el primero.

Mosix y *Beowulf* son estos dos paradigmas de computación concurrente. *Mosix* (Mosix, 2006, *OpenMOSIX* – The OpenMosix Project, 2006) para computación distribuida y *Beowulf* (The Beowulf clúster Site, 2007, Sterling *et al.*, 1995) para computación paralela. En *Mosix*, es el sistema operativo el encargado de repartir las tareas, mientras que en un clúster *Beowulf* las tareas deben separarse mediante la programación de las mismas.

No se han considerado herramientas comerciales como por ejemplo, Windows HPC Server 2008 o Windows Compute Cluster Server 2003 (Microsoft, 2007), IBM Cluster Systems Management (IBM, 2007) o Sun Cluster (Sun, 2007) por su elevado coste, su escasa difusión en el momento de comenzar este trabajo o por estar más orientadas a sistemas de alta disponibilidad o tolerancia a fallos más que a la computación de altas prestaciones.

2.5.2.2 Herramientas middleware

En Buyya (1999) se presenta una arquitectura de clúster de PCs o de estaciones de trabajo en la que éstas se pueden agrupar bajo una capa de software (*middleware*) que permite abstraer la arquitectura física de la arquitectura lógica del sistema sobre el que se ejecutan las aplicaciones, sean paralelas o secuenciales (ver Figura 10). De esta forma, las aplicaciones "ven" al clúster como un solo ordenador (SSI – *Single System Image* u OSI – *One System Image*).

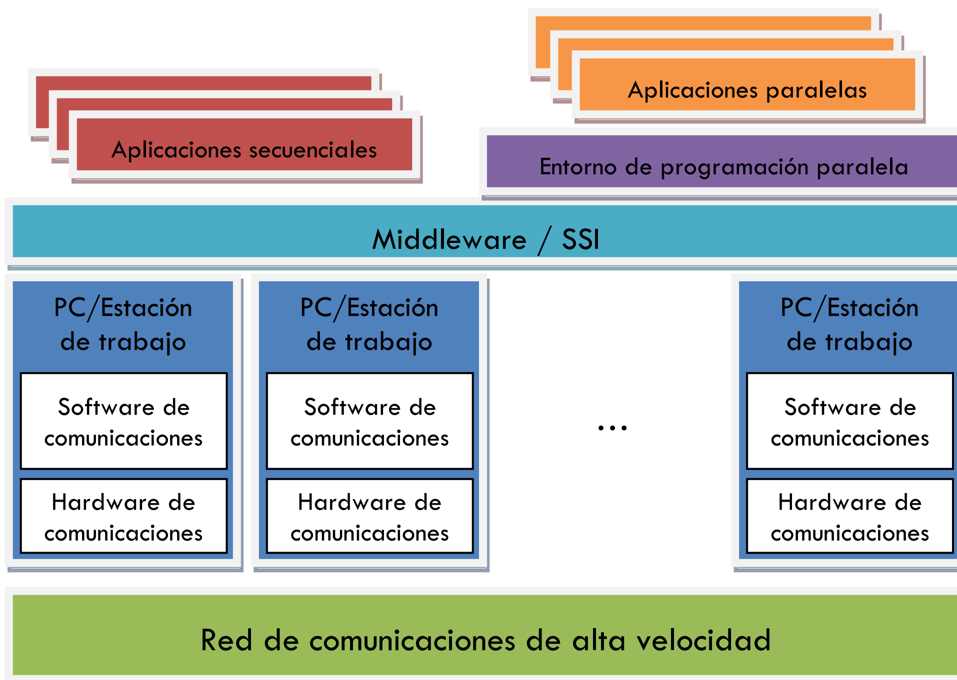


Figura 10: Esquema de clúster según Buyya (1999)

2.5.2.3 Hilos o threads

La división de la ejecución de cualquier aplicación puede hacerse mediante hilos, que bifurcan la ejecución en varias tareas que se ejecutan de forma simultánea o seudo simultánea dependiendo de las posibilidades tanto del sistema operativo como del hardware sobre el que se ejecuta. En el apartado 2.2.1 se han descrito con más detalles los hilos.

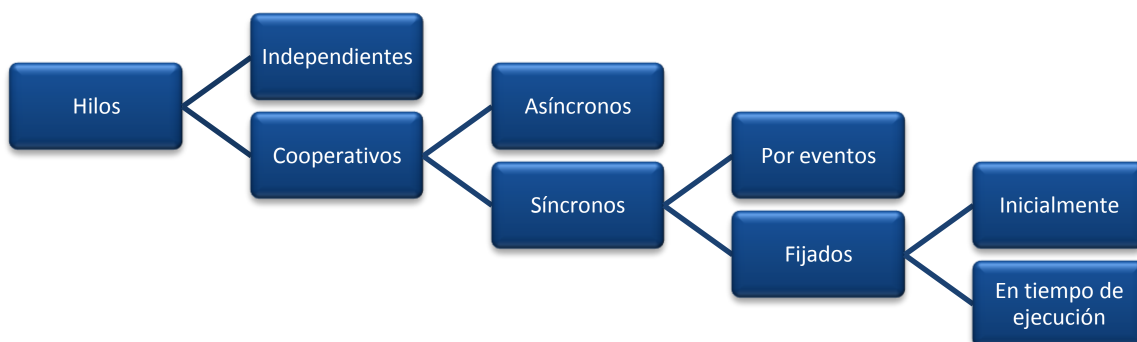


Figura 11: Tipos de hilos de ejecución en paralelo

Algunos lenguajes de programación orientados a objetos (como Java) disponen de clases que permiten, con relativa facilidad, dividir la ejecución de aplicaciones mediante hilos. De forma esquemática, en la Figura 11 pueden distinguirse los diferentes tipos de hilos de ejecución citados por Crainic y Toulouse (2002) de la siguiente forma:

Normalmente, las aplicaciones que emplean procesos o hilos que cooperan, suelen dar los mejores resultados en metaheurísticas (Crainic y Toulouse, 2002) frente a aplicaciones paralelas en las que se emplean procesos independientes.

2.5.2.4 Llamadas a procedimientos remotos

Las llamadas a procedimientos remotos (RPC-*Remote Procedure Call*) consisten en la posibilidad de lanzar aplicaciones desde un ordenador en otro diferente. La forma más inmediata es el empleo de objetos proporcionados por lenguajes de programación general como C++ y Java. Este último ha tenido más difusión gracias a su portabilidad y su empleo en servicios web.

Aparte de para programación multihilo, el lenguaje Java dispone de objetos para la llamada a procedimientos remotos que se conocen como RMI (*Remote Method Invocation*). Estos objetos pueden emplearse en sistemas multiordenador. La contrapartida a la sencillez de su implementación y a su portabilidad está en su elevada latencia. Según Maassen *et al.* (2001) esta latencia es de 1316 μs para una llamada sin argumentos en un clúster de 32 nodos con una red Myrinet frente a los 31 μs conseguidos al emplear RPC mediante lenguaje C. Esto ha relegado a RMI a aplicaciones basadas en web, donde estos tiempos de latencia son admisibles. Estos autores proponen una implementación de RMI compilada, de forma que se reduce drásticamente la sobrecarga de la comunicación y hace competitivo el empleo de RMI para la computación de altas prestaciones. En este caso, no sólo C++ sino Java podrían emplearse para programación paralela de altas prestaciones.

2.5.2.5 Sockets

Los sockets (ver Comer y Stevens, 1993, o Stanislav, 2007) son canales de comunicación punto a punto entre dos procesos que emplean los protocolos de Internet TCP/IP. Existen librerías para la comunicación mediante sockets en lenguajes de

programación como Java, C o C++ que facilitan la tarea de implementar este tipo de comunicación. Para ello, los intercambios de información entre procesos se programan como operaciones de lectura o escritura similares a las que se realizan con archivos, de forma que el programador puede despreocuparse de temas internos de la comunicación.

La comunicación mediante sockets se basa en el empleo de búferes de datos: Debe igualarse el número de bytes leídos con el número de bytes escritos y viceversa para evitar así el bloqueo de la comunicación. Dado que cada operación de lectura/escritura del socket devuelve el número de bytes afectados, se puede emplear un socket adicional para el control de la comunicación.

Por tanto, frente al empleo de la comunicación mediante llamadas a procedimientos remotos, se tiene más control de la comunicación y por tanto una comunicación más rápida a cambio de mayor complejidad en el desarrollo de aplicaciones.

2.5.2.6 Interfaces de programación en paralelo

En esta sección se presentan las alternativas más usadas para comunicación entre procesos mediante librerías de paso de mensajes. Con estas librerías se facilita la programación de la comunicación entre procesos que se ejecutan en varios ordenadores. Aunque en alguno de los casos se les ha añadido funcionalidad para su funcionamiento en sistemas multiprocesador o masivamente paralelos con memoria compartida, su campo de aplicación fundamental han sido las redes de estaciones de trabajo y los multiordenadores con memoria distribuida.

Hempel *et al.* (1994) hablan de la orientación de los entornos de paso de mensajes a sistemas de ordenadores concurrentes con memoria distribuida. Se han definido lenguajes paralelos, lenguajes de coordinación, extensiones del lenguaje, objetos compartidos y memoria compartida virtual, pero lo más empleado en sistemas distribuidos es el paso de mensajes. Cuando se escribió el artículo, casi cada fabricante de hardware tenía su estándar de paso de mensajes. Otros grupos de investigación han diseñado librerías portables: PVM, p4 y ZipCode son libres, mientras que Express y PARMACS son comerciales. Linda queda un poco aparte porque si bien puede emplearse para paso de mensajes, no es una interfaz pura de paso de men-

sajes sino un sistema para compartir datos. El MPI-forum (2009) es un intento de estándar similar al realizado por High Performance Fortran Forum (2007).

Según McBryan (1994), la mayoría de los ordenadores masivamente paralelos son sistemas MIMD (ver apartado 2.6.2.) con memoria distribuida y casi todos emplean el paso de mensajes. La ventaja del empleo de paso de mensajes según McBryan (1994) es que proporciona dos elementos clave en la programación paralela: la sincronización entre procesos y la posibilidad de leer y escribir en la memoria de los otros procesos.

2.5.2.6.1 *Parallel Virtual Machine* (*PVM*)

PVM o *Parallel Virtual Machine* es una librería para paso de mensajes entre procesos (Sunderam, 1990, Geist *et al.*, 1994, Parallel Virtual Machine, 2007), que permite emplear nodos heterogéneos en la comunicación. Siguiendo esta misma filosofía, se desarrolló la Interfaz de Paso de Mensajes o MPI. Skjellum *et al.* (1994) lo definen como una interfaz portable para sockets y XDR (*External Data Representation* – una especificación de la empresa Sun Microsystems sobre el formato de los datos en la capa de presentación del modelo OSI, sobre todo empleada con el protocolo de llamada a procedimientos remotos de la misma empresa)

En un principio, PVM podía emplearse tanto en redes heterogéneas de estaciones de trabajo como en sistemas multiprocesador con memoria compartida y en sistemas masivamente paralelos y es por tanto válido para sistemas paralelos y para sistemas distribuidos.

La versión 3 de PVM permite, además del intercambio de mensajes entre procesos, gestionar procesos, añadiéndolos o quitándolos del sistema según sea necesario y ser empleado en clústeres heterogéneos.

El modelo de computación mediante PVM y el modelo de la arquitectura de PVM se muestran en la Figura 12. Emplea el mecanismo más rápido de comunicación según en la red donde se ejecute, p.ej. TCP/UDP en redes tipo intranet/Internet.

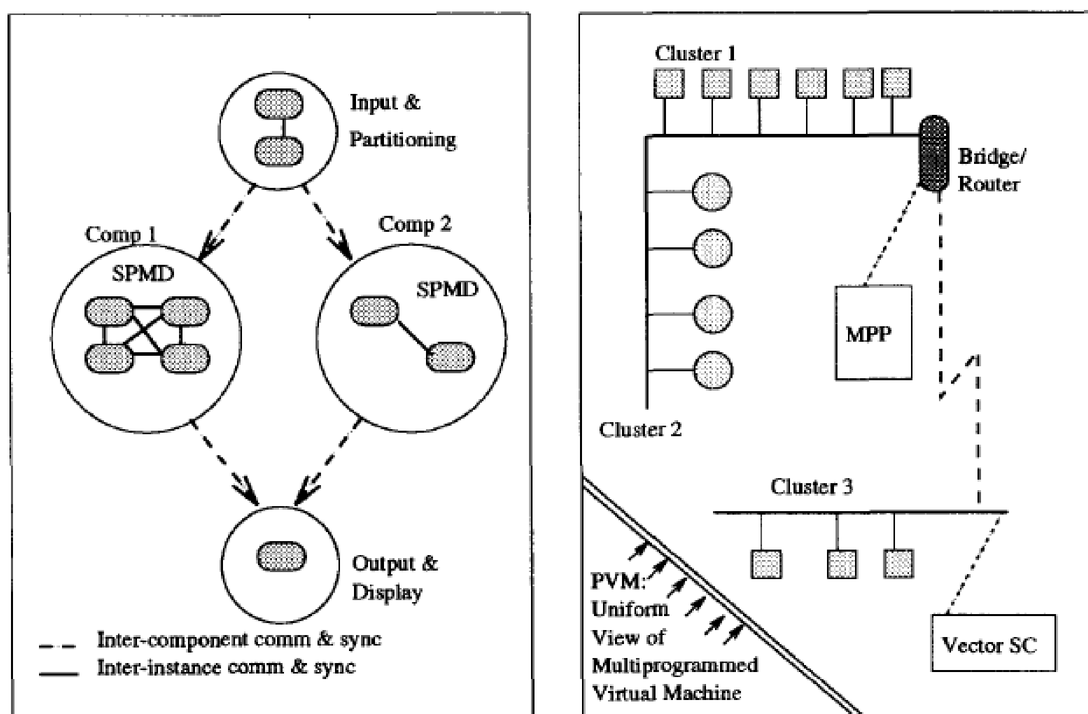


Figura 12: Modelo de computación y modelo de la arquitectura de PVM (Sunderam *et al.*, 1994)

2.5.2.6.2 Unify

Unify (Vaughan *et al.,* 1995) es un sistema para paso de mensajes entre procesos que se encuentra a medio camino entre PVM y MPI. Unify facilita la portabilidad de aplicaciones entre PVM y MPI ya que, funcionando sobre PVM, permite hacer llamadas a algunas de las funciones de MPI. Posteriormente los mismos autores proponen pasar a un entorno PVM sobre MPI que llaman Simplify.

2.5.2.6.3 Message Passing Interface (*MPI*)

Esta es una de las posibilidades más conocidas de implementación de aplicaciones en paralelo, (Gropp *et al.,* 1999). Esta interfaz de programación consiste en un conjunto de librerías que se añaden a lenguajes de propósito general, como C, C++ o Fortran, con las que se logran prestaciones especiales de intercambio de mensajes entre procesos que pueden residir o no en el mismo ordenador. Está orientado a conjuntos de ordenadores en red, no necesariamente en forma de clúster. MPI tiene similitudes con otras librerías como Express, NX/2, Vertex, PARMACS y P4 e incluye características tomadas de CHIMP, Zipcode y EUI de IBM (Walker, 1994).

A continuación se resumen las características principales de MPI:

- Facilidad de implementación y portabilidad. MPI es fácil de implementar y portar entre diferentes plataformas informáticas. Así, se puede realizar el desarrollo y depuración de aplicaciones en un solo ordenador, para pasar después a su despliegue en un clúster de ordenadores o en un conjunto de ordenadores en red, sin necesidad de cambios en la aplicación.
- Posibilidad de lanzar varios programas de forma simultánea. Frente al empleo de procesos o hilos, MPI tiene la ventaja de poder iniciar varios programas ejecutables al mismo tiempo (Hughes y Hughes, 2003), sin necesidad por tanto de crear procesos a partir de un proceso inicial.
- Ocultación de la arquitectura física. Una vez definido el conjunto de procesadores que constituye la red, la arquitectura de la misma es transparente para las aplicaciones desarrolladas. El programador puede concentrarse entonces en las tareas que realizan los procesos y no en qué procesador se ejecuta cada uno de ellos.

Otras de sus prestaciones son la difusión de mensajes o *broadcast*, el que permite realizar operaciones matemáticas y estadísticas con el contenido de los mensajes y detener o reiniciar procesos. Entre sus carencias, cabe comentar las citadas por Walker (1994):

- Aún estando orientado a sistemas multihilo, no dispone de funciones para la programación de hilos.
- No es una herramienta completa de programación en paralelo porque carece de características de depuración en paralelo, de división del programa o de compilación paralela.
- No permite el empleo de mensajes activos o canales de comunicación virtuales de forma explícita.

Otra de las dificultades de emplear la arquitectura MPI es que únicamente puede emplearse el esquema de memoria distribuida. Por tanto, no se beneficia de las posibilidades de la memoria compartida en cuanto a que todos los procesos dispongan de acceso a la memoria común para mantener información relevante.

El esquema de memoria compartida puede imitarse con ciertas dificultades. Una forma es que cada proceso emplee parte del tiempo en procesamiento y parte en hacer disponible la memoria a la que tiene acceso. El problema que se plantea es doble: por un lado se sobrecarga el procesador con el trabajo extra que supone hacer disponible su espacio de memoria. Por otro lado, se sobrecarga la red con el tráfico adicional de mensajes para consultas de memoria entre los nodos. Para disminuir este problema, puede dedicarse una parte de los procesos a servicio de memoria y otra parte a cálculos.

Una de las posibilidades que facilita el desarrollo de aplicaciones en paralelo mediante paso de mensajes es el empleo de entornos gráficos de programación. Existen algunos trabajos en este sentido, por ejemplo, el de Acacio *et al.* (2002), en el que se muestra una adaptación del entorno de desarrollo Delphi para su empleo con bibliotecas de paso de mensajes MPI.

Otro de los entornos de programación desarrollados para aplicaciones de paso de mensajes es el presentado por Stankovic y Zhang (1999) en el que se propone un entorno de desarrollo orientado a objetos, válido para PVM y para MPI, con el que se visualizan tanto los intercambios de datos como el funcionamiento de los procesos como base para posterior depuración de las aplicaciones.

Algunos autores han adaptado MPI a sistemas físicos concretos. Liu *et al.* (2004) han conseguido mejorar la comunicación en redes tipo *InfiniBand* para el empleo de acceso remoto directo a memoria (RDMA – *Remote Direct Memory Access*). De esta forma mejoran la latencia, el ancho de banda, la carga de trabajo de los procesadores cuando se transmiten mensajes pequeños y el tráfico de mensajes de control para mensajes grandes.

Existen varias implementaciones de MPI. Las más conocidas son MPICH, LAM y CHIMP y se describen brevemente a continuación.

Common High-Level Interface to Message Passing (CHIMP) es una implementación de paso de mensajes desarrollada en el Parallel Computing Centre de la Universidad de Edimburgo. Ha tenido una repercusión más limitada que otras interpretaciones de paso de mensajes, probablemente debido a que se encuentra disponible única-

mente para grandes sistemas como Cray, SunOS, Solaris, Iris o AIX. Su desarrollo se detuvo en 1996.

Chameleon (Gropp y Smith, 1993) es una interfaz de paso de mensajes precursora de MPICH. Según estos autores, pretendían que fuera una interfaz que sobrecargara poco el sistema y que sirviera de paso previo a una normalización de las librerías de paso de mensajes orientadas a multiordenadores de memoria distribuida.

La implementación inicial de la librería de paso de mensajes MPICH (MPI + Chameleon) fue desarrollada por Gropp y Lusk (1992) en el Mathematics and Computer Science Division del Argonne National Laboratory del Departamento de Energía de Estados Unidos. El rápido desarrollo de esta versión, una vez conocida la especificación de MPI, se debió a basarse en PVM o P4 y Chameleon. Chameleon proporcionó una serie de herramientas para programación paralela exportables a distintas plataformas y tanto PVM como P4 proporcionaron herramientas de control adicionales para sistemas multiprocesador. El conjunto fue adaptado a la especificación MPI.

En Gropp y Smith (1993) se describen tres versiones de MPICH basadas en tres versiones de interfaz de dispositivo abstracto (ADI – *Abstract Device Interface*). La primera generación de ADI fue preparada para ordenadores masivamente paralelos con sistemas propietarios de intercambio de mensajes y conseguía buenas prestaciones sin sobrecargar el sistema. Era bastante portable en este tipo de sistemas, usando dos implementaciones de ADI, una basada en las librerías propietarias de paso de mensajes de Intel y nCUBE y la otra basada en Chameleon. La segunda generación trataba de conseguir mayor flexibilidad en la implementación y en la comunicación. La tercera generación fue implementada para dar soporte a sistemas que permiten el acceso remoto a memoria.

Una de las implementaciones más extendidas de MPI es LAM (Local Area Multicomputer, 2007), desarrollado en sus orígenes por el Ohio Supercomputer Center y mantenido en la actualidad por el Open Systems Laboratory (OSL) en la Universidad de Indiana, Estados Unidos. En la actualidad se continúan introduciendo mejoras a estas librerías, como la propuesta (y ya incluida en la versión 7 de LAM) por Sankaran *et al.* (2003) para mejorar la tolerancia a fallos y la fiabilidad en la

comunicación entre procesos mediante mecanismos de comprobación y vuelta atrás (o *rollback*). En la actualidad, las librerías LAM-MPI se encuentran también bajo la denominación OpenMPI (2007).

2.5.2.6.4 Linda

Linda (Carriero y Gelernter, 1989, o, más recientemente, Martins *et al.,* 2002) es un conjunto de primitivas para intercambio de mensajes entre procesos que conformarían un lenguaje de programación específico. El diseño de esta interfaz se separa un poco del resto en el sentido de que emplea el concepto de memoria compartida virtual. Sunderam *et al.* (1994) comentan que el diseño de Linda se basa en el concepto de "espacio de tuplas", que es una abstracción de memoria compartida distribuida, empleada para la comunicación entre procesos. En este caso, los procesos se coordinan porque comparten datos, frente al envío y recepción de mensajes entre procesos de la interfaz de paso de mensajes. El uso de memoria compartida simplifica la programación paralela porque imita en cierto modo el comportamiento de un programa secuencial. Gracias a estas primitivas, se podrían emplear sistemas de memoria distribuida como si fuesen sistemas virtuales de memoria compartida, que implicarían desarrollos en paralelo más eficientes.

Carriero *et al.* (1994) mostraron que sus prestaciones son similares y en algún caso mejores en cuanto a escalabilidad que PVM y EUI (*External User Interface*), la interfaz nativa de intercambio de mensajes de IBM.

Como dificultad principal se encuentra la menor portabilidad y escalabilidad a sistemas masivamente paralelos de memoria distribuida de los programas desarrollados empleando memoria compartida virtual. Pese a esto, Carriero *et al.* (1994) resaltan su aplicabilidad a sistemas multiprocesador de memoria compartida, a hipercubos y a redes de estaciones de trabajo (NOW – *Network Of Workstations*), existiendo dos versiones desarrolladas por Scientific Computing Associates, Inc., una orientada a C y la otra a Fortran. En la actualidad, esta empresa comercializa tres productos basados en el sistema Linda de memoria compartida virtual: Linda TCP, Paradise y NetWork Spaces.

2.5.2.6.5 Otros entornos de paso de mensajes

A continuación se citan otras librerías y sistemas de paso de mensajes entre procesos. En primer lugar, se describen desarrollos que hacen hincapié en su portabilidad y en segundo lugar desarrollos propietarios.

- Portable Instrumented Communication Library (PICL): es una librería para comunicación entre procesos. Hace hincapié en la portabilidad, la sencillez de programación y el seguimiento de la comunicación entre procesos. Mediante Paragraph (Blanco *et al.*, 1993) se puede obtener una imagen gráfica de los datos obtenidos con PICL. La versión 1.0 aparece en Geist *et al.* (1990) y la versión más reciente, MPICL (*Portable Instrumentation Library for MPI*) es de 1999.
- P4: Butler y Lusk (1994) describen las librerías P4 para programación en paralelo. Una característica fundamental del P4 es su soporte de múltiples formas de comunicación paralela, que la dota de versatilidad para ser aplicada a diferentes sistemas. También permite gestionar clústeres pero carece de soporte para arquitecturas SIMD (ver apartado 2.6.2.).
- PARallel MACroS (PARMACS) (Hempel, 1991) es un derivado de P4. Es un conjunto muy simple de macros aplicables a C y a Fortran que no sobrecarga la comunicación (Calkin *et al.,* 1994). En este caso se trata de un producto comercial. Calkin *et al.* (1994) hacen hincapié en que los sistemas con memoria distribuida tienen mayores posibilidades de emplearse con éxito para computación paralela de altas prestaciones. PARMACS tiene como ventaja su portabilidad a varios sistemas.
- Zipcode (Skjellum *et al.,* 1994). Este sistema no solo sirve para paso de mensajes sino también para gestión de procesos. MPI ha tomado algunas de sus características como nombres abstractos para los procesos, contextos de comunicación, soporte de grupos estáticos de procesos, *mailers* (comunicadores en MPI) y soporte de topologías virtuales. Se puede emplear en multiordenadores y en redes homogéneas de ordenadores.
- Express: Según Flower y Kolawa (1994), es más un entorno de desarrollo que una librería para paso de mensajes. En sus primeras versiones contenía algunas prestaciones superiores a MPI en cuanto a división automática del trabajo

entre procesadores, desplazando los datos entre los procesadores disponibles, en cuanto a funciones de entrada/salida o en cuanto a funciones para depuración y seguimiento de los programas. Express dispone además de una herramienta que traduce automáticamente código secuencial en código paralelo.

- NX (Pierce, 1994) es una interfaz para aplicaciones en paralelo aplicable principalmente a multiordenadores de Intel. Hace hincapié en el compromiso entre usabilidad y prestaciones. Se trata de llamadas a funciones que se pueden hacer desde C o Fortran. Sus autores presentan como ventajas de esta interfaz una baja latencia y un gran ancho de banda. Como inconvenientes, destacan que no sigue exactamente la especificación POSIX en cuanto a la gestión de errores. Propone HPF (ver apartado 2.5.2.7) para el desarrollo de aplicaciones sobre esta interfaz.

2.5.2.6.6 Librerías propietarias

Englobamos en este apartado librerías diseñadas para sistemas concretos, como las apropiadas a los sistemas nCUBE, CM-5, iPSC/860.

- External User Interface (EUI) (Bala *et al.*, 1994) es un conjunto de librerías de programación en paralelo que se pueden emplear en sistemas paralelos de IBM mediante C o Fortran desde 1993.
- Mensajes Activos y CMMD: Los Mensajes Activos (*Active Messages*) son un mecanismo de comunicación asíncrono (Eicken *et al.,* 1992) que se puede implementar como una librería de paso de mensajes (Tucker y Mainwaring, 1994). Por tanto, se encuentran a un nivel más cercano al procesador que las librerías descritas con anterioridad. Sobre ellos desarrollaron la librería de paso de mensajes CMMD para el sistema CM-5 (C*onnection Machine* – 5).
- Sistema CS-2 (C*omputing Surface* – 2): Los ordenadores del tipo CS-2 de la empresa Meiko son sistemas masivamente paralelos MIMD (ver apartado 2.6.1) con memoria distribuida que emplean paso de mensajes (Barton *et al.,* 1994). Emplea dos modelos de programación paralela diferenciados por la sincronización entre procesos. Uno es SPMD síncrono, empleando lenguajes de

datos paralelos y el otro emplea paso de mensajes. Existe la posibilidad de combinar ambos métodos en una sola aplicación.

- nCUBE 2S: El sistema nCUBE emplea una arquitectura propietaria de paso de mensajes estructurada en tres capas: interconexión, mensaje y encaminamiento (Schmidt-Voigt, 1994).

2.5.2.7 Lenguajes para programación en paralelo

- HPF (High Performance Fortran)

El lenguaje HPF (High Performance Fortran Forum, 2007) es otra de las posibilidades citadas por Martins *et al.* (2002) para desarrollo de aplicaciones en paralelo. La implementación del paralelismo mediante este lenguaje se basa en el paralelismo de datos. Algunos de los lenguajes en los que se basa son Fortran-D y Vienna Fortran. Como una extensión al Fortran 90, contiene una serie de librerías que permiten la ejecución de programas en paralelo enviando conjuntos diferentes de datos a diferentes procesadores. HPF es una herramienta de más alto nivel que otras más antiguas como Mensajes Activos. En la actualidad, ha quedado en desuso, salvo en Japón, donde se emplea en el superordenador Earth Simulator (Kennedy *et al.*, 2007).

2.5.2.8 Conclusiones sobre alternativas software

De entre las alternativas software para implementar computación de altas prestaciones en paralelo, las basadas en librerías de paso de mensajes son las que requieren un menor esfuerzo previo de configuración del sistema. Tanto la configuración de un clúster Beowulf como el empleo de Mosix requieren además del esfuerzo de desarrollo de las aplicaciones una configuración del sistema operativo que no es inmediata.

Comparándolos con las librerías de paso de mensajes, el empleo de sockets para la comunicación entre procesos también permite iniciar y detener dichos procesos, aunque es más complejo en general. La gestión de los errores de comunicación se puede realizar mediante sockets y también al emplear librerías de paso de mensajes.

Por otro lado, tanto el empleo de hilos como el de sockets son procedimientos más complejos para enviar mensajes entre procesos que mediante una librería de paso de mensajes. De las librerías existentes, la de mayor difusión es MPI. Se encuentra disponible en las distribuciones de Linux gratuitas más empleadas, como Debian y sus variantes, Fedora o Suse. Al comienzo de esta investigación únicamente se disponía de la versión LAM en las distribuciones Linux, razón por la que se ha seleccionado para realizar este trabajo. En la actualidad ya se pueden encontrar configuradas versiones de MPICH y de PVM.

No existen muchas comparativas entre las prestaciones de las diferentes librerías de paso de mensajes existentes. Una de ellas es la presentada por Markus *et al.*, (1996), en la que se comparan las prestaciones de MPICH, CHIMP y LAM en su aplicación a la resolución de problemas de ecuaciones en derivadas parciales mediante elementos finitos. Las pruebas se ejecutaron sobre distintos sistemas con memoria distribuida y arquitectura MIMD, llegando a la conclusión de que MPICH saca una ligera ventaja a las otras dos implementaciones. Quizá vale la pena plantearse la migración de LAM a MPICH, pero no consideramos que la ventaja sea significativa, sobre todo teniendo en cuenta que el algoritmo todavía puede ser refinado.

Otra comparación sobre las prestaciones de las librerías de la interfaz de paso de mensajes aparece en Van Voorst y Seidel (2000), esta vez sobre un sistema de memoria compartida, Sun Enterprise 4500. Compara LAM, MPICH y Sun MPI, resultando esta última opción mejor que las dos anteriores por ser una librería adaptada al sistema sobre el que se realizaron las pruebas. No ofrece en cambio conclusiones definitivas sobre las otras dos alternativas, dependiendo las mejores prestaciones de LAM o MPICH del problema concreto.

Por tanto, se va a emplear un esquema de clúster con paso de mensajes para la implementación de nuestro algoritmo en paralelo. De las posibilidades existentes entre las herramientas para intercambio de mensajes entre procesos, se ha seleccionado la librería LAM (*Local Area Multicomputer*) tanto por su disponibilidad como por la sencillez para adaptarla a nuestras necesidades.

Para adaptar el equipo a nuestro entorno de trabajo, hubo que modificar la configuración estándar de LAM en dos puntos:

a) Seguridad en las comunicaciones:

Desde un principio se observó la necesidad de establecer comunicaciones seguras entre los nodos del clúster, ya que el sistema estará conectado a la red local de la Universidad. Para la comunicación ha debido compilarse LAM para así emplear el protocolo SSH v.2 para la autentificación de los nodos en la red. Esta versión es más segura que las anteriores de SSH y mucho más segura que emplear telnet o RSH, ya que la autentificación de la conexión emplea mecanismos de clave privada y pública, que aseguran el tráfico encriptado de toda la información de autentificación tanto de ordenadores como usuarios.

b) Herramientas de análisis

Otra ventaja de la compilación a medida de la aplicación es poder disponer de otros módulos para el análisis de las aplicaciones, como es XMPI, que permite el análisis gráfico del comportamiento del sistema. De este modo, se puede monitorizar el comportamiento de los procesos, sobre todo a la hora de analizar aquellos que están sobrecargados y aquellos que están ociosos. Esto podremos corregirlo asignando distintas cargas de trabajo al variar los parámetros de la ejecución. Esta herramienta se ha empleado únicamente en las primeras fases de desarrollo de los algoritmos.

2.5.3 Herramientas para evaluación de las prestaciones de algoritmos paralelos

Ya se han mostrado en el apartado dedicado a las librerías MPI algunas de las herramientas disponibles para evaluar el rendimiento de algoritmos paralelos implementados con dichas librerías de programación.

Las prestaciones de los algoritmos en paralelo se pueden medir atendiendo a unas pocas variables. Las más estudiadas, como se ha comentado en el apartado 2.3 son la aceleración, bien absoluta o relativa, y la eficiencia del algoritmo paralelo frente al secuencial. Sea cual sea, se debe tener en cuenta la arquitectura del sistema donde se realizan los experimentos. Para un estudio de estas medidas sobre sistemas MIMD, es interesante consultar el trabajo de Barr y Hickman (1993). Estos auto-

res tienen en cuenta los factores que influyen en las variables usadas para medir las prestaciones de los algoritmos paralelos.

Calzarossa *et al.* (2004) diseñan una metodología para medir las prestaciones de algoritmos paralelos que se basa en controlar el tiempo que emplea cada procesador en ejecutar cada una de las tareas y medir la dispersión de estos tiempos con respecto al que deberían emplear si todos emplearan el mismo tiempo en ejecutar la misma tarea. Las tareas con una mayor dispersión, son las mejores candidatas a ser modificadas o refinadas para mejorar el comportamiento del algoritmo.

Sistema	**Fuente**
GNU profiler	Linux
VT (Visualization Tool)	(IBM, 2007)
AIMS (Automated Instrumentation and Monitoring System)	(AIMS, 2007)
	(Yan, 1994)
Paradyn	(Parallel Performance Tools, 2007)
	(Miller *et al.,* 1995)
Paragraph	(Heath y Finger, 2003)
	(Blanco *et al.,* 1993)
SvPablo	(Renaissance Computing Institute, 2007)
	(De Rose *et al.,* 1998)
	(Reed *et al.,* 1998)
VAMPIR (Visualization and analysis of MPI resource)	(Leibniz Rechenzentrum, 2007)
	(Galarowicz y Mohr, 1998)
	(Nagel *et al.,* 1996)
XPVM	(Netlib Repository at UTK and ORNL, 2007)
XMPI	(Local Area Multicomputer, 2007)

Tabla 2: Herramientas para evaluación del rendimiento de aplicaciones en paralelo

Martins *et al.* (2002) separan las herramientas para evaluar las prestaciones de los desarrollos en paralelo en tres tipos: perfiladores, contadores y trazadores de eventos. Los perfiladores permiten conocer el estado de cada uno de los procesos, mostrando los intercambios de información entre los mismos y el tiempo de ejecución de cada uno, bien mediante informes o bien en modo gráfico. De esta forma se

pueden reconocer los procesos ociosos o aquellos que acaparan toda la capacidad de proceso del sistema. Los contadores simplemente registran el número de veces que se produce un suceso cualquiera y los trazadores de eventos registran el instante en que se produce cada evento. Estos dos últimos sistemas pueden precisar algo más de codificación por parte del programador.

En la Tabla 2, se muestran otras herramientas para la evaluación del rendimiento de aplicaciones desarrolladas en paralelo no comentadas previamente.

2.6 Implementación de algoritmos paralelos. Mecanismos de paralelización de algoritmos

En la literatura se han descrito numerosos mecanismos para implementar algoritmos paralelos para resolver diversos problemas. Se comentan a continuación brevemente los métodos más empleados para implementar en paralelo la resolución de problemas en general. Todos estos mecanismos son aplicables a cualquier algoritmo paralelo, por lo que, de hecho, en las aplicaciones reales suelen encontrarse combinaciones de varios de estos procedimientos. En este trabajo, hemos extendido la clasificación de Crainic *et al.* (1997), ver Figura 13, hasta llegar a la mostrada al final de esta sección (ver Tabla 3).

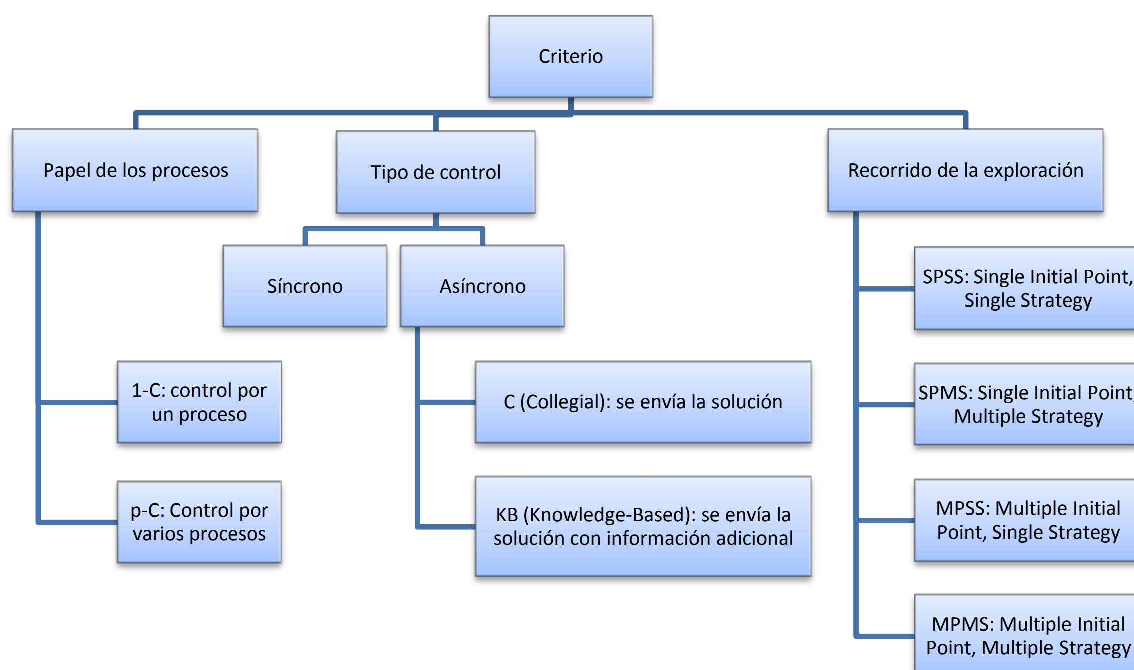


Figura 13: Clasificación de algoritmos de búsqueda en paralelo según Crainic *et al.* (1997)

Los criterios que vamos a usar en este trabajo son:

- Estrategia de exploración del espacio de soluciones
- División del trabajo
- Comunicación entre procesos
- Granularidad

Cada uno de estos criterios es descrito con más detalle en los siguientes apartados.

2.6.1 Estrategia de exploración del espacio de soluciones

Este criterio de clasificación está muy orientado a los métodos de exploración o búsqueda local. Se pueden distinguir los distintos métodos según la estrategia de exploración del espacio de soluciones por cada uno de los procesos (véase Verhoeven y Aarts (1995) o bien Resende *et al*. (1999) para una referencia más completa):

- Algoritmos de pasada única o *single walk*, que realizan una sola exploración en cada procesador, devolviendo la mejor solución encontrada. A su vez, se pueden dividir en:
    - Algoritmos de un solo paso: los procesos evalúan las soluciones vecinas de forma simultánea para después dar un paso.
    - Algoritmos de pasos múltiples: los procesos dan varios pasos de forma consecutiva en el espacio de soluciones.
- Paralelismo de múltiples pasadas o *multiple walk*, en el caso de que cada uno de los procesos explore varios caminos a partir de soluciones que no tienen por qué coincidir de una pasada a la siguiente.

2.6.2 División del trabajo

Esta clasificación se debe a Flynn (1966). Las siguientes alternativas se diferencian en la forma en que reparte el trabajo a los procesadores, si bien las dos últimas, SISD y MISD (Barr y Hickman, 1993), exceden en cierto sentido los límites de la programación paralela.

- SIMD (*Single Instruction Stream, Multiple Data Stream*), utiliza el mismo conjunto de instrucciones en distintos procesadores. No se reduce el número de operaciones, pero hay más procesadores realizándolas. Se conoce también como

paralelismo de datos, al dividir el espacio de soluciones en partes que son exploradas por un proceso distinto cada una. La idea en este tipo de algoritmos es equilibrar la carga de trabajo entre los procesadores (Plastino *et al.*, 2003) de forma que se aproveche toda la potencia de cálculo disponible. Esta forma de división de la ejecución, puede asimilarse a SPMD (*Single Program, Multiple Data*).

- MIMD (*Multiple Instruction Stream, Multiple Data Stream*) carga conjuntos de instrucciones diferentes en cada procesador. Cuando los cálculos que se pretenden realizar son complejos, puede ser adecuado dividir las tareas en segmentos diferentes que se ejecutan por separado. Se conoce también como paralelismo funcional. En el caso de algoritmos de búsqueda local, explora cada parte del espacio de soluciones con diferentes estrategias. Por un lado, se mejora la calidad de las soluciones al emplear no solo diferentes puntos de partida en la exploración sino diferentes métodos (Le Bouthillier y Crainic, 2005). Por otro lado, se dificulta en gran medida la comparación con el algoritmo secuencial, dado que cada uno de los procesos del algoritmo paralelo son diferentes entre sí y también diferentes del algoritmo secuencial.
- SISD (*Single Instruction, Single Data*) se corresponde con arquitecturas de un solo procesador.
- MISD (*Multiple Instruction, Single Data*) se correspondería con una arquitectura en la que se aplican operaciones distintas de forma simultánea al mismo conjunto de datos.

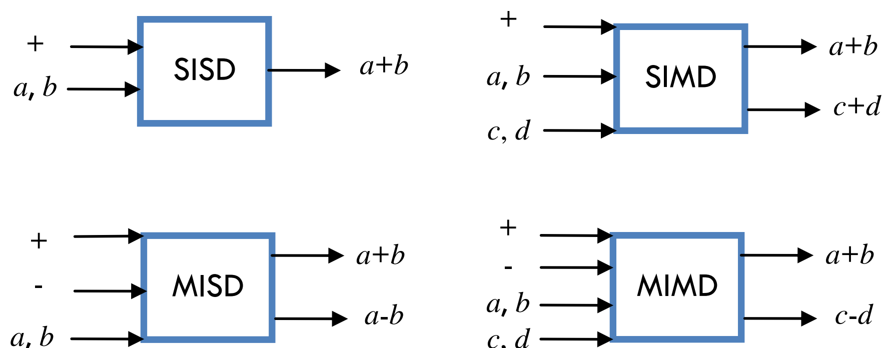


Figura 14: Ejemplo de las diferentes alternativas de Flynn (1966) según Kindervater y Lenstra (1989)

En Kindervater y Lenstra (1989) se presenta gráficamente la diferencia entre estas cuatro alternativas. Ver Figura 14.

En algún caso, entendemos que en las estrategias de exploración diferentes, se emplean en cada uno de los procesos diferentes parámetros, mientras que en la división del trabajo entre SIMD y MIMD nos encontramos con funciones diferentes en cada uno de los procesos.

En caso de elegir una estrategia SIMD, el empleo de técnicas de búsqueda local, para que cada proceso explore una parte diferente del espacio, implica la división del espacio. Algunos autores (Roucairol, 1996) advierten de las dificultades que conlleva la división del espacio de soluciones dentro de los problemas de optimización combinatoria, ya que debe generarse alguna métrica para distinguir unas partes de otras en el espacio de soluciones.

Otra forma de repartir diferentes soluciones entre procesos se muestra en Jelasity *et al.* (2001), para problemas de optimización. Estos autores dividen el espacio de soluciones en clústeres, especies o poblaciones no superpuestas que son explorados por procesos diferentes según métodos de escalada (*hill climbing*). A medida que avanza la búsqueda, estos agrupamientos de soluciones se pueden reorganizar intercambiando algunas soluciones para evitar el estancamiento en zonas del espacio de búsqueda.

Para el problema del viajante (TSP), Shi *et al.* (1999) proponen un método que llaman Particiones Anidadas (*Nested Partitions*) para obtener soluciones bastante cercanas al óptimo, alcanzándolo con el suficiente tiempo de computación. Para conseguirlo, dividen el espacio de soluciones en conjuntos del mismo número de soluciones. Para alcanzar la mejor calidad de soluciones, es necesario que el método de partición del espacio que se emplee deje agrupadas en la misma región las soluciones de mayor calidad. Para ello, en cada iteración del algoritmo se identifica una región del espacio como la más prometedora y se considera otra región al resto del espacio.

Liu y Tseng (2000) presentan tres algoritmos para la división del espacio de soluciones cuando se afrontan problemas de minimización sin restricciones mediante procesamiento en paralelo.

Muy relacionada con la dificultad de división del espacio de soluciones, debe tenerse en cuenta y corregirse la posibilidad de exploración repetitiva de algunas zonas del espacio de soluciones por la superposición de las rutas de búsqueda. Esta tarea resulta compleja sobre todo en el caso de procesos independientes, donde las estrategias de exploración deberán diseñarse de forma más cuidadosa.

Cung *et al.* (2001) y Crainic *et al.* (2002) distinguen los siguientes modelos, según el papel que juegan los procesos:

- Modelo distribuido, simétrico, entre iguales o *peer-to-peer*, en el que todos los procesos son equivalentes.
- Modelo maestro-esclavo, en el que uno de los procesos coordina el funcionamiento de los demás.

Algunos autores, (ver Sittig *et al.,* 1991 o Talbi *et al.*, 1998), hacen una precisión a esta notación y hablan de procesos jefe y trabajadores (*master-worker*) cuando el proceso principal realiza algún procesamiento adicional al de coordinar a los procesos esclavos. También en Vazquez *et al.* (2002) se emplea una notación similar: *manager-worker*. Esta será la nomenclatura empleada en este trabajo de investigación.

2.6.3 Comunicación entre procesos

Cuando existen secciones de código que se ejecutan en paralelo, es factible que los diferentes procesos o hilos no trabajen de forma independiente, sino que intercambien información. Esta información puede ser, por ejemplo, el valor de la función objetivo alcanzado por algunos caminos o las soluciones que ya han sido exploradas, de forma que no se repitan pasos de exploración.

Según el mecanismo que se emplee para coordinar el momento en que se producen los intercambios de información entre procesos, se puede hablar de paralelismo síncrono o asíncrono. En el paralelismo síncrono los procesos realizan los intercambios a intervalos de tiempo prefijados. En el paralelismo asíncrono los intercambios de información entre procesos se producen cuando tienen lugar ciertos eventos. El paralelismo síncrono es más adecuado para arquitecturas homogéneas y procesos igua-

les. En caso contrario, es más interesante la comprobación de condiciones para el intercambio de datos.

La sincronización requiere que algunos procesos esperen a que otros terminen sus tareas. Estos tiempos de espera implican una disminución en la eficiencia del algoritmo cuando aumenta el número de procesos.

2.6.4 Granularidad

Atendiendo a la relación entre el tiempo de computación y el tiempo de comunicación, se distinguen entre algoritmos de grano fino – *fine grained* – o de grano grueso – *coarse grained* (Tosic, 2004). En los de grano fino, el tiempo de computación es del orden del tiempo de comunicación, mientras que en los de grano grueso tiene mayor importancia el tiempo de computación.

En el caso de la implementación en paralelo de un algoritmo de búsqueda local, una mayor comunicación entre procesos puede implicar mayor conocimiento sobre la exploración del espacio de soluciones y mayores posibilidades de evitar explorar zonas del espacio que ya han sido exploradas por otros procesos.

En general, los algoritmos de grano grueso son adecuados para sistemas multiordenador con arquitectura de memoria distribuida, mientras que los algoritmos de grano fino pueden aprovechar las ventajas de los sistemas multiprocesador de memoria compartida (Lilja, 1994).

Una clasificación que no se encuadra en ninguno de los apartados anteriores es la de Vazquez *et al.* (2002). Estos autores distinguen los algoritmos paralelos según se hayan desarrollado a partir de un algoritmo secuencial o por el contrario, si han sido desarrollados desde un principio como algoritmos paralelos. También distingue entre algoritmos que separan la ejecución del código y algoritmos que ejecutan el mismo código sobre diferentes partes del dominio, en lo que se conoce como descomposición del dominio.

2.6.5 Revisión de tipos de implementación en paralelo

En la Tabla 3 se resumen los principales tipos de algoritmos paralelos revisados en los apartados anteriores. Hemos considerado dentro de la misma clasificación los

algoritmos paralelos en los que se distingue el modelo de la implementación y el control de la exploración.

Criterio	**Tipos**	**Subtipos**
Estructura del programa	División del trabajo	SIMD/SPMD – misma estrategia de exploración – SPSS, MPSS
		MIMD – diferentes estrategias de exploración – SPMS, MPMS
	Punto de partida de la exploración	Único – SPSS, SPMS
		Diferentes – MPSS, MPMS
	Estrategia de exploración del espacio de soluciones	Pasada única o *single-walk*
		Pasada múltiple o *multiple-walk*
Comunicación	Papel de los procesos	Maestro-esclavo o *1*-C
		Simétrico, entre iguales o *peer-to-peer* o *p*-C
	Colaboración	Colaboración (tipo C o tipo KB)
		Independientes
	Comunicación entre procesos	Síncronos
		Asíncronos
	Granularidad	Grano fino
		Grano grueso

Tabla 3: Tipologías de algoritmos paralelos

En algún caso, por ejemplo con la división del trabajo, se han reunido conceptos similares designados por diferentes autores con nomenclaturas distintas. Así, se consideran dentro del mismo grupo aplicaciones SIMD, SPMD, SPMS y MPSS. De la misma forma, existen tipos de implementación presentados por algunos autores que pueden englobarse en más de un grupo, como SPMS.

La mayoría de las divisiones que se han hecho se encuentran combinadas en los algoritmos paralelos. Por ejemplo, un algoritmo paralelo de búsqueda puede implementarse en un sistema con memoria distribuida, emplear un modelo maestro-esclavo, partir de un solo punto, emplear estrategias de exploración diferentes pero las mismas funciones (SIMD) para cada uno de los procesos esclavos, empleando

más esfuerzo en la computación que en la comunicación (grano grueso) y que los procesos intercambien información a instantes dados (síncronos).

2.6.6 Procedimientos para el desarrollo de aplicaciones paralelas

Se presentan a continuación algunos procedimientos o metodologías de desarrollo propuestos por diferentes autores que facilitan la programación de aplicaciones en paralelo, haciendo especial énfasis en las diferentes propuestas para conseguir equilibrar la carga de trabajo de los procesadores.

- DPAC (*Distributed and Parallel Computing Framework*), de Raghavan y Waghmare (2002), hace uso de las posibilidades de interconexión que brinda Internet, para poner a trabajar en común un número indefinido de ordenadores. Basado en Java, su mayor interés es la tolerancia a fallos, debido al uso de una red menos fiable que una red de área local para conectar los nodos del sistema. Según sus autores, en la aplicación a la resolución del problema del viajante mediante ramificación y acotación, consigue aceleraciones superiores a la lineal bajo diferentes configuraciones físicas.
- SAMBA (*Single Application, Multiple load BAlancing*), de Plastino *et al.* (2003), para arquitecturas SIMD, hace hincapié en el equilibrio de la carga de trabajo soportada por cada procesador en la depuración del algoritmo secuencial. Emplea varias clases en el diseño del algoritmo: la clase *Mediator* es la encargada de crear tareas y gestionar recursos, mientras que la clase *Repository* mantiene un *pool* de tareas pendientes, representadas por los objetos de la clase *Task*. La clase *LoadBalancer* es la que se encarga de equilibrar el trabajo entre los procesadores. Para ello, incluye una clase *TransferChannel* que permite el intercambio de tareas, pasando éstas de los procesadores más ocupados a los procesadores ociosos.

El problema de la comunicación entre procesos ha sido estudiado para encontrar la mejor forma de resolver los procesos de intercambio de información uno a varios y varios a varios (Fraigniaud y Vial, 1999) de forma que los intercambios entre distintos grupos de procesos no interfieran con otros y se equilibre la carga de trabajo entre procesadores. Para el estudio del comportamiento de los distintos procesos en

el reparto de la carga, pueden emplearse herramientas como Paragraph, descrita por Blanco *et al.* (1993), con la que puede visualizarse el funcionamiento de los procesos que componen una aplicación paralela en un entorno gráfico.

Algunos autores proponen algoritmos con los que descomponer los cálculos que se deben realizar en varios subproblemas, (Gondzio *et al.,* 2001), de forma que su implementación pueda llevarse a cabo de forma semi-automática.

Otra de las posibilidades es la división automática de la ejecución entre los procesadores disponibles enviando datos redundantes en un pequeño porcentaje. Es lo que Hascoët (2001) llama superposición (*overlapping*), comunicando datos sólo cuando han quedado obsoletos. La aplicación típica de este método es en sistemas SPMD, en los que es complicado optimizar los instantes en los que debe producirse la comunicación. Algunos de sus experimentos muestran una aceleración superlineal empleando este método.

La división del espacio de soluciones es una de las formas más simples en las que puede descomponer la ejecución de algoritmos para su posterior implementación en paralelo. Roli y Blum (2001) lo han aplicado a la resolución mediante búsqueda local y consiste en dejar sin alterar una parte de la solución mientras que se varía el resto. Sus conclusiones apuntan a que esta división del espacio lleva a una mejora de las prestaciones del algoritmo de búsqueda, que es mayor cuanto mayor es la subdivisión. Esta mejora tiene un punto a partir del cual, subdividir aún más el espacio empeora las prestaciones. Este nivel óptimo de paralelismo está relacionado con el balance necesario en los algoritmos de búsqueda entre intensificación y diversificación.

El equilibrio de la carga de trabajo entre los distintos procesos está muy relacionado con la división del espacio de soluciones. Una adecuada división del espacio de soluciones junto con un reparto de tareas óptimo entre los procesos llevará a un aprovechamiento máximo de la capacidad de proceso disponible. El reparto de tareas entre procesos puede hacerse de forma dinámica o bien a priori. En el primer caso pueden aprovecharse las facilidades que ofrece MPI para iniciar nuevos procesos, detener otros o migrar alguno de un procesador a otro.

Algunos autores proponen que el reparto de la carga se debe realizar por la propia aplicación, sin necesidad de conocer a priori el comportamiento de los distintos procesos. Feschet *et al.* (1998) proponen implementar librerías que faciliten la tarea de equilibrar la carga de trabajo entre procesadores empleando una asignación dinámica de procesos a procesadores. Con esto se consigue reducir el tiempo de desarrollo de estas aplicaciones en paralelo y mejorar la portabilidad de las mismas al haberse desarrollado en forma de librerías de programación.

2.7 Referencias sobre programación en paralelo

Las tablas siguientes son un resumen de las referencias empleadas en este capítulo sobre técnicas de programación en paralelo.

Conceptos básicos	**Comentarios**
Hughes y Hughes (2003)	Conceptos sobre programación paralela y distribuida

Tabla 4: Referencia sobre conceptos básicos en algoritmos paralelos

Librerías de paso de mensajes	
MPI	
Gropp *et al.* (1999)	
Sankaran *et al.* (2003)	Mejoras en la fiabilidad de LAM
MPICH	
Gropp y Lusk (1992)	Implementación inicial de MPICH
Gropp y Smith (1993)	Tres versiones de MPI
LINDA	
Martins *et al.* (2002)	Revisión de herramientas de programación en paralelo
Carriero *et al.* (1994)	Descripción general de LINDA
Mejoras en la comunicación entre procesos	
Fraigniaud y Vial (1999)	Revisión de metaheurísticas en paralelo
Liu *et al.* (2004)	Redes tipo *infiniBand*

Tabla 5: Resumen de referencias sobre librerías de paso de mensajes entre procesos

Entornos de desarrollo de aplicaciones paralelas	
Acacio *et al.* (2002)	Entorno gráfico
Stankovic y Zhang (1999)	Entorno orientado a objetos

Tabla 6: Resumen de referencias sobre entornos de desarrollo

Revisión de procedimientos de paralelización	
Procedimientos de una o varias pasadas	
Verhoeven y Aarts (1995)	Clasificación de algoritmos en pasada única o múltiples pasadas
Resende *et al.* (1999)	Revisión de implementaciones de meta-heurísticas en paralelo
Equilibrado de carga entre procesadores	
Plastino *et al.* (2003)	Metodología para el equilibrado de la carga en aplicaciones SPMD
Feschet *et al.* (1998)	ParList, un sistema para equilibrado de la carga entre procesadores mediante estructuras de datos
Modelos de comunicación	
Cung *et al.* (2001)	Comenta la mejora en velocidad y robustez gracias al algoritmo paralelo
Crainic *et al.* (2002)	Modelos de procesos equivalentes y maestro-esclavo entre otros
Otras clasificaciones	
Barr y Hickman (1993)	Conceptos generales sobre tipos de algoritmos paralelos y evaluación de sus prestaciones
Brawer (1986)	Modelos abstractos de sistemas paralelos
Gibbons *et al.* (2003)	Transformación de algoritmos de grano fino en algoritmos de grano grueso
Vazquez *et al.* (2002)	Clasificación según el proceso de desarrollo del algoritmo
Raghavan y Waghmare (2002)	Sistema paralelo a través de Internet
Le Bouthillier y Crainic (2005)	Procedimientos de exploración diferentes para cada proceso
Crainic y Toulouse (2002)	Tipos de hilos de ejecución en paralelo
Flynn (1966)	Arquitecturas SIMD y MIMD

Tabla 7: Resumen de referencias sobre procedimientos de paralelización

División del trabajo	
División del espacio de soluciones	
Roli y Blum (2001)	Algoritmo de búsqueda local
Jelasity *et al.* (2001)	Métodos de escalada. Los conjuntos de soluciones varían en tiempo de ejecución
Roucairol (1996)	Necesidad de una métrica para la división
Shi *et al.* (1999)	Particiones anidadas
Liu y Tseng (2000)	Aplicación al entrenamiento de redes neuronales
Vazquez *et al.* (2002)	Bloques de soluciones que pueden pasar de un proceso a otro
División de la aplicación en hilos	
Le Bouthillier y Crainic (2005)	Puntos de partida diferentes
Descomposición de tareas	
Gondzio *et al.* (2001)	Descomposición automática de las tareas
Hascoët (2001)	Reenvío o superposición de datos cuando quedan obsoletos

Tabla 8: Resumen de referencias sobre división del trabajo entre procesos

Evaluación de prestaciones	
Barr y Hickman (1993)	Conceptos generales sobre tipos de algoritmos paralelos y evaluación de sus prestaciones
Calzarossa *et al.* (2004)	Metodología para medición de prestaciones
Martins *et al.* (2002)	Revisión de herramientas de programación en paralelo
Blanco *et al.* (1993)	Herramienta gráfica
Comparación entre prestaciones de varios sistemas de paso de mensajes	
Markus *et al.* (1996)	Comparación de las prestaciones de MPICH, CHIMP y LAM
Van Voorst y Seidel (2000)	Comparación de las prestaciones de librerías de paso de mensajes

Tabla 9: Resumen de referencias sobre evaluación de las prestaciones de algoritmos paralelos

2.8 Conclusiones

En este capítulo se ha presentado una introducción a la programación en paralelo revisando, para ello, la literatura existente en cuanto al desarrollo de aplicaciones. Se ha hecho hincapié en aquellos trabajos orientados a la implementación de metaheurísticas, ya que este es el enfoque que se adoptará en el trabajo.

Se ha comentado el concepto de programación o computación concurrente. La primera distinción en la forma de repartir la ejecución de un programa, está en si la ejecución del programa se reparte entre los procesadores de un ordenador o entre varios ordenadores. En el primer caso se habla de computación paralela, mientras que en el segundo se habla de computación distribuida. Se ha comentado la diferencia entre hilos y procesos: el proceso dispone de recursos independientes, mientras que los hilos no. La sincronización ayuda a controlar las modificaciones que realizan hilos o procesos diferentes sobre las mismas variables pero incrementa el tiempo de espera entre procesos.

En este capítulo se han revisado medidas para valorar el aprovechamiento tanto el esfuerzo de diseño del programa en paralelo como la inversión en equipos informáticos y en su configuración. Para ello, se emplean en este trabajo entre otras, la aceleración y la eficiencia del algoritmo paralelo. Ambas miden de forma similar si la potencia de cálculo disponible está siendo aprovechada.

Se han descrito las ventajas e inconvenientes de la computación paralela y se han revisado diferentes arquitecturas informáticas para implementar aplicaciones en paralelo. Tratándose de un trabajo de investigación orientado a herramientas adoptables por empresas pequeñas o medianas, la mejor opción por su relación coste/prestaciones es la de un clúster de ordenadores, ya que el resto de opciones resultan más costosas o de una difusión escasa.

De entre las alternativas software para configurar el sistema paralelo se ha optado por la distribución Debian de Linux, ya que es la que permitió poner en funcionamiento el sistema en un espacio más corto de tiempo, teniendo un desarrollo actual. Las librerías de paso de mensajes MPI en su versión LAM se han seleccionado para este trabajo.

Como principales aportaciones de este capítulo se pueden destacar las siguientes:

- Se han clasificado y resumido las principales ventajas que se citan en la literatura de los algoritmos paralelos así como las principales objeciones al empleo de sistemas paralelos que se han documentado desde los primeros momentos de la computación paralela.
- Se han extraído y presentado las posibilidades que existen en cuanto a hardware y en cuanto a software para implementar aplicaciones en paralelo, siempre teniendo en cuenta su aplicabilidad tanto a la resolución del problema de secuenciación como a empresas de nuestro entorno.
- Se han clasificado las políticas que suelen emplearse en la literatura para el reparto de las tareas que debe realizar un programa paralelo. Se han agrupado políticas que suelen encontrarse con distinta nomenclatura pero comportamientos similares en cuanto a división tanto de la ejecución del programa como del conjunto de datos sobre los que trabaja dicho programa.

3 Computación paralela aplicada a problemas de secuenciación en flujo uniforme

3.1 Introducción

Una vez vistas las distintas opciones que ofrece la implementación de aplicaciones informáticas en paralelo, en este capítulo se pasa a describir sus aplicaciones para la búsqueda de soluciones de uno de los problemas de optimización combinatoria más intensamente investigados como es el de la secuenciación de trabajos en entornos de flujo uniforme.

Se han revisado las referencias en la literatura en una doble vertiente: por un lado, referencias sobre los procedimientos más usuales para la implementación en paralelo de métodos para resolver problemas de optimización. Por otro lado, referencias que describan la aplicación exitosa de estos métodos a la resolución del problema $F_m/prmu/C_{max}$.

Así, en el apartado 2 se describe en detalle el problema objeto de este trabajo de investigación, comentando sus características principales. En el apartado 3 se ha hecho una revisión de los métodos más conocidos empleados en la resolución de problemas de optimización, centrándonos en los aplicados al problema de secuenciación en flujo uniforme. En el apartado 4 se resumen las referencias recogidas sobre aplicaciones en paralelo a la resolución del problema objeto de estudio, clasificándolas según las variantes comentadas en el capítulo 2 de este documento.

Finalmente, el apartado 5 contiene las conclusiones y un resumen de las aportaciones más significativas de este capítulo.

3.2 La secuenciación de trabajos en entornos de flujo uniforme

En numerosas plantas de producción o montaje se emplean configuraciones o disposiciones de las máquinas en las que conjuntos de trabajos, tareas u operaciones deben realizarse en una serie de máquinas o puestos de trabajo. Cuando los trabajos deben realizarse todos en el mismo orden en cada una de las máquinas, el entorno de producción se conoce como entorno de flujo uniforme, talleres de flujo o entornos tipo *flowshop*.

La notación de este problema, siguiendo la empleada por Graham *et al.* (1979), es $F_m/prmu/C_{max}$. Según esta notación, los problemas de secuenciación se describen mediante tres elementos, $\alpha|\beta|\gamma$. El primero de ellos describe el tipo de problema de secuenciación. En nuestro caso, es un entorno de flujo uniforme, F, con m máquinas. El segundo elemento describe características especiales del tipo de problema. No se permite el adelantamiento de trabajos, por lo que nos encontramos con un flujo de permutación, $prmu$. El tercer elemento representa la función objetivo. Tratamos de minimizar el tiempo en el que termina el último trabajo desde que empezó el procesamiento del primero. Este es el tiempo total de ejecución de la secuencia de trabajos o *makespan*, que se representa por C_{max}.

La forma de calcular el *makespan* se basa en ir sumando, de forma recursiva, los tiempos de terminación de cada uno de los trabajos en cada una de las máquinas. Así, para la secuencia de trabajos $j_1, \ldots, j_n$, si el tiempo de procesamiento del trabajo j_k en la máquina i es p_{i,j_k}, entonces C_{i,j_1}, el tiempo de terminación del primer trabajo de la secuencia j_1 en la máquina i es la suma de los tiempos de proceso del primer trabajo en todas las máquinas hasta la máquina i:

$$C_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1} \qquad \forall i = 1,\ldots,m \tag{8}$$

El tiempo de terminación del k-ésimo trabajo de la secuencia j_k en la primera máquina es la suma de los tiempos de proceso de todos los trabajos hasta el j en la primera máquina:

$$C_{1,j_k} = \sum_{l=1}^{k} p_{1,j_l} \quad \forall k = 1,\ldots,n \tag{9}$$

De este modo, el tiempo de terminación de un trabajo cualquiera j_k en una máquina cualquiera i es:

$$C_{i,j_k} = \max\left(C_{i-1,j_k}, C_{i,j_{k-1}}\right) + p_{i,j_k} \quad \forall i = 1,\ldots,m; \forall k = 1,\ldots,n \tag{10}$$

Otra forma de calcular el valor del *makespan* es (ver e.g. Pinedo, 2002), resolver el grafo dirigido que se construye asignando al peso de cada nodo el valor del tiempo de proceso de cada uno de los trabajos en cada una de las máquinas (ver Figura 15).

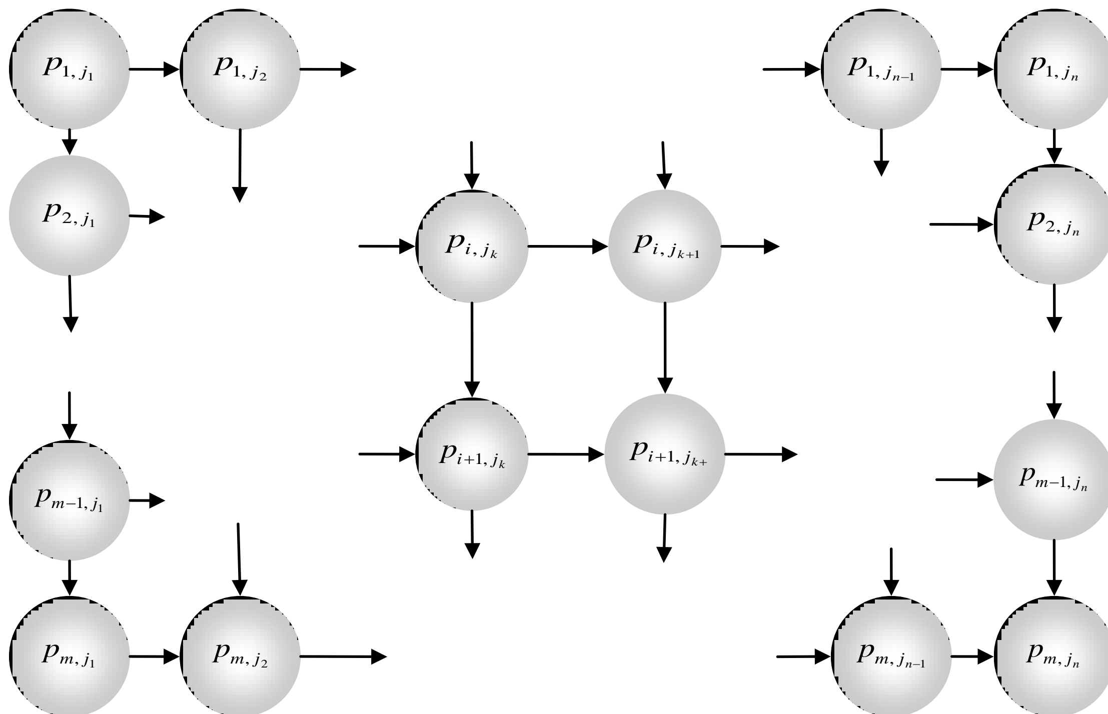


Figura 15: Grafo dirigido para el cálculo de makespan

Este problema es NP-completo para un número de máquinas mayor de 2 (Garey *et al.,* 1976), por lo que la mayor parte de las investigaciones se han centrado en la obtención de soluciones de buena calidad pero sin garantías de optimalidad. Cuando el número de máquinas es 2, la solución óptima se obtiene en tiempo polinomial mediante la regla de Johnson (1954).

Taillard (1990) puso de manifiesto algunas particularidades sobre la distribución de las soluciones del problema $F_m/prmu/C_{max}$, al menos en problemas de pequeña dimensión. Mediante una colección de 500 problemas generados con tiempos de proceso aleatorios entre 1 y 100 para 9 trabajos y 10 máquinas, obtuvo una serie de resultados sobre la distribución de los *makespan*. De acuerdo con estos resultados, menos del 0.02% de las soluciones obtenidas están en un entorno de 1% del óptimo, luego será difícil encontrar soluciones con este nivel de calidad. Por otro lado, el valor de *makespan* de una solución al problema generada de forma aleatoria está, de media, a un 20% del óptimo, por lo que se obtendrán gran cantidad de soluciones en una franja relativamente estrecha de valores de la función objetivo.

3.3 Métodos de resolución del problema objeto de estudio mediante computación paralela. Estado del arte

Se presentan a continuación los métodos de resolución que se han aplicado al problema objeto de estudio siguiendo la clasificación que aparece en Ladhari y Haouari (2005) y completada con la que propone Alba (2005). En primer lugar, se distinguen entre métodos exactos o completos y métodos aproximados. Dentro de éstos, se distinguen las heurísticas constructivas y las metaheurísticas. Se ha considerado más completa esta clasificación que otras que separan los métodos en inspirados o no en la naturaleza (p.ej. optimización mediante colonias de hormigas está inspirado en la naturaleza) o bien en métodos con o sin memoria (p. ej. la búsqueda tabú es un método con memoria)

Se han revisado con más detalle aquellos procedimientos de los que se encuentran referencias aplicadas al problema $F_m/prmu/C_{max}$ y sobre todo de las que se conocen aplicaciones en paralelo. La Figura 16 contiene un esquema de los métodos revisados en este capítulo.

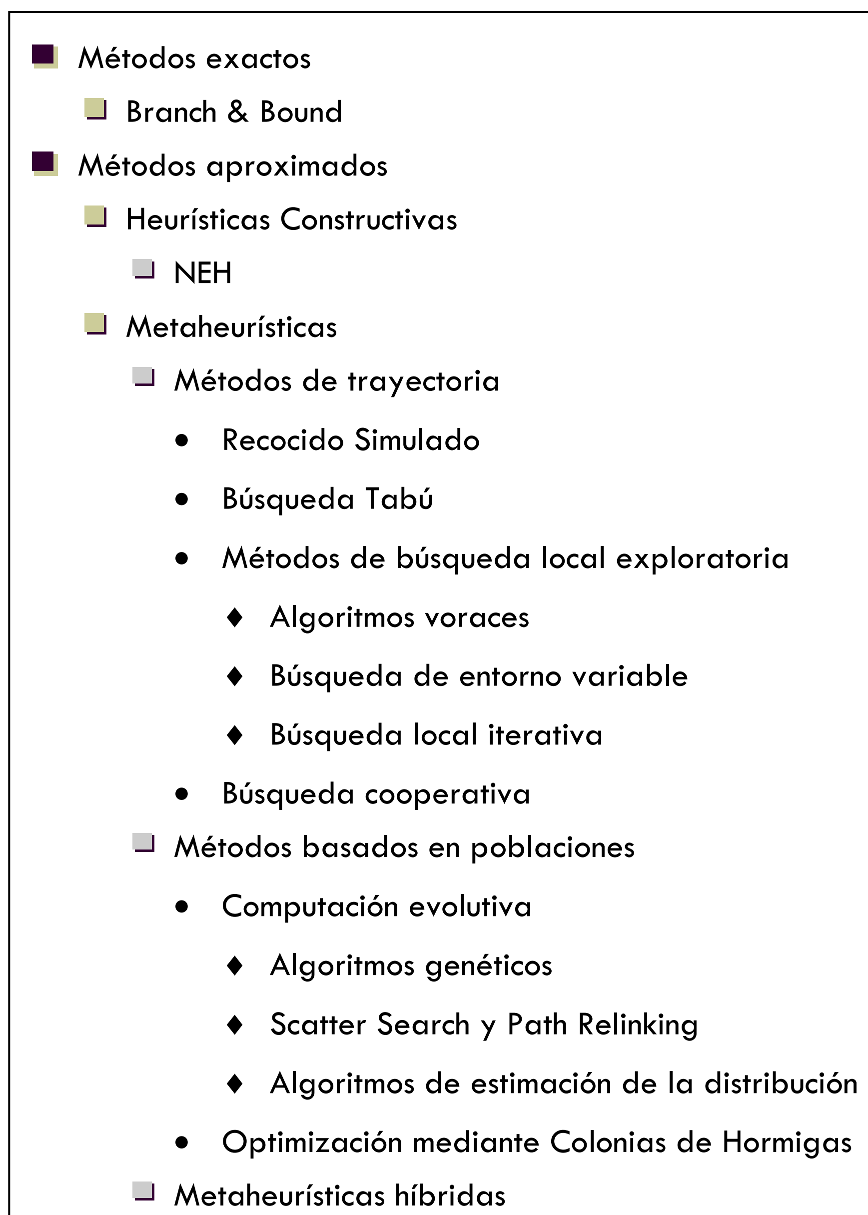


Figura 16: Métodos de resolución en paralelo aplicados al problema $F_m|prmu|C_{max}$

Además de la clasificación anterior, que se empleará en el resto del capítulo, en la Figura 17 se presenta otra clasificación de procedimientos para implementar meta-heurísticas descritos en Talbi (2002). La primera distinción se hace entre su implementación en ordenadores especialmente diseñados para la resolución de un problema concreto como la descrita en Abramson (1992) para el recocido simulado y la implementación en ordenadores de propósito general.

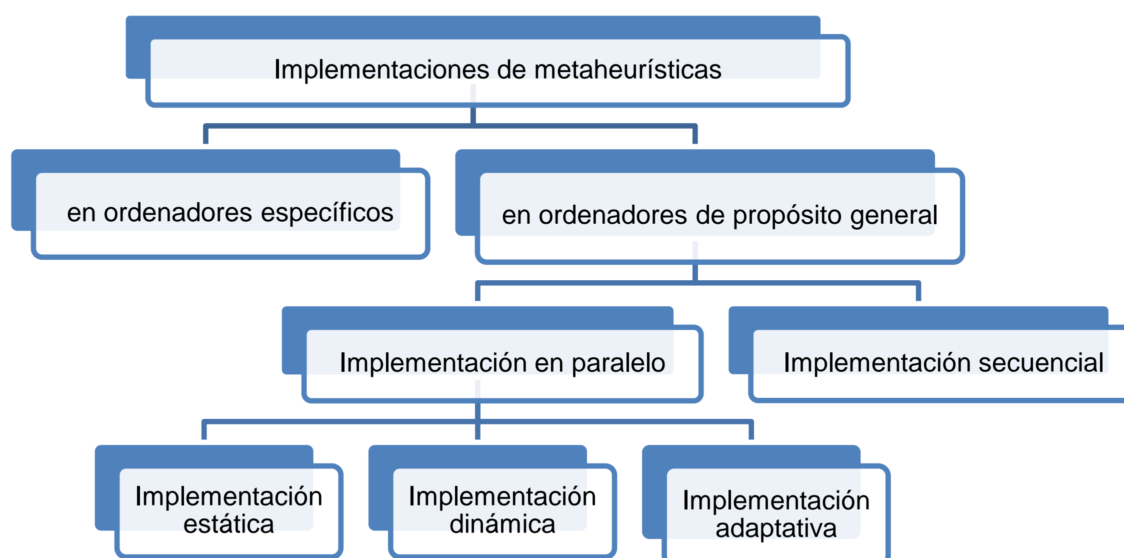


Figura 17: Tipos de implementación de metaheurísticas según Talbi (2002).

A su vez, sobre los sistemas paralelos, el mismo autor propone una segunda clasificación (ver Figura 18) similar a la presentada por otros autores como Flynn (1966), ya descrita en el capítulo 2 de esta tesis.

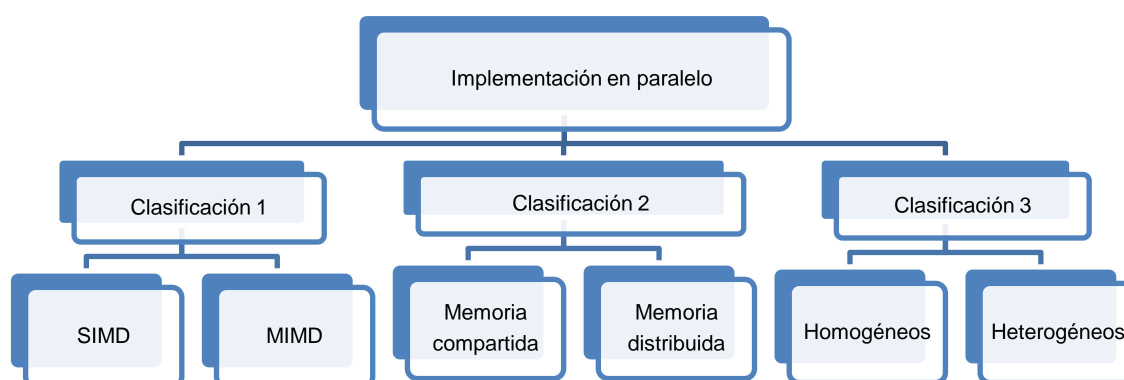


Figura 18: Clasificación de tipos de implementación en paralelo de metaheurísticas según Talbi.

3.3.1 Métodos Exactos: Branch&Bound

Los métodos exactos de resolución, como ramificación y acotación (o *branch & bound* – B&B, Land y Doig, 1960), son sólo aplicables al problema de estudio cuando el

número de máquinas es pequeño ya que la explosión combinatoria puede llevar a tiempos de resolución inaceptables cuando aumenta la dimensión del problema.

El método consiste en una búsqueda estructurada del espacio de soluciones mediante la cual se divide el mismo de forma iterativa (ramificación), resultando en problemas de cada vez menor dimensión. En problemas de minimización se calcula una cota inferior (acotación) para el coste de las soluciones de cada subconjunto. Además, se calcula una cota superior como el valor de la mejor solución admisible encontrada. Después de cada subdivisión, aquellos subconjuntos con una cota inferior que excede el valor de una solución admisible son excluidos de posteriores particiones (poda). De este modo, es posible excluir de la búsqueda conjuntos de soluciones sin llegar a explorarlas. Se continúa la subdivisión hasta que se encuentra una solución admisible cuyo coste no es mayor que la cota superior de cualquier subconjunto. El éxito de la técnica depende del número de soluciones examinadas antes de alcanzar una solución óptima (Gillet, 1976).

Habitualmente, se representa el problema en forma de árbol, siendo cada nodo del árbol un subproblema con menor dimensión que su antecesor, etiquetando los nodos con la cota inferior para todas las ramas o subproblemas que parten de ese nodo.

Las variantes de B&B consisten en diferentes formas de hacer la subdivisión del espacio o ramificación, diferentes procedimientos para acotar o encontrar una cota inferior del problema y diferentes criterios para descartar o podar conjuntos de soluciones.

En la Figura 19 se muestra un pseudocódigo del algoritmo B&B general.

Paso 1. Inicializar Z_0 a un valor positivo arbitrariamente grande. Tomar IOP como $(CP)_I$.

Paso 2. **Mientras** $LC \neq \{\emptyset\}$

a. **Para** $I = 1$ Hasta K

i. Elegir un criterio de relajación $(CPR)_I$ para $(CP)_I$.

ii. Encontrar una cota Z_I para $(CP)_I$.

iii. **Si** la solución del $(CPR)_I$ es una solución admisible de IOP

1. **Si** el valor de la solución $(CP)_I$ es menor que Z_0, asignar a Z_0 el valor de la solución de $(CP)_I$.

iv. **Si no**

1. **Si** $Z_I < Z_0$, $LC := LC \cup \{(CP)_I\}$

b. Eliminar un problema de la lista de candidatos para ramificar. Etiquetar el problema CP.

c. Ramificar en CP. Partir CP en K subproblemas nuevos $(CP)_I$, $I = 1, 2, ..., K$ e ir al paso 2.

Paso 3. **Si** se ha alcanzado una solución admisible, La mejor solución admisible hasta el momento es una solución óptima de IOP.

Paso 4. **Si no,** no existe solución admisible para IOP.

Paso 5. **Stop**.

Figura 19: Pseudocódigo general de Branch and Bound (Adaptado de Gillet, 1976)

La notación que se emplea es la siguiente:

IOP	*Initial Optimization Problem* (se supone minimizar). Es el problema original.
Z_0	Menor cota superior actual de la solución óptima
$F(A)$	Conjunto de soluciones admisibles del problema A
K	Número de subproblemas relajados en los que se divide el problema original
$(A_1, ..., A_k)$	Subproblemas de A tales que:
	a) Cada solución admisible de A es una solución admisible de exactamente un A_I

b) Una solución admisible de cualquier A_I es una solución admisible de A.

$(CP)_I$ Problema candidato actual que está siendo explorado

$(CPR)_I$ Problema candidato relajado actual que está siendo explorado

Z_I Cota para el problema $(CP)_I$

LC Lista de candidatos: Subproblemas activos candidatos a ser explorados.

Algunas notas sobre el pseudocódigo anterior:

Paso 2.a.i.: Generalmente, el problema original, $(CP)_I$, es imposible o muy difícil de resolver. Se genera un problema, $(CPR)_I$, a partir del anterior relajando alguna de las restricciones del problema original. Si $(CPR)_I$ no tiene soluciones admisibles, tampoco las tendrá el problema sin relajar $(CP)_I$. El valor mínimo de la solución de $(CP)_I$ no es menor que el de la solución de $(CPR)_I$.

Paso 2.a.ii.: Aplicar un algoritmo apropiado a $(CPR)_I$ para acotar todas las soluciones posibles de $(CP)_I$.

Paso 2.a.: Se completa cuando todos los subproblemas creados por la última rama han sido explorados (acotados y analizados). En cada valor de I se explora uno de los subproblemas creados en la última rama.

Paso 2.c.: Se emplean normalmente reglas de decisión del tipo "ramificar desde la menor cota", que se conoce también como ramificar en profundidad, o "ramificar desde la cota activa más reciente", también conocido como ramificar en horizontal, o bien una combinación de ambas reglas. La regla de decisión para ramificar desde la cota activa más reciente, nombrada a veces como *last-in-first-out* (LIFO), selecciona el último problema añadido a la lista de candidatos.

Una revisión bastante completa de B&B aplicado al problema de secuenciación de trabajos en flujo uniforme se encuentra en Ladhari y Haouari (2005). Para un número pequeño de máquinas, citan los procedimientos secuenciales propuestos por Ignall y Schrage (1965), Lomnicki (1965), Brown y Lomnicki (1966), McMahon y Burton

(1967), Ashour (1970), Lageweg *et al.* (1978), Potts (1980), Grabowski (1980), Carlier y Rebaï (1996) y Cheng *et al.* (1997). Para la resolución de pequeñas instancias del problema $F_m|prmu|C_{max}$, Tseng y Stafford (2008) emplearon programación lineal entera mixta para resolver problemas de hasta 10 máquinas y 15 trabajos.

En cuanto a versiones en paralelo del algoritmo B&B, existen varios artículos de revisión de la literatura. Roucairol (1996) se concentra en la implementación de métodos B&B en paralelo. En esta revisión, la autora comenta las dificultades de la implementación en paralelo de métodos B&B: reparto del trabajo entre procesadores, comunicación entre los mismos o exploración inútil de algunos de los nodos. También comenta que la implementación en paralelo de metaheurísticas se encuentra con dificultades similares. Por otro lado, distingue dos tipos de implementación en paralelo de algoritmos B&B: Distribuida o vertical y centralizada u horizontal. En el tipo distribuido, todos los procesos son iguales, exploran cada uno nodos diferentes del árbol y sólo intercambian soluciones admisibles. En el tipo centralizado, un proceso es el que se encarga de distribuir los nodos que van a ser explorados al resto de procesos que exploran uno o dos niveles antes de devolver el resultado al proceso central. Para evitar cuellos de botella en este proceso, se suele emplear más de un proceso central. También propone el desarrollo de librerías que faciliten la tarea de la implementación en paralelo de este tipo de algoritmos.

Otra revisión es la de Anstreicher (2003), centrada en el problema de asignación cuadrática (QAP) que, aparte de metaheurísticas, recoge referencias de aplicación de métodos B&B en paralelo: Brüngger *et al.* (1998) y Clausen y Perregaard (1997).

En la revisión de Crainic *et al.* (2006) se describen dos técnicas para implementar B&B en paralelo:

- Técnicas basadas en nodo, que buscan reducir el tiempo a base de realizar en paralelo operaciones a nivel de nodo, como el cálculo de cotas.
- Técnicas basadas en árbol, que buscan reducir el tiempo mediante la exploración en paralelo del árbol.

Para el problema de secuenciación en flujo uniforme se han desarrollado dos versiones en paralelo del algoritmo B&B en Okamoto *et al.* (1994). En Companys y Mateo (2007) se describe un algoritmo B&B secuencial que podría implementarse en paralelo empleando dos procesadores. Algunas de las características de ambos métodos se describen brevemente a continuación.

El trabajo de Okamoto *et al.* (1994) y también Okamoto *et al.* (1995) está basado en la versión de B&B secuencial de Ignall y Schrage (1965). Cuando en la ramificación se llega a problemas con 5 o menos trabajos se emplea el método de enumeración, por precisar este método de menos operaciones que B&B con 5 o menos trabajos. Para encontrar la cota inferior correspondiente a cada nodo, emplea la expresión (11):

$$b(\sigma) = \max_j \{q_j(\sigma) + \sum_{i \in \sigma'} p_{ij} + \min_{i \in \sigma'} \{p_{ij+1} + p_{ij+2} + \cdots + p_{im}\}\} \qquad (11)$$

Donde σ es la secuencia parcial de la que se calcula la cota inferior b, σ' son los trabajos que no están en σ y q_j es el último tiempo de terminación de todos los trabajos de la secuencia parcial σ en la máquina j. Para ramificar, emplea la política de explorar primero en profundidad.

Okamoto *et al.* (1994) y Okamoto *et al.* (1995) realizaron dos implementaciones de B&B en paralelo. Una en un sistema con memoria distribuida (nCUBE2, de 32 nodos) donde la arquitectura de comunicación entre procesos es maestro-esclavo, aunque los llama trabajadores en la descripción del algoritmo. Éste alcanza aceleración superlineal en problemas de 5 máquinas y hasta 14 trabajos. La segunda es una versión del algoritmo paralelo para un equipo con memoria compartida (LUNA88k2, 4 CPUs) empleando procesos iguales.

En la versión maestro-esclavo, uno de los nodos ejecuta un proceso maestro que se encarga de enviar los problemas a los nodos trabajadores, de recoger y administrar las mejores soluciones encontradas y de la finalización del algoritmo. Los procesos trabajadores ejecutan el algoritmo de B&B y devuelven la mejor solución encontrada al proceso maestro. Los autores obtuvieron en promedio valores de acelera-

ción de 26 empleando en todo momento B&B y 20.5 cuando se empleó enumeración al llegar a 5 trabajos. La versión para memoria compartida, aun con los mismos promedios de aceleración, registró más experimentos con aceleración sublineal. La experimentación se llevó a cabo para problemas generados de forma aleatoria de 5 a 14 trabajos y de 5 a 14 máquinas. Los tiempos de proceso se generaron de forma aleatoria.

El trabajo de Companys y Mateo (2007) propone un algoritmo secuencial con características que facilitarían su implementación en paralelo. Esta característica es el empleo de dos secuencias, una inversa de la otra para calcular sus cotas. Este cálculo se podría realizar en dos procesadores diferentes. Emplea como batería de pruebas problemas generados de forma aleatoria de hasta 5 máquinas y 15 trabajos y también la batería de Taillard (1993) para comparar el comportamiento de las soluciones del problema de secuenciación ($F_m/prmu/C_{max}$) con las del problema sin almacenes entre máquinas ($F_m|block|C_{max}$). La versión de B&B empleada es el algoritmo doble *branch and bound*, de Lomnicki Pendular o LOMPEN, de Companys (1999). A su vez, esta versión está basada en la versión de B&B de Lomnicki (1965). El cálculo de las cotas inferiores de las secuencias parciales, $b(\sigma)$, llamada la *cota de dos máquinas*, consiste en calcular las cotas inferiores mediante tres términos. Para dos máquinas j y k tales que $1 \leq j \leq k \leq m$, se suman:

(a) el tiempo $f_{j,r}(\sigma)$ en el que se terminan en la máquina j los trabajos de σ.

(b) Una cota para el tiempo empleado por el último trabajo de $\bar{J}$ después de salir de la máquina k.

(c) Una cota para el *makespan* del subproblema definido con todas las operaciones de trabajos de $\bar{J}$ en las máquinas de la j a la k incluidas.

La cota inferior $b(\sigma)$ se elige como el máximo del valor calculado de las cotas de dos máquinas. Para $k=j$, el término c) es $\sum_{i \in \bar{J}} p_{j,i}$, mientras que para $k=j+1$ se puede obtener con la regla de Johnson (1954). Para $k>j+1$, se puede obtener con una relajación del subproblema. Estos autores emplean un problema $F_2|l_i, prmu|C_{max}$ como problema relajado.

En el algoritmo LOMPEN se aprovecha la llamada propiedad de reversibilidad: Si I es una instancia del problema $F_m/prmu/C_{max}$, la instancia inversa I' se construye tomando como tiempos de procesamiento $p'_{j,i}=p_{m-j+1,i}$ con $j=1, ..., m$; $i=1, ...,n$ la secuencia S de la instancia I tiene el mismo valor de *makespan* que la secuencia inversa S' de la instancia I'. En algunos casos, puede ser más fácil calcular el valor de *makespan* o una cota para la secuencia inversa.

Referencia	Prob	Notas
Roucairol (1987)	QAP	Aceleración casi lineal hasta 4 procesadores. Memoria compartida.
Pardalos y Crouse (1989)	QAP	4 procesadores. Instancias hasta n=15
Laursen (1993)	QAP	Dos implementaciones en paralelo de B&B: La primera distribuye los subproblemas entre procesadores y les impide la comunicación. La segunda usa distribución dinámica y sincroniza los procesos.
Mautor y Roucairol (1994)	QAP	Emplea características del problema para simplificar el algoritmo B&B en paralelo y obtener mejores resultados que otras versiones de B&B que hacen hincapié en encontrar buenas cotas inferiores
Mans *et al.* (1995)	QAP	Aceleración lineal. Memoria compartida.
Clausen y Perregaard (1997)	QAP	Implementación en un sistema paralelo con 16 procesadores.
Brüngger *et al.* (1997), Brüngger *et al.* (1998)	QAP	Librería ZRAM de búsqueda local en paralelo
Clausen *et al.* (1998)	QAP	Estudio del empleo de cotas inferiores basadas en descomposición en todos los nodos del árbol de B&B.
Moe (2004), Moe y Sørevik (2005)	QAP	Arquitectura distribuida (GReat International Branch-and-Bound search – GRIBB)
Roucairol (1989)	TSP	Revisión sobre B&B en paralelo. Comenta las anomalías que presenta B&B cuando se implementa en paralelo. La eficiencia depende de la función de acotación, las estructuras de datos y la estrategia de búsqueda.
Pardalos y Rodgers (1990)	0-1 KP	Multiprocesador con memoria distribuida. Divide el problema en tantos subproblemas como procesadores.
Clausen y Larsson Träff (1991)	GPP	Prueba dos posibilidades de cálculo de cotas. Sistema con 32 procesadores.
Ikegami *et al.* (1993)	ILP	Implementación en un sistema nCUBE con memoria distribuida

Tabla 10: Algoritmos tipo B&B en paralelo aplicados a problemas de permutación (ILP – *Integer Linear Programming*, 0-1 KP – *Knapsack Problem*)

La posible implementación en paralelo que apuntan los autores se basa en que la exploración de la instancia I y la inversa I' puede hacerse con dos procesos B&B intercambiando datos entre ellos, pero no realizan esta implementación.

En la Tabla 10 se mencionan brevemente otras referencias encontradas sobre implementaciones en paralelo de B&B aplicadas fundamentalmente al problema de asignación cuadrática (QAP).

3.3.2 Métodos aproximados

En este apartado se revisan métodos de aproximación de soluciones clasificados en heurísticas constructivas y metaheurísticas.

3.3.2.1 Heurísticas constructivas

Autores como Turner y Booth (1987), Taillard (1990) o más recientemente Kalczynski y Kamburowski (2007) han demostrado que, para el problema de la secuenciación de trabajos en flujo uniforme con el objetivo de minimizar el *makespan*, la heurística constructiva que ofrece un mejor comportamiento es la NEH, de Nawaz *et al.* (1983). Este resultado se ha confirmado en trabajos como el de Framiñán *et al.* (2003), donde también se puso de manifiesto las mejores prestaciones de la heurística NEH en cuanto a tiempo de computación para obtener soluciones de calidad en cuanto a *makespan* y en cuanto a *flowtime* y en el trabajo de Ruiz y Maroto (2005) tanto en calidad de soluciones como en tiempo de ejecución.

También en Rad *et al.* (2006) se proponen cinco heurísticas constructivas que mejoran las prestaciones de la NEH, al aplicarlas al problema $F_m/prmu/C_{max}$ en cuanto a calidad de soluciones, con un aumento despreciable en el tiempo de computación. Éstas tienen además la capacidad de mejorar varias metaheurísticas que emplean la NEH como punto de partida. Respecto a la la búsqueda local iterada (*Iterated Local Search* – ILS) de Stützle (1998a), la FRB5 de Rad et al. (2006) ofrece en promedio peor calidad de soluciones, 1.44 frente a 1.00 de ARPD pero mejor tiempo de ejecución, 4.88 s frente a los 9.95 s de promedio para toda la batería de Taillard (1993) que necesitó la ILS. Para medir la calidad del conjunto de soluciones obtenidas, se suele emplear el promedio del porcentaje de desviación con respecto a la mejor solución conocida o ARPD (*Average Relative Percentage Deviation*).

En Oliveira y Pardalos (2004) se combinan en paralelo heurísticas constructivas para resolver problemas de asignación multidimensional. El algoritmo que se implementa en paralelo es una heurística voraz no determinista (no se selecciona la mejor opción para completar la solución, sino una al azar de entre las mejores opciones). La primera versión es una implementación maestro-esclavo. El proceso maestro hace una asignación inicial y envía una posibilidad de asignación a cada uno de los procesos esclavos. El proceso maestro recoge los cálculos e incorpora como siguiente componente a la solución el de menor coste. La segunda versión de implementación en paralelo sigue un esquema jerárquico, en el que cada proceso funciona como el proceso maestro para sus procesos hijos. Una vez construida así una buena solución, los autores proponen un algoritmo paralelo de búsqueda local para alcanzar al menos un óptimo local. El proceso maestro es el que decide qué partes de la solución permanecen fijas para cada proceso esclavo y recibe el resultado de la mejora posible obtenida por cada proceso esclavo intercambiando posiciones en la solución. En general, obtienen mejoras en el tiempo de ejecución con respecto al algoritmo secuencial excepto cuando el número de procesos es mayor de 2, caso en el que una mejora en la calidad de las soluciones pero un empeoramiento del tiempo de ejecución, debido al aumento de la comunicación y la sincronización entre procesos.

3.3.2.2 Metaheurísticas

Ruiz y Maroto (2005) realizan una revisión de metaheurísticas empleadas con éxito en la aplicación al problema $F_m/prmu/C_{max}$, llegando a la conclusión de que, aplicados a la batería de pruebas de Taillard (1993), los mejores resultados se obtienen con la ILS de Stützle (1998a) y con el algoritmo genético propuesto por Reeves (1995).

A continuación, se comentan las metaheurísticas que se muestran de manera esquemática en la Figura 16 junto con los algoritmos tanto secuenciales como paralelos que se han empleado para aplicarlas a la búsqueda de soluciones del problema objeto de estudio.

3.3.2.2.1 Métodos de trayectoria

3.3.2.2.1.1 Recocido simulado

En la Figura 20, se muestra un pseudocódigo del algoritmo secuencial de recocido simulado, SA – *Simulated Annealing* (Kirpatrick *et al.,* 1983).

Paso 1. Sea $S_{best} := S$

Paso 2. Para k=1, 2, ... , it

2.1. Seleccionar al azar $S_{neig} \in N(S)$

2.2. Si $F(S_{neig}) < F(S_{best})$

2.2.1. $S_{best} := S_{neig}$

2.3. Si $F(S_{neig}) < F(S)$

2.3.1. $S := S_{neig}$

2.3.2. Si no

2.3.2.1. Si $P(S, S_{neig}, T) > random([0, 1])$

2.3.2.1.1. $S := S_{neig}$

Paso 3. Modificar la temperatura T

Paso 4. Si no se cumple la condición de parada, volver al paso 1

Figura 20: Pseudocódigo de algoritmo secuencial de recocido simulado

Las variables empleadas en la Figura 20 son las siguientes:

S: Una solución admisible

$N(S)$: Vecindad o entorno de S

$F(S)$: Valor de la función objetivo en S

S_{best}: Mejor solución hasta el momento

S_{neig}: Una solución en $N(S)$.

T: Temperatura

I: Número de iteraciones para cada valor de temperatura T fijo.

$P(S, S_{neig}, T)$: Función de probabilidad de aceptación de soluciones, definida de forma que las soluciones que suponen gran pérdida en la función objetivo tienen

menos probabilidad de ser seleccionadas, mientras que aquellas que suponen un empeoramiento menor en la función objetivo tienen mayor probabilidad de ser elegidas. A la sucesión de pasos desde la temperatura inicial hasta la final se le suele llamar cadena de Markov, ya que cada estado definido por cada temperatura, depende exclusivamente del estado anterior.

Como función de aceptación de soluciones, suele emplearse la función de Boltzmann:

$$P(S, S_{neig}, T) = exp\left(-\frac{F(S_{neig}) - F(S)}{T}\right) \quad (12)$$

La temperatura, después de elegir la temperatura inicial T_0 de forma experimental, suele variarse (paso 3) según un esquema de enfriamiento. Uno de los esquemas de enfriamiento posibles es el mostrado en la expresión siguiente:

$$T := c \cdot T,\ 0 < c < 1 \quad (13)$$

Este algoritmo permite, en la exploración del espacio de soluciones, escapar de la convergencia excesivamente rápida hacia mínimos locales al permitir, en las primeras fases de la exploración, seguir contando con soluciones cuya función objetivo no es excesivamente prometedora (Kirpatrick *et al.,* 1983).

Se han desarrollado aplicaciones secuenciales del recocido simulado al problema de secuenciación de trabajos en flujo uniforme con el objetivo de minimizar el *makespan* y el tiempo de flujo (Osman y Potts, 1989, Varadharajan y Rajendran, 2005). Sobre su aplicación en paralelo, existen en la literatura diversas referencias acerca del uso de herramientas generales que emplean versiones en paralelo de SA y que podrían aplicarse al problema de secuenciación de trabajos. Entre ellas cabe destacar parSA de Kliewer (2000a), MPSA de Sanvicente y Frausto (2002), el método propuesto por Roussel-Ragot y Dreyfus (1990) y el trabajo teórico de Steinhöfel *et al.* (2002). A continuación, se describen brevemente las características más importantes de estos procedimientos.

parSA (Kliewer, 2000a, y Kliewer, 2000b) es una librería basada en C++ para SA que emplea MPI para la comunicación entre procesos. En su aplicación al problema de asignación semanal de flotas aéreas, obtuvieron aceleraciones algo inferiores a la lineal. Permite emplear procesos independientes (*Multiple Independent Runs – MIR*) o procesos en cooperación bien con una pasada (*Single Walk*) o con varias (*Multiple Walk*).

MPSA (*Methodology to Parallelize Simulated Annealing*) de Sanvicente y Frausto (2002) es un método de paralelización para recocido simulado. Está orientado al empleo de un gran número de procesadores. Sigue una arquitectura de procesos iguales que se comunican de forma asíncrona, mediante difusión de la mejor solución encontrada en cada temperatura. Distinguieron dos variantes de desarrollo en paralelo de SA:

a) Algoritmos pseudoparalelos. Se divide el conjunto de datos entre los procesadores, en los que se ejecuta el mismo código secuencial.

b) Algoritmos paralelos. En este caso, se dividen las tareas entre los procesadores.

En el trabajo de Roussel-Ragot y Dreyfus (1990), de las dos posibilidades de paralelismo en las aplicaciones de SA (separar la ejecución de cada paso o nivel de temperatura o bien separar la ejecución de la sucesión completa de pasos desde la solución de partida hasta la temperatura final) los autores se decantan por esta última opción. Proponen un modelo de implementación paralela de SA de tipo maestro-esclavo sobre una red de *transputers*. Es interesante en este artículo que sus autores pueden estimar la aceleración del mismo. Por otro lado, observaron que la eficiencia del algoritmo aumenta cuando se reduce el número de puntos de sincronización entre procesos.

En Steinhöfel *et al.* (2002) se muestra una estimación del tiempo de computación así como de la calidad de las soluciones al aplicar SA. Además, se presentan cotas superiores tanto para el tiempo de ejecución según el número de procesadores como para el *makespan*. Las pruebas se basan en un ordenador ideal tipo PRAM con gran número de procesadores y un algoritmo paralelo basado en el modelo de Gibbons y Ritter (1988) (ver Figura 8, apartado 2.5.1.4).

Para el problema de secuenciación en flujo uniforme con el *makespan* como función objetivo, Wodecki y Bozejko (2002) proponen dos métodos para implementar el algoritmo de recocido simulado en paralelo. En ambos métodos todos los procesos parten de la misma solución inicial. En el primero, se realiza una búsqueda independiente en cada uno de los procesos sin comunicación entre ellos seleccionando la mejor solución de entre las encontradas por todos los procesos. En el segundo, el proceso que encuentra una solución mejor que la mejor obtenida hasta el momento la difunde al resto de procesos. El modelo de computación paralela es una máquina SIMD sin memoria compartida, esto es, se implementa en un ordenador compuesto por varios procesadores pero sin compartir la memoria principal. El espacio de soluciones no se divide de forma explícita, sino que se generan soluciones al azar a partir de una inicial que es igual para todos los procesos. Esto no garantiza que se exploren regiones diferentes del espacio de soluciones. La ley de enfriamiento es la misma en todos los procesos, partiendo de la misma temperatura inicial.

$n \times m$	1 proc	4 procesadores independientes	4 procesadores con comunicación	NEH
20×5	0.87%	0.64%	0.62%	2.87%
20×10	2.29%	1.82%	1.70%	4.74%
20×20	1.94%	1.91%	1.82%	3.69%
50×5	0.13%	0.08%	0.13%	0.89%
50×10	1.87%	1.31%	0.92%	4.53%
50×20	2.75%	2.32%	2.29%	5.24%
100×5	0.0011%	0.0003%	0.0003%	0.46%
promedio	**1.41%**	**1.15%**	**1.07%**	**3.20%**

Tabla 11: Resultados (ARPD) del algoritmo SA paralelo de Wodecki y Bozejko (2002) y la heurística NEH frente a la batería de Taillard (1993)

La calidad de las soluciones obtenidas se compara en base a su valor de *makespan* con las obtenidas mediante la heurística NEH midiendo la diferencia relativa con la batería de Taillard (1993), tal como reproducimos en la Tabla 11. Los autores comentan que, en cierto sentido, el algoritmo paralelo consigue una aceleración mayor que 4 ya que se emplean 200 iteraciones cuando se ejecuta el algoritmo secuencial, mientras que en el algoritmo paralelo cada uno de los 4 procesadores ejecuta 50 iteraciones. Por tanto, el algoritmo paralelo necesita menos iteraciones que el algo-

ritmo secuencial para alcanzar la misma calidad de soluciones, aunque no comentan nada del tiempo de ejecución en uno y en otro caso.

Como conclusión, los autores afirman que el recocido simulado ofrece mejores resultados que la mejora iterativa debido a la selección aleatoria de soluciones. En segundo lugar, observan la mejora que se obtiene cuando los procesos intercambian información. En tercer lugar, comentan que la frecuencia de comunicación entre procesos no debe ser muy alta para no empeorar la aceleración del algoritmo paralelo. En su caso, la comunicación se produce cada vez que un proceso encuentra una solución mejor que la actual. Según los autores, es una frecuencia de comunicación pequeña.

Los mismos autores han estudiado en Bozejko y Wodecki (2004a) varias modificaciones a este procedimiento para afrontar el mismo problema ahora con la suma de los tiempos de terminación como función objetivo ($F_m/prmu/C_{sum}$). La arquitectura es maestro-esclavo con múltiples trayectorias. El recorrido de la exploración es diferente en cada proceso por emplear leyes de enfriamiento diferentes en cada proceso esclavo y diferentes soluciones de partida.

El proceso maestro es el responsable de la memoria a largo plazo, manteniendo la mejor solución conocida y una lista de soluciones de partida junto con la temperatura empleada. Los procesos esclavos ejecutan SA cada uno de ellos a diferente temperatura. Cuando encuentran una nueva mejor solución global, la envían al proceso maestro y realizan una fase de intensificación durante 10 iteraciones. Por otro lado, si no consiguen mejora durante 20 iteraciones sin modificación de temperatura, toman nuevas soluciones de la lista del proceso maestro. Todos los procesos realizan n iteraciones antes de cambiar de temperatura. Para el mismo número total de iteraciones, el procedimiento paralelo consigue mejor calidad de soluciones que el algoritmo secuencial.

3.3.2.3 Búsqueda tabú

El algoritmo de búsqueda tabú (Glover, 1989) es un algoritmo de tipo búsqueda local en el que se mantiene una memoria de características de las soluciones que ya han sido exploradas o lista tabú. La mayoría de algoritmos de búsqueda local

incluyen una fase de intensificación y otra de diversificación. La fase de diversificación permite escapar de mínimos locales. El uso de memoria evita que se estudien de nuevo soluciones ya exploradas.

En la Figura 21 se muestra un pseudocódigo de búsqueda tabú, donde S es una solución admisible, S_{best} la mejor solución encontrada hasta el momento, T la lista tabú, N el conjunto de soluciones, k el contador de iteraciones, k_{tope} el máximo número de iteraciones, $F(S)$ el valor de la función objetivo en S.

Paso 1. $S_{best}:= S, k:=0, T :=\emptyset$
Paso 2. **Si** $N\text{-}T \neq \emptyset$
    $k:=k+1$
    Seleccionar la mejor $S_k \in N\text{-}T$
Paso 3. $S:=S_k$. Si $F(S)<F(S_{best})$, $S_{best}:=S$
Paso 4. **Si** $k>=k_{tope}$ o si $N\text{-}T=\emptyset$
        Parar
Paso 5. **Si no**
        actualizar T y volver a 2

Figura 21: Versión básica de búsqueda tabú (Glover, 1989)

En este método de búsqueda se consideran tres tipos de memoria. El primero es la memoria a corto plazo, con la que se mantienen elementos característicos de las soluciones visitadas recientemente. De este modo, se impide volver a soluciones que han sido visitadas recientemente, almacenadas en la lista tabú, a menos que cumplan alguna condición o criterio de aspiración. Otros tipos de memoria empleada, a medio y a largo plazo, permiten considerar qué atributos de las soluciones son empleados con mayor frecuencia en soluciones de calidad. Se consideran así frecuencias de transición o cambio en atributos y frecuencias de permanencia de otros. La memoria a corto y a medio plazo apoya la fase de intensificación, mientras que la memoria a largo plazo apoya la fase de diversificación. La intensificación de la búsqueda hace que regiones del espacio donde se han encontrado soluciones de calidad sean exploradas más a fondo, de forma que se encuentren todas las soluciones de calidad que existen en esas zonas.

Una primera aproximación al problema $F_m/prmu/C_{max}$ mediante búsqueda tabú se encuentra en el trabajo de Widmer y Hertz (1989). Respecto a aplicaciones en paralelo de la búsqueda tabú, Talbi *et al.* (1998) desarrollan una versión capaz de adaptar el grado de paralelismo a la carga de trabajo del sistema en el que se ejecuta basado en el modelo MARS (*Multi-user Adaptive Resource Scheduler*) de Hafidi *et al.* (1996) (Véase también Kebbal *et al.*, 2001).

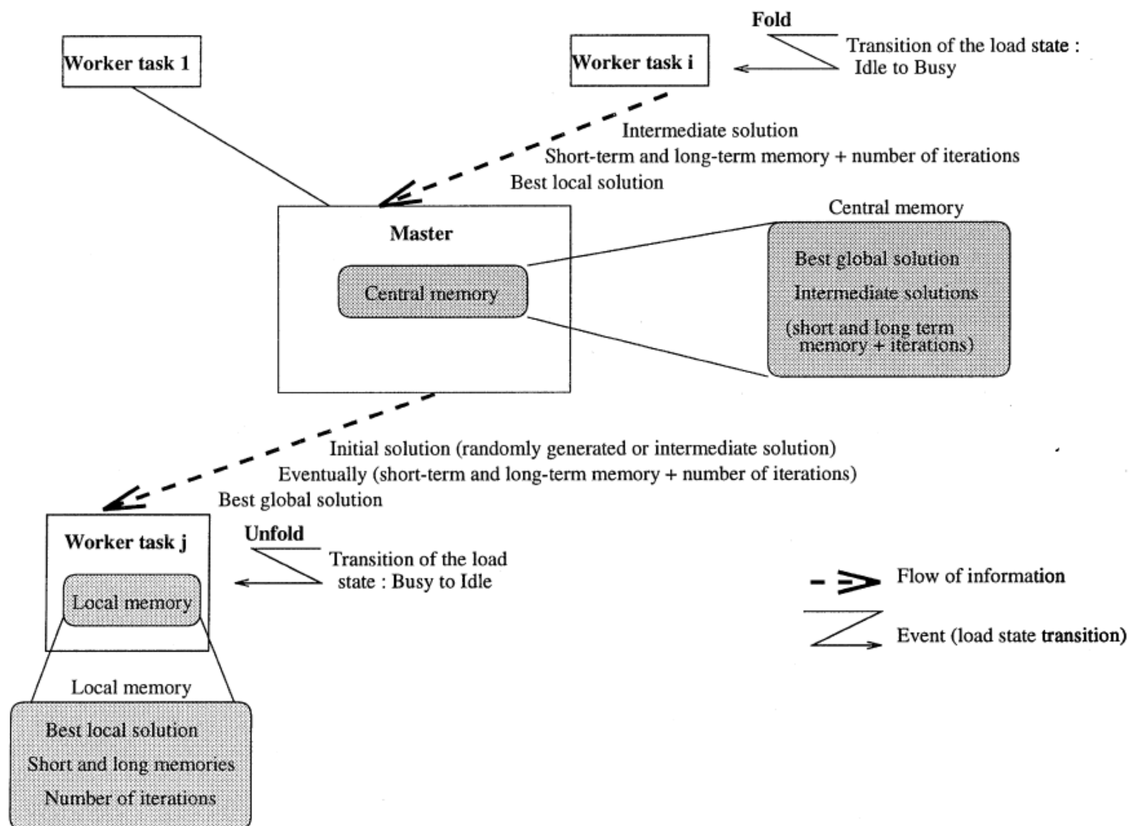


Figura 22: Algoritmo paralelo adaptativo de búsqueda tabú (Talbi *et al.*, 1998)

El algoritmo paralelo de Talbi *et al.* (1998) sigue la arquitectura jefe/trabajador (*master/worker*). El proceso principal o maestro distribuye el trabajo a los procesos trabajadores y recibe las mejores soluciones encontradas. La adaptabilidad del algoritmo consiste en que el proceso principal reacciona ante un nodo ocupado recogiendo su trabajo pendiente, con lo que el proceso trabajador termina, o bien reacciona ante un nuevo nodo disponible iniciando un nuevo proceso trabajador. Finalmente, entrega la solución inicial y la mejor solución global o bien el trabajo

pendiente de otro proceso que se haya detenido sin terminar. Este trabajo consiste en la última solución que estaba siendo explorada, la mejor solución local, el contenido de la memoria tanto a largo plazo como a corto plazo y el número de iteraciones realizadas sin haber mejorado la actual mejor solución local.

Los procesos trabajadores son procesos independientes que ejecutan la búsqueda tabú partiendo de soluciones diferentes y empleando diferentes tamaños de lista tabú. El esquema de funcionamiento del algoritmo paralelo adaptativo se muestra en la Figura 22.

La búsqueda tabú emplea una memoria a corto plazo que contiene una lista de parejas de posiciones que no deben intercambiarse y una memoria a largo plazo consistente en una matriz con el número de veces que ha aparecido cada valor en cada posición, normalizado con el número de iteraciones. El número de procesos trabajadores empleado es la dimensión del problema n y el tamaño de la memoria a corto plazo es diferente en cada uno de los n procesos, desde $n/2$ a $3n/2$ incrementándose de 1 en 1. Las fases de intensificación y diversificación también dependen de la dimensión del problema. La fase de intensificación se realiza tras $100n$ iteraciones sin obtener mejora. Esta fase se realiza durante otras $100n$ iteraciones. La fase de diversificación se realiza durante otras $10n$ iteraciones. La experimentación se desarrolló sobre un sistema compuesto por 126 estaciones de trabajo heterogéneas y una granja con 16 procesadores, debiendo competir la aplicación de búsqueda tabú en paralelo con los usuarios de los ordenadores.

En los experimentos que realizan Talbi *et al.,* (1998), comparándolos con problemas de la librería QAPLIB (Burkard *et al.*, 1997) superan la calidad y tiempo de ejecución en instancias con distancias y flujos pseudoaleatorios. En instancias generadas al azar siguiendo una distribución uniforme en distancias y flujos, se alcanzaron soluciones de calidad ligeramente inferior a las mejores conocidas en su momento. En instancias similares a problemas reales, nuevamente superaron las mejores soluciones conocidas.

Schulze y Fahle (1999) presentan un algoritmo paralelo basado en búsqueda tabú mediante procesos de búsqueda independientes, que parten cada uno de una solución diferente y que trata de mejorarse mediante un proceso de optimización local.

Esta arquitectura no precisa de un proceso que coordine al resto, con lo que se reducen los puntos de sincronización y, por tanto, se mejora la eficiencia del algoritmo paralelo. El problema analizado fue el de asignación de rutas de vehículos con ventanas de tiempo.

Attanasio *et al*. (2004) han comparado ocho métodos de implementación del algoritmo de búsqueda tabú en paralelo para el problema de carga y descarga con ventanas de tiempo y demandas iguales (*Dial a Ride Problem* – DARP), según la búsqueda comience en una o en varias soluciones y según se empleen una o varias estrategias de búsqueda. Su experimentación con algoritmos paralelos, según la clasificación propuesta en el capítulo 2 se centró en dos extremos: un solo programa ejecutándose sobre distintos conjuntos de datos o bien, programas diferentes actuando sobre el mismo conjunto de datos. Ambos siguiendo una arquitectura entre iguales. Cada proceso difunde al resto la mejor solución encontrada sin más información. La versión que ejecutaba diferentes algoritmos sobre el mismo conjunto de datos fue la que ofreció mejores resultados.

En cuanto a referencias aplicadas a problemas de secuenciación, Bożejko y Wodecki (2002) proponen otra versión paralela de búsqueda tabú para la resolución del problema de secuenciación de trabajos en flujo uniforme con el objetivo de minimizar el tiempo de flujo o *flowtime*. Para su implementación, emplean el concepto de bloques de trabajos presentado por Nowicki y Smutnicki (1996). El algoritmo paralelo es muy similar a la versión paralela de recocido simulado ya descrita en el apartado anterior de los mismos autores (Wodecky y Bożejko, 2002).

Los autores desarrollaron el algoritmo sobre un ordenador ideal CREW PRAM (*Concurrent Read Exclusive Write - Parallel Random Access Machine*). En su experimentación, emplearon un ordenador *Sun Enterprise* con 4 procesadores. La longitud de la lista tabú fue de 10 soluciones, la distancia relativa es del 0.25% y el número de iteraciones sin mejora 10. Se hicieron experimentos con 1, 2 y 4 procesadores y las iteraciones totales fueron 2000 y 4000. Es decir, para 4000 iteraciones, con un procesador, éste realizó las 4000 iteraciones, con dos procesadores, cada uno realizó 2000 y con 4 procesadores, cada uno realizó 1000 iteraciones. Igual que en el caso del algoritmo de recocido simulado en paralelo, comparan sus resultados

con los obtenidos con la heurística NEH y comentan que, al obtener mejor calidad de solución con el algoritmo paralelo que con el secuencial para el mismo número de iteraciones, la aceleración debe ser mayor que 4 con los 4 procesadores.

Los mismos autores (Bożejko y Wodecki, 2004b) proponen otras dos versiones en paralelo de búsqueda tabú, una síncrona (PSTS - *Parallel Synchronous Tabu Search*) y otra asíncrona (PATS - *Parallel Asynchronous Tabu Search*). En este caso, el modelo ideal de computación paralela es el EREW PRAM (*Exclusive Read Exclusive Write - Parallel Random Access Machine*). Los autores emplean de nuevo el concepto de bloques para dividir el espacio de búsqueda en secuencias parciales o subpermutaciones cuya vecindad es explorada por cada proceso esclavo.

La versión síncrona, PSTS, es un algoritmo paralelo de una pasada y de un paso, esto es, cada proceso busca únicamente la mejor solución en su vecindad a partir de la solución inicial y devuelve su resultado al proceso maestro, descartando toda la información entre una iteración y la siguiente. La versión asíncrona, PATS, emplea una lista de t soluciones $\alpha = (\alpha_1, \ldots, \alpha_t)$ actuales y dos niveles de paralelismo, que implica dividir en dos el número de procesadores esclavos disponibles. En el primero, cada procesador recibe una de estas soluciones α_i del proceso maestro, la parte en bloques calculando el camino crítico y se la envía a un procesador del segundo nivel. En éste, el procesador determina la permutación representativa de la vecindad generada a partir del bloque recibido y se la envía al proceso maestro. A su vez, el proceso maestro descarta las permutaciones representativas prohibidas por la lista tabú salvo que su valor sea mejor que el mejor conocido. Las permutaciones admitidas, se añaden a la lista α, descartando la peor en caso de que el número de soluciones en la lista sea mayor que t.

Los resultados que obtuvieron tras 1000 iteraciones empleando 1, 4, 8 y 12 procesadores y aplicando estos métodos a las instancias del problema $F_m|prmu|C_{max}$ fueron mejores en la versión PATS que en la PSTS. La calidad de las soluciones mejoró con el número de procesadores y también fue mejor que la obtenida con la heurística NEH en problemas de la batería de Taillard (1993) de 20 trabajos y 5 máquinas hasta los de 100 trabajos y 5 máquinas.

Belkadi *et al.* (2006a) también aplican una versión de la búsqueda tabú en paralelo a la resolución del problema de secuenciación de trabajos en talleres de flujo híbridos. Cada uno de los procesos lleva a cabo una búsqueda tabú y se comunican entre ellos los resultados obtenidos en puntos de sincronización determinados, método al que llaman TS_MSI o *Tabu Search based on Multiple search with Syn-chronous Interaction*. Estudiaron dos variantes, partiendo en la exploración del mismo punto o de puntos diferentes con estrategias de búsqueda diferentes. Se trata de esquemas tipo SPMS y MPMS síncronos que siguen una arquitectura maestro/esclavo. Con la versión MPMS obtuvieron mejora sobre la calidad de soluciones entregada por el algoritmo secuencial.

3.3.2.4 Métodos de búsqueda local exploratoria

En este apartado se realiza una revisión sobre algoritmos basados en búsqueda local, haciendo hincapié en aquellos de los que se han encontrado referencias sobre su aplicación al problema $F_m/prmu/C_{max}$ y a problemas similares.

Este tipo de métodos de búsqueda exploratoria suele emplear los conceptos de vecindad, movimiento y mínimo local. Se define la vecindad o entorno de una solución como una función $N: S \rightarrow 2^S$ que asigna a cada solución $s \in S$ un conjunto de vecinas $N(s) \subseteq S$. Ésta se emplea para realizar los cambios que deben hacerse sobre una solución para generar todas sus vecinas. Se suele llamar movimiento a la operación que produce una vecina $S' \in N(S)$ de una solución S (Alba *et al.*, 2005).

El concepto de mínimo local se define con respecto a un entorno o estructura de vecindad N como una solución $\hat{S}$ tal que $\forall S \in N(\hat{S}): F(\hat{S}) \leq F(S)$. $\hat{S}$ es un mínimo local estricto si $\forall S \in N(\hat{S}): F(\hat{S}) < F(S)$.

A continuación, se presentan los distintos tipos de búsqueda local que pueden encontrarse en la literatura aplicados al problema de estudio.

3.3.2.4.1 Algoritmos voraces

El trabajo de Ruiz y Stützle (2007) presenta un algoritmo voraz iterado o *Iterated Greedy* – IG aplicado al problema $F_m/prmu/C_{max}$. La particularidad de este algo-

ritmo, según sus autores, es que no hace uso de características concretas del problema, luego podría aplicarse a problemas de secuenciación en flujo uniforme con otros objetivos o incluso a otras variantes del problema.

Con respecto a la aplicación de GRASP (*Greedy Randomized Adaptive Search Procedure*) en paralelo a problemas similares al de estudio, en Pardalos *et al.* (1994) se consiguen aceleraciones cercanas a la lineal en función del número de procesadores cuando se trata de su aplicación a la resolución del problema de asignación cuadrática. Hace hincapié en el hecho de que la implementación en paralelo de GRASP únicamente requiere que todos los procesos conozcan la mejor solución encontrada hasta el momento, con lo que se reducen las necesidades de comunicación entre procesos y por tanto se mejora la eficiencia del algoritmo. Esto no sucede con algoritmos que requieren mayor comunicación entre procesos, por ejemplo, en implementaciones en paralelo de búsqueda tabú en las que no solo se envían las mejores soluciones sino también el tamaño de la lista tabú o incluso el contenido de esta (Bożejko y Wodecki, 2002). También sucede lo mismo en el caso de recocido simulado, en el que además de la mejor solución encontrada junto con su valor de la función objetivo, se comunica la temperatura inicial (Wodecki y Bożejko, 2002).

En el caso de Aiex *et al.* (2002), se justifica la posibilidad de alcanzar aceleración lineal con la implementación en paralelo de GRASP mediante procesos independientes, por la distribución de probabilidad del tiempo para encontrar una solución subóptima mediante el algoritmo secuencial, tal como ya demostraron Verhoeven y Aarts (1995) para otros tipos de heurísticas de búsqueda local en paralelo. Aiex *et al.* (2002) demuestran que el tiempo para encontrar una solución subóptima mediante varias implementaciones secuenciales de algoritmos voraces sigue una distribución de probabilidad exponencial de dos parámetros, tal como aparece en la ecuación siguiente.

$$F(t) = 1 - e^{-(t-\mu)/\lambda} \tag{14}$$

Donde λ es la media de la distribución y μ el desplazamiento de la misma respecto al origen y por tanto, puede alcanzarse aceleración lineal. En el mismo sentido apunta el trabajo de Aiex *et al*. (2005). Demuestra que, en su implementación de GRASP junto con *path relinking* aplicada al problema de asignación tridimensional y al problema de secuenciación en entornos tipo taller, la variable aleatoria "tiempo para encontrar una solución con una calidad dada" sigue una distribución exponencial con dos parámetros y por tanto, su implementación en paralelo puede alcanzar aceleración lineal.

En su trabajo, Aiex *et al*. (2005) estudian una estrategia paralela empleando procesos independientes. Un proceso se encarga de leer los datos del problema y distribuirlos al resto, que funcionan de forma independiente hasta encontrar una solución de una calidad dada, para devolverla después al proceso principal. Los procesos comienzan con un algoritmo GRASP hasta completar el tamaño de la lista *elite*. A partir de aquí, si se encuentra en la fase de búsqueda local de GRASP una solución con calidad igual o mejor que las que se encuentran en la lista *elite*, se emplea *path relinking* entre ésta y las soluciones de la lista con objeto de encontrar una de mejor calidad. En ese caso, se incluye en la lista *elite*, descartando la peor de la lista. Después de un número de iteraciones, se pasa a una fase de intensificación de GRASP y se termina con la aplicación de *path relinking* entre las soluciones que han quedado en la lista *elite*. Con esta estrategia, consiguen aceleración aproximadamente lineal con 4 combinaciones de GRASP con Path Relinking en paralelo, al menos hasta con 8 procesadores. Empleando un mayor número de procesadores, se degrada la aceleración del algoritmo paralelo.

No se han encontrado hasta ahora referencias sobre la aplicación en paralelo de GRASP a la resolución del problema objeto de estudio.

3.3.2.4.2 Búsqueda de entorno variable

El método de búsqueda de entorno variable (*Variable Neighborhood Search* – VNS – Hansen y Mladenovic, 2001), se basa en la búsqueda en diferentes entornos de vecindad entre los que se alterna la búsqueda con objeto de evitar el estancamiento en mínimos locales. Un pseudocódigo de VNS se muestra en la Figura 23.

Paso 1. Solución inicial S. $k := 1$

Paso 2. Mientras no se alcance la condición de parada

a) Agitación:

$S := S'$ Generar una solución aleatoria $S' \in N_k(S)$

b) Búsqueda local:

Aplicar búsqueda local desde S' mediante la estructura de vecindad $N(\cdot)$ hasta encontrar un mínimo local S''

Figura 23: Pseudocódigo de VNS (adaptado de García López *et al.*, 2002)

VNS es un método flexible ya que según el mecanismo de selección de soluciones de la vecindad o entorno, se obtienen diferentes variantes del método. Hansen y Mladenovic (2001) presentan las tres primeras variantes. La cuarta se describe en Hansen *et al*. (2003):

- La búsqueda de entorno variable descendente, (*Variable Neighborhood Descent,* VND) cuando el mecanismo de selección es totalmente *greedy*, cambiando de estructura de vecindad cada vez que se alcanza un mínimo local.
- La búsqueda de entorno variable reducida (*Reduced Variable Neighborhood Search*, RVNS) se suele aplicar a problemas grandes y consiste en seleccionar soluciones al azar dentro de la vecindad, hasta llegar a un número de iteraciones sin mejora, momento en el que se cambia de vecindad, repitiendo también hasta un número de iteraciones sin mejora.
- La búsqueda de entorno variable básica (*Basic Variable Neighbourhood Search*, BVNS) combina métodos deterministas y aleatorios en los cambios entre soluciones o entre entornos.
- La búsqueda de entorno variable general (*General Variable Neighbourhood Search*, GVNS) cambia la búsqueda local que se realiza en la BVNS por una VND.

En la literatura, no se encuentran apenas referencias de la aplicación de VNS al problema de secuenciación en flujo uniforme. Una de ellas está en Den Besten y Stützle (2001), que estudian su aplicación a cuatro problemas: el problema de secuenciación de n trabajos en una máquina con el objetivo de minimizar el retraso total, el mismo problema para minimizar el retraso total ponderado, mínimo tiempo total de terminación ponderado y el problema de secuenciación de n trabajos en m máquinas. Para este problema, se seleccionan dos tipos de vecindad: el intercambio general, y la inserción. Emplear inserción en el problema de secuenciación en flujo uniforme dio mejores resultados en cuanto a tiempo de ejecución con una calidad de soluciones similar con respecto a las otras variantes de VNS. Esto contradice la idea de que sea mejor emplear primero la vecindad de menor tamaño. La estrategia empleada antes de cambiar de vecindad es la de la mayor mejora, esto es, se selecciona la mejor solución posible de toda la vecindad, y ésta se toma como punto de partida para la siguiente iteración con la otra estructura de vecindad.

No se han encontrado referencias de la aplicación de VNS en paralelo al problema $F_m/prmu/C_{max}$. Ochi *et* al. (2001) combina VNS y GRASP en su aplicación a una generalización del problema del viajante (*Traveling Purchaser Problem* – TPP) y Sevkli y Aydin (2006) aplicado a entornos tipo taller son las aplicaciones e problemas parecidos.

En el algoritmo paralelo de Ochi *et al.* (2001) varios procesos ejecutan GRASP y el resto VNS dando mejores resultados en cuanto a tiempo y calidad de soluciones que versiones secuenciales de ambos algoritmos y que las versiones paralelas en las que todos los procesos emplean GRASP o VNS, obteniendo valores de aceleración del orden del número de procesadores. Experimentaron con versiones paralelas maestro esclavo y también distribuidas o entre iguales o *peer-to-peer*. La coordinación de la búsqueda, en lugar de realizarla un solo proceso, se realiza dividiendo cada proceso en otros dos: uno realiza la búsqueda y el otro comprueba las condiciones de terminación y los intercambios de información con el resto de procesos. Sobre las versiones distribuidas no han presentado resultados experimentales.

En el caso de Sevkli y Aydin (2006), se experimenta con cuatro formas de implementar VNS en paralelo: dos algoritmos cliente-servidor con intercambios de infor-

mación síncronos y asíncronos y dos algoritmos con procesos iguales con diferentes arquitecturas de comunicación, en malla o en anillo unidireccional. En la versión cliente-servidor se obtuvieron mejores resultados en cuanto a calidad de las soluciones y tiempo de ejecución con la versión síncrona, mientras que en el caso de procesos iguales se obtuvieron mejores resultados con la versión en anillo, que fueron incluso superiores a los de la versión maestro-esclavo. Las arquitecturas de comunicación se refieren a la forma en que se comunican los procesos entre sí. Los procesos se numeran de forma correlativa. En la estructura de malla, cada proceso comienza la siguiente generación de soluciones seleccionando como punto de partida la mejor de tres soluciones: las comunicadas por el proceso anterior y posterior y la generada por él mismo. En el caso del anillo, cada proceso toma como inicial la solución que le comunica el proceso anterior. Es un anillo porque el primer proceso toma la solución que le envía el último.

3.3.2.5 Búsqueda local iterativa

Los procedimientos de búsqueda local iterativa (*Iterated Local Search* – ILS), propuestos inicialmente por Martin *et al.* (1991), han dado buenos resultados en su aplicación al problema $F_m|prmu|C_{max}$, según aparece en Stützle (1998a). El procedimiento para escapar a mínimos locales, en este caso, consiste en perturbar la solución cada vez que se alcanza un mínimo local. Hasta el momento, no se han encontrado aplicaciones al problema de secuenciación en flujo uniforme de ILS en paralelo.

3.3.3 Métodos basados en poblaciones

3.3.3.1 Computación evolutiva

Dentro de la computación evolutiva se encuentran tres tipos principales de procedimientos. La programación evolutiva, las estrategias de evolución y los algoritmos genéticos. Por su importancia con respecto a los otros dos procedimientos, dedicamos el estudio más detallado a los algoritmos genéticos.

3.3.3.2 Algoritmos genéticos

Las posibilidades que brinda la imitación de los procesos evolutivos para resolver problemas complejos han sido aprovechadas en muchas ocasiones para abordar los problemas de optimización combinatoria. Muchos autores distinguen entre algoritmos evolutivos y genéticos según los operadores utilizados. Así, los algoritmos evolutivos emplean operadores de mutación y selección, mientras que los algoritmos genéticos añaden operadores de recombinación entre individuos. Las variantes actuales incluyen versiones híbridas con otros algoritmos, por ejemplo, en Digalakis y Margaritis (2004). Posteriormente, en el capítulo 6 se incluye una referencia de estos métodos híbridos.

En cuanto a la ejecución en paralelo de algoritmos genéticos existen algunas revisiones en la literatura, entre las que destacamos las de Alba y Troya (1990), Cantú-Paz (1997), Nowostawski y Poli (1999), Cantú-Paz (2001) y Vallada (2006).

En Cantú-Paz (1997) aparecen referencias al problema de secuenciación. Las referencias citadas son aplicaciones al problema de secuenciación en flujo taller, (Tamaki y Nishikawa, 1992, y Liu *et al.,* 1997). El mismo autor destaca en Cantú-Paz (2001) que con este tipo de algoritmos paralelos puede lograrse aceleración superior a la lineal.

Otro trabajo de revisión muy extensa de la literatura, no sólo en cuanto a algoritmos genéticos en paralelo, sino también de búsqueda tabú y de recocido simulado en paralelo se encuentra en Vallada (2006) (ver Tabla 12).

En Vallada (2006) se propone una versión en paralelo de algoritmo genético aplicado al problema de secuenciación en talleres de flujo con fechas de entrega como función objetivo. Aunque no realiza directamente un estudio de la eficiencia de su algoritmo paralelo, sí se puede pensar que ofrece una aceleración lineal, ya que obtiene mejores calidades de solución cuando lo compara con otros algoritmos secuenciales de la literatura ejecutados de forma independiente un número de veces igual al número de procesadores disponibles en el sistema paralelo.

En Abramson y Abela (1992) se propone una versión en paralelo de algoritmo genético para resolución del problema de horarios de clases. Este trabajo pone de

manifiesto que la naturaleza inherentemente paralela de una de las fases del algoritmo genético, la generación de nuevos individuos, hace que sea relativamente fácil alcanzar una aceleración lineal con este tipo de algoritmos paralelos.

Problema	**Algoritmo**	**Referencia**	
Asignación cuadrática	SA	Mühlenbein (1989)	Laursen (1994)
		Boissin y Lutton (1993)	
Cela (1998)			
	TS	Taillard (1991)	Battiti y Tecchioli (1994)
		Battiti y Tecchioli (1992)	De-Falco *et al.* (1994)
	GA+SA	Huntley y Brown (1991)	
	TS+SA	Talbi *et al.* (1998)	Talbi *et al.* (1998)
	GRASP	Pardalos *et al.* (1995b)	Li *et al.* (1995)
	SS	Cung *et al.* (1997)	
	GA+TS	Bachelet y Talbi (2000)	
	AC	Talbi *et al.* (1999)	Talbi *et al.* (2001)
	PR	James *et al.* (2005)	
TSP	SA	Felten *et al.* (1985)	Ram *et al.* (1993)
		Allwright y Carpenter (1989)	Pao, Lam y Fang (1999)
Lawler *et al.*(1985)		Jeong y Kim (1991)	Bavilacqua (2002)
		Nabhan y Zomaya (1995)	Sanvicente y Frausto (2002)
		Diekmann y Simon (1993)	
	SA+GA	Miki *et al.* (2003)	Chen, Flan y Watson (1998)
	GA	Mühlenbein *et al.* (1987)	Kohlmorgen *et al.* (1999)
		Mühlenbein *et al.* (1988)	Sena *et al.* (2001)
		Knight y Wainwright (1992)	Baraglia *et al.* (2001)
		Logar, *et al.* (1992)	KLatayama *et al.* (2003)
	TS	Malek *et al.* (1989)	De-Falco *et al.* (1994)
		Chakrapani y Skorin-Kapov (1993)	Fiechter (1994)
	AC	Stützle (1998b)	Middlendorf *et al.* (2000)
		Bullnheimer *et al.* (1998)	Randall y Lewis (2002)
VRP	TS	Rego y Roucairol (1996)	
	GA	Ochi *et al.* (1998)	Alba y Dorronsoro (2004)
VRP con ventanas de tiempo	SA	Czech y Czarnas (2002)	
	GA	Berger y Barkaoui (2004)	
	TS	Taillard (1993b)	Rochat y Taillard (1995)
	CS	Gehring y Hamberger (2001)	
	TS+GA	Le Bouthillier y Crainic (2005)	
Coloreado de grafos	GA	Kokosinski *et al.* (2005)	
Partición de grafos	SA	Durand (1989)	Laursen (1994)
		Lee y Lee (1992)	
	GA	Mühlenbein (1991)	Maruyama *et al.* (1993)
		Collins y Jefferson (1991)	Lin *et al.* (1994)
		Talbi y Bessiere (1991)	Diekmann *et al.* (1996)
	CS	Hidalgo *et al.* (2003)	Baños *et al.* (2004)
		Toulouse *et al.* (1999)	
Satisfacibilidad booleana – SAT ó MAX-SAT	SA	Sohn (1996)	
	GA	Wilkerson y Neemer-Preece (1998)	Folino *et al.* (1998)
	IS	Pitsoulis *et al.* (1996)	Singer y Wagner (2006)
	Varios	Hyvarinen *et al.* (2006)	

AC: Colonias de Hormigas — PR: Path Relinking
CS: Búsquedas Cooperativas — SA: Recocido Simulado
GA: Algoritmos Genéticos — SS: Scatter Search
IS: Búsquedas Independientes — TS: Búsqueda Tabú

Tabla 12: Referencias en Vallada (2006)

Otras aproximaciones de estos métodos implementadas con éxito en paralelo son el algoritmo genético celular con perturbaciones, de Kirley (2002) y la paralelización mediante programación multihilo, en Oguz *et al.* (2003), aplicada al problema de secuenciación con máquinas en paralelo. También existen versiones que emplean un tipo de comunicación mínima entre procesos sin alterar prácticamente el algoritmo secuencial, ver Salhi *et al.* (1998), o aprovechando la división del espacio de soluciones para mejorar las prestaciones del algoritmo secuencial, como en Solar *et al.* (2002).

Se han desarrollado también metodologías para el desarrollo e implementación de algoritmos genéticos paralelos. Se pretende así lograr que el diseño de algoritmos en paralelo sea más simple y portable, no solo a nivel de código. Muchos de ellos se muestran en Cahon *et al.* (2004), proponiendo una metodología basada en C++ que puede emplearse no sólo para la implementación en paralelo de algoritmos evolutivos, sino de cualquier metaheurística. Otro desarrollo similar es el realizado por Surry y Radcliffe (1994) en forma de un lenguaje de programación interpretado.

Otros trabajos sobre algoritmos genéticos en paralelo son los siguientes:

Belkadi *et al.,* (2006b) y Belkadi *et al.,* (2006c) son dos algoritmos aplicados al problema de secuenciación en flujo híbrido. Ofrecen buenos resultados en cuanto a eficiencia y calidad de soluciones, observando además que la eficiencia y la calidad dependen fuertemente de los parámetros del algoritmo y de la frecuencia con la que los procesos intercambian soluciones.

El algoritmo de França *et al.* (2006) se aplica al problema de secuenciación con tiempos de preparación dependientes de la máquina. Es un algoritmo paralelo, aunque los autores únicamente presentan experiencias de una versión ejecutada en un equipo monoprocesador.

En Digalakis y Margaritis (2004), se muestra un algoritmo memético. Existen varias posibilidades de ejecutarlo en paralelo: dividir la población, acometer la resolución de las funciones de recombinación y evaluación en paralelo, o combinaciones de ambas. Los autores han empleado la segunda opción por sus mejores prestaciones respecto a las otras dos. Cada proceso mantiene una subpoblación junto con una

lista de procesos donde encontrar otros individuos. La arquitectura es maestro-esclavo. Los intercambios de información con el proceso maestro son asíncronos excepto al comienzo de cada generación, cuando este proceso difunde al resto los individuos que comenzarán a evolucionar en paralelo. El mejor resultado se obtuvo combinando con el algoritmo genético la búsqueda local guiada (GLS) frente a la búsqueda tabú y al recocido simulado, cuando se aplicó el algoritmo a la resolución de una batería de diez funciones no lineales multimodales.

En Kohlmorgen *et al.* (1999) se presenta un estudio de varias aplicaciones de los algoritmos genéticos en paralelo. Entre ellas se encuentra una aplicación al problema de secuenciación en flujo uniforme. Los autores destacan las ventajas del algoritmo paralelo sobre el secuencial en cuanto a la aceleración del mismo y en cuanto a la calidad de las soluciones al diversificar la población en tantas subpoblaciones como procesadores disponibles en el sistema. Emplearon como mecanismo de combinación el de cruce.

3.3.4 Scatter Search y Path Relinking

Scatter Search (Glover, 1977) posteriormente reformulada también como *Path Relinking* en Glover (1998) y en Laguna y Martí (2003), es un método de búsqueda que trabaja con un conjunto de soluciones de referencia. Mediante este procedimiento, se construye un camino entre dos soluciones del conjunto inicial, tomándose como candidato a agregar al conjunto inicial el mejor elemento del camino. Un pseudocódigo del algoritmo *scatter search* secuencial se muestra en la Figura 24.

En Haq *et al.* (2007) se propone una versión secuencial de *scatter search* que, según sus autores, supera a la búsqueda tabú en calidad de las soluciones en su aplicación a la resolución de problemas de secuenciación en flujo uniforme.

Una revisión de las posibilidades de implementación en paralelo de *scatter search* se encuentra en Adenso-Díaz *et al.* (2006), aplicada al problema de la mochila.

En Bożejko y Wodecki (2008a) se describe una aplicación de *scatter search* en paralelo para el problema de secuenciación de trabajos en flujo uniforme con el objetivo de minimizar el *makespan.* Su trabajo se basa en la aplicación en paralelo de *path relinking* de James *et al.* (2005) al problema de asignación cuadrática. Igual

que en otros trabajos presentados por Bożejko y Wodecki, emplean una arquitectura SIMD sin memoria compartida para su algoritmo paralelo, dado que la experimentación la desarrollan posteriormente en un clúster con 152 procesadores y memoria distribuida.

Paso 1. Generar soluciones.

Paso 2. Admisibilidad y mejora de B soluciones, para construir la población Pop.

Paso 3. Seleccionar b soluciones para construir el conjunto de referencia (*RefSet*).

Paso 4. Combinar las soluciones de *RefSet*.

Paso 5. Admisibilidad y mejora de soluciones.

Paso 6. Actualizar *RefSet* y en el caso de añadir nuevas soluciones, volver al paso 4.

Figura 24: Pseudocódigo de *scatter search* (adaptado de Adenso-Díaz *et al.*, 2006)

Los autores proponen dos modelos de comunicación. El primero, al que llaman global, se basa en una arquitectura cliente-servidor, donde cada procesador ejecuta *path relinking* sobre sus datos locales y se comunica con el resto de procesadores para ir creando un conjunto común de soluciones de partida que serán empleadas en sucesivas iteraciones. El segundo modelo, al que llaman independiente, hace ejecutar la búsqueda a cada procesador por separado, tomando sólo al final la mejor de todas ellas. A diferencia de otros trabajos de los mismos autores, en éste presentan los tiempos de ejecución totales del algoritmo y de cpu para las 50 instancias del problema de estudio tomadas de la librería OR (OR-Library, 2008).

En la experimentación que realizan emplean de 1 a 8 procesadores. Cuando se emplean 2 procesadores, cada uno ejecuta 800 iteraciones, cuando se emplean 4 procesadores, 400 cada uno y con 8, 200 iteraciones por procesador. Los autores concluyen que el algoritmo paralelo en la versión con comunicación entre procesos alcanza aceleración superlineal porque emplea el mismo tiempo o incluso menos con

varios procesadores para llegar a soluciones de mayor calidad que la obtenida con el algoritmo secuencial. Por otro lado, es lógico el empeoramiento de la calidad de las soluciones cuando se emplean menos iteraciones en cada uno de los procesos si no existe comunicación entre ellos.

En cuanto a aplicaciones en paralelo de *path relinking*, los mismos autores en Bożejko y Wodecki (2008b) proponen una versión basada en *scatter search* a los problemas $F_m/prmu/C_{max}$ y $F_m|prmu|C_{sum}$. Los pasos de *path relinking* se toman del trabajo de Reeves y Yamada (1998). La experimentación se lleva a cabo en un ordenador SGI Altrix con 128 procesadores y memoria distribuida. El modelo de algoritmo paralelo es análogo al trabajo anterior (Bożejko y Wodecki, 2008a). Los autores vuelven a emplear los modelos global e independiente. Nuevamente los resultados obtenidos indican una aceleración superlineal empleando esta vez de 1 a 16 procesadores.

3.3.5 Optimización mediante colonias de hormigas (ACO)

Otros procedimientos hacen hincapié en el concepto de inteligencia colectiva. Dentro de estos sistemas se encuadran los algoritmos de tipo colonia de hormigas (*Ant Colony Optimization* – ACO). Una primera descripción de estos algoritmos se encuentra en Dorigo *et al.* (1996).

Los algoritmos basados en colonias de hormigas imitan el comportamiento de éstas para conseguir, en su aplicación a problemas de optimización combinatoria, tres ventajas según Dorigo *et al.* (1996):

- El descubrimiento rápido de soluciones de calidad gracias a su realimentación positiva.
- Evitar la convergencia rápida hacia mínimos locales gracias a la computación distribuida.
- Localizar soluciones aceptables en las etapas iniciales del proceso de búsqueda gracias al empleo de una heurística constructiva voraz.

```
Metaheuristica_ACO()
 Paso 1.    mientras (no_condicion_terminacion)
            a.  programar_actividades
                    i.  generacion_hormigas_y_actividad ();
                   ii.  evaporacion_feromonas();
                  iii.  acciones_base(); {opcional}
            b.  fin programar_actividades
 Paso 2.        fin
Función generacion_hormigas_y_actividad()
 Paso 1.     mientras (recursos_disponibles)
            a.  programar_creacion_hormiga_nueva();
            b.  nueva_hormiga_activa ();
 Paso 2.     fin
Función nueva_hormiga_activa ();
Paso 1. inicializar_hormiga();
Paso 2. M= actualiza_memoria_hormiga ();
Paso 3. mientras (estado_actual ≠ estado_objetivo)
    3.1. A = leer_tabla_local_de_rutas_de_hormigas ();
    3.2. P = calcular_probabilidades_transicion (A, M, restricciones_problema);
    3.3. siguiente_estado = aplicar_politica_decision_hormigas (P, restricciones_problema);
    3.4. moverse_al_siguiente_estado (siguiente_estado);
    3.5. si (actualizacion_feromona)
         3.5.1.  depositar_feromona_en_el_arco_visitado ();
         3.5.2.  actualizar_tabla_rutas_hormigas ();
    3.6. M = actualizar_estado_interno ();
Paso 4. si (retrasada_actualización_feromona()
    4.1. evaluar_solucion ();
    4.2. depositar_feromona_en_todos_los_Arcos_visitados ();
    4.3. actualizar_tabla_rutas_hormigas();
Paso 5. finalizar();
Paso 6. fin
```

Figura 25: Pseudocódigo (Dorigo y Di Caro, 1999) de optimización mediante colonias de hormigas

En la Figura 25 se muestra un pseudocódigo de la metaheurística propuesto por Dorigo y Di Caro (1999) para problemas estáticos, aquellos cuyas características no varían mientras se están resolviendo. Un ejemplo de problema dinámico es, por ejemplo, el de enrutamiento en redes.

Existen dos aplicaciones secuenciales de algoritmos basados en colonias de hormigas que han dado buenos resultados en su aplicación al problema $F_m/prmu/C_{max}$ en Rajendran y Ziegler (2004), mejorando la mayoría de los resultados de Liu y Reeves (2001) y en Ying y Liao (2004). Sin embargo, no se han encontrado aplicaciones en paralelo de colonias de hormigas al problema $F_m/prmu/C_{max}$. Otras aplicaciones que podrían servir de punto de partida para una aplicación al problema objeto de estudio son Stützle (1998b), Talbi *et al.* (2001) y Randall y Lewis (2002):

Una aplicación bastante simple de colonias de hormigas en paralelo para el problema del viajante es la que aparece en Stützle (1998b). Aunque se presentan varias estrategias de implementación en paralelo, sólo aparece referenciada la experimentación para varios procesos independientes. En lugar de comparar la aceleración o la eficiencia de la implementación, se compara la calidad de las soluciones obtenidas empleando un tiempo de computación igual al empleado con un solo procesador dividido por el número de procesadores empleado. Sólo se alcanzan soluciones de mejor calidad que con el algoritmo secuencial en un pequeño porcentaje de los experimentos.

Talbi *et al.* (2001) desarrollan una versión en paralelo de un algoritmo de colonia de hormigas complementado con un método de búsqueda tabú. En su aplicación al problema de asignación cuadrática, consiguen mejores resultados que otros métodos basados en colonias de hormigas y que métodos basados en la ejecución de múltiples instancias independientes de búsqueda tabú.

Randall y Lewis (2002), en su aplicación al problema del viajante, han obtenido valores de eficiencia entre el 20 y 55%, según distintas implementaciones habiendo previsto unos valores máximos teóricos de entre el 17 y el 50%. Los autores proponen cinco métodos de paralelización: colonias de hormigas paralelas independientes o interactivas, hormigas paralelas, evaluación en paralelo de elementos de la solución y combinación en paralelo de hormigas y evaluación de elementos de la solución. Los métodos que incluyen la evaluación de elementos de la solución en paralelo son adecuados a problemas en los que esta evaluación es más difícil. El mejor método para problemas como el TSP es el tercero, en el que se realiza una exploración completa por cada proceso-hormiga.

3.4 Resumen de prestaciones de algoritmos paralelos en cuanto a eficiencia y aceleración

Aiex *et al.* (2002) implementaron algunas versiones en paralelo de GRASP con *Path relinking* al problema de asignación con tres índices. En sus experimentos, mediante un ordenador SGI Challenge con 28 procesadores, obtuvieron valores de eficiencia que se resumen en la Tabla 13.

Procesadores	2	4	8	16
Eficiencia	1.04	1.08	0.96	0.69

Tabla 13: Eficiencia del algoritmo paralelo propuesto por Aiex *et al.* (2002)

Entre los autores que han basado su experimentación con algoritmos paralelos en el empleo de redes de *transputers* está Eric Taillard. En Taillard (1990) consiguió valores de aceleración entre 1.92 y 1.99 al resolver el problema de secuenciación en entorno taller con 10 trabajos y 10 máquinas empleando dos *transputers* y una arquitectura maestro-esclavo en la que el trabajo del maestro era muy inferior al de los esclavos, con lo que le permitía correr experimentos con tres procesos en dos procesadores, el maestro y un esclavo en uno y otro esclavo en el otro.

Taillard (1991) consiguió una eficiencia de 0.85 empleando 10 *transputers* en anillo. Nuevamente, se observó una ligera disminución en la eficiencia con el número de procesadores. Esto confirma nuevamente la ley de Amdahl a pesar de emplear la aproximación más cercana al “ejército de hormigas” (ver capítulo 2 sección 4). Taillard (1991) predijo la posibilidad de este comportamiento al emplear un método consistente en comenzar el algoritmo muchas veces partiendo, en la exploración, desde diferentes soluciones y el resultado se basa en la observación de la distribución de probabilidad de no haber encontrado una solución de una calidad dada después de un número de iteraciones. Se trata de una distribución exponencial, con lo que sería posible alcanzar una aceleración superlineal con ese tipo de algoritmo paralelo. Este resultado fue también demostrado por Verhoeven y Aarts (1995).

		Aceleración: trabajos				Eficiencia: trabajos			
		5	20	40	60	5	20	40	60
máquinas	2	1.9	1.7	1.6	1.5	0.95	0.85	0.8	0.75
máquinas	4	3.3	3.1	3.0	2.9	0.83	0.78	0.75	0.73
máquinas	6	4.9	4.4	4.0	3.8	0.82	0.73	0.67	0.63
máquinas	10	7.3	5.5	4.8	4.4	0.73	0.55	0.48	0.44
máquinas	20	8.4	5.6	4.8	4.4	0.48	0.28	0.24	0.22

Tabla 14: Aceleración y eficiencia del algoritmo paralelo propuesto por Taillard (1994)

Otra versión de búsqueda tabú en paralelo fue presentada por Taillard (1994). Igualmente, se aplicó al problema de secuenciación en entorno taller y se implementó en un anillo de *transputers*. Se empleó el mismo número de *transputers* que de máquinas en el problema y los resultados en cuanto a aceleración se resumen en la Tabla 14 (nuevamente se han calculado los valores de eficiencia para hacerlos comparables al resto de referencias).

Las prestaciones relativamente pobres de este algoritmo cuando se empleaban muchos procesadores se debían, según Taillard (1994), al límite teórico de la aceleración S, siendo N el total de operaciones y A_i el máximo de operaciones en un camino desde 0 hasta i, cuando se representa el problema en un grafo:

$$s = \frac{N}{\max_i A_i} \quad (15)$$

Autores como Adenso-Díaz *et al.* (2006) han estimado un valor bajo de eficiencia, 0.38, en una versión paralela de *scatter search* simulando un sistema de 10 procesadores en un sistema monoprocesador. Propusieron varias formas de paralelizar la metaheurística y llevaron a cabo numerosos experimentos. Explicaron que este valor bajo de eficiencia se debe a emplear condiciones de parada diferentes entre las versiones paralela y secuencial del algoritmo.

Hay autores que presentan sus resultados según aceleración en lugar de eficiencia. Citamos ahora algunos de ellos, traduciendo los valores de aceleración que presentan en valores de eficiencia, de forma que sean comparables a los anteriores.

Procesadores	**2**	**3**	**4**	**5**	**6**	**7**
Aceleración	12	13	12.5	12.5	14.5	16
Eficiencia	6	4.333	3.125	2.5	2.417	2.286

Tabla 15: Aceleración y eficiencia obtenidas por el algoritmo paralelo propuesto por Malek *et al.* (1989)

Algunos autores han conseguido aceleración superlineal con sus algoritmos, como por ejemplo en Malek *et al.* (1989) con una versión paralela de SA. Los resultados que se resumen en la Tabla 15 deben tomarse como aproximados ya que no comparan los tiempos necesarios para alcanzar soluciones de la misma calidad sino en alcanzar la mejor solución. Los valores serían mejores si se considera que la calidad de las soluciones que obtuvieron fue mejor cuanto mayor fue el número de procesadores. En el caso de la búsqueda tabú en paralelo, no han obtenido valores de aceleración significativos al emplear un proceso padre y 4 procesos hijos. En cambio, la calidad de las soluciones encontradas con búsqueda tabú fue muy superior a la encontrada por el recocido simulado en el mismo número de iteraciones.

Aunque la aceleración indicada por Liu y Tseng (2000) sea mejor que lineal, decrece ligeramente con el número de procesadores. Según los autores, se debe a la sobrecarga que produce la red de comunicación y podría resolverse con una red más rápida. Éstas y otras versiones de heurísticas en paralelo se resumen en la Tabla 16.

Según cada implementación, el comportamiento observado es muy dispar en cuanto a la aceleración o a la eficiencia con el número de procesadores. En algunos casos, los mejores resultados se obtienen con muchos procesadores, mientras que en otros se obtienen con pocos procesadores.

Cita	**Problema**	**Metaheurística**	**Dimens. del problema**	**np**	**Sistema**	**Aceleración**
Abramson y Abela (1992)	School Timetabling	GA	15x15x15	1, 2, 5, 10, 15	Encore Multimax shared memory multiprocessor.	9.3 (np=15) 9.2 (np=10)
Battiti y Techioli (1992)	QAP & N-k	Tabu Search GA	-	1-512	-	Lineal
Bianchini y Brown (1993)	0-1 ILP	GA	50 variables y 10 restricciones	1-8	8 *transputers* T805.	5.5-7 (np=8)
Calegari *et al.* (1997)	Cobertura de un conjunto	GA	Malla de 287x287 y 49 transmisores	90	10 grupos de 9 workstations Sparc-4	Casi lineal (93% de eficiencia – np=80)
Feo *et al.* (1994)	Máximo conjunto independiente	GRASP		8	Alliant FX/80 MIMD	Casi lineal (7.5 – np=8)
Fiechter (1994)	TSP	Tabu Search	500-10000	1-24	*Transputers*	2 np=2, 9 np=10, 14 np=16, 16-18 np=20
García-López *et al.* (2002)	p-mediana	VNS (tres variantes)	20-100	1, 2, 4, 8		Max: 1.98 np=2, 3.81 np= 4, 6.87 np=8, mayor con problemas pequeños
Liu y Tseng (2000)	Minimización	Minimización de la descomposición del espacio (tres variantes)	100-10000	1, 2, 4, 6	Pentium 166 MHz con conexión Ethernet	Máx: 1.99 (np=2), 4.05 (np=4), 5.94 (np=6)
Martins *et al.* (2002)	Árbol de Steiner	GRASP (tres variantes)		2-16	IBM RS6000 390	Máx: 1.84 np=2, 3.24 np=4, 5.37 np=8, 8.03 np=16
Pardalos *et al.* (1996)	MAX-SAT	GRASP	800-900	5, 10, 15	15 SUN-SPARC 10	Lineal
Pardalos *et al.* (1995b)	QAP	GRASP	<120	64	Kendall Square KRS-1	62
Suh y Van Gutch, (1987)	TSP	GA	100-1000	≤ 90	BBN Buterfly	Casi lineal

Tabla 16: Resumen de las prestaciones de otros algoritmos paralelos

Nota: np: número de procesos, SA: Recocido simulado, GA: algoritmos genéticos, TSP: problema del viajante, QAP: Problema de asignación cuadrática.

Un trabajo que presenta aceleración menor que la lineal es el presentado por Brüngger *et al.* (1999). En él, se propone una librería de búsqueda paralela porta-

ble, ZRAM, para mejorar precisamente la portabilidad de algoritmos paralelos de búsqueda local. En la Tabla 17 se muestran los valores de aceleración obtenidos para diferentes sistemas y número de procesadores. Para los autores, el valor de eficiencia del 35.2% es bastante bueno para la pequeña dimensión del problema, obteniendo mejores resultados en problemas más complejos.

Sistema	Procesadores	Aceleración	Eficiencia
Paragon MP	10	9.4	0.94
	100	35.2	0.35
	150	38.4	0.26
Cenju-3	10	9.4	0.94
	100	34.2	0.34
Gigabooster	7	6.1	0.87

Tabla 17: Aceleración conseguida por el algoritmo paralelo propuesto por Brüngger *et al.* (1999)

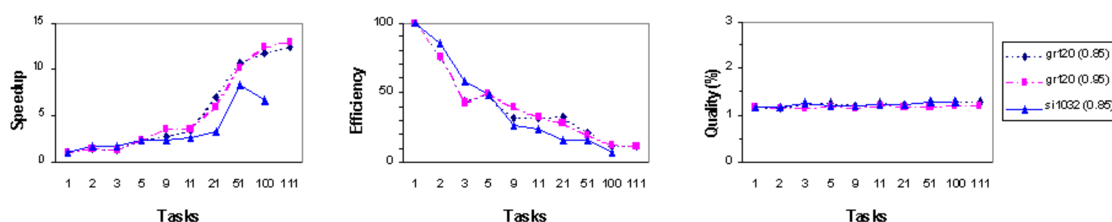


Figura 26: Aceleración, eficiencia y calidad de las soluciones en el algoritmo de Sanvicente y Frausto, (2002)

Sanvicente y Frausto (2002) proponen en su trabajo una metodología para implementar SA en paralelo. Comprobaron la metodología con tres instancias del problema del viajante, simulando un sistema masivamente paralelo con dos ordenadores Silicon Graphics conectados en una red de área local y empleando PVM para la comunicación entre procesos. En su trabajo puede observarse cómo decrece la eficiencia cuando se incrementa el número de tareas asignadas a los procesadores disponibles. Para mostrarlo, los autores han comparado los tiempos para alcanzar soluciones de la misma calidad empleando diferentes patrones de enfriamiento. Nuevamente, se tiene el mismo problema que con el trabajo de Malek *et al.* (1989), ya que no se comparan tiempos de ejecución para alcanzar la misma calidad de las

soluciones sino que la calidad de las soluciones obtenidas mediante el algoritmo paralelo es mejor que la obtenida mediante el algoritmo secuencial. Por otro lado, la eficiencia se mide según el número de tareas y no según el número de procesadores.

Steinhöfel *et al.* (2002) han estimado un valor máximo teórico de aceleración. Este valor es función del camino más largo en un problema con un trabajo y m máquinas, pero la comparación con el problema $F_m/prmu/C_{max}$ resulta compleja.

Fiechter (1994) llevó a cabo otro trabajo en el campo de la implementación en paralelo de la búsqueda tabú mediante redes de *transputers*. Encontró una aceleración casi lineal usando gran número de procesadores y también observó menores valores de eficiencia en problemas grandes al emplear más de diez procesadores.

Otra medida de la eficiencia de la versión en paralelo de metaheurísticas puede encontrarse en Wodecki y Bożejko (2002) y en Bożejko y Wodecki (2004a). Los autores comparan el algoritmo paralelo y el secuencial mediante el valor de *makespan* obtenido después de n iteraciones del algoritmo secuencial y después de n/p del paralelo, siendo p el número de procesadores empleados. Concluyen que su algoritmo paralelo consigue una aceleración mejor que lineal ya que el valor de la función objetivo obtenido así con el algoritmo paralelo es mejor que el obtenido por el algoritmo secuencial.

Otros trabajos sobre la paralelización de metaheurísticas se concentran en mostrar la mejora de la calidad de las soluciones sobre el algoritmo secuencial como, por ejemplo, en Caricato *et al.* (2003), Blesa *et al.* (2001) o Crainic y Gendreau (2002). En Gendreau *et al.* (2001) no tiene sentido considerar la aceleración, ya que el algoritmo está funcionando continuamente. Los autores consideran la cantidad de trabajo realizada por la versión paralela como una medida de su eficiencia. Con esta consideración, Gendreau *et al.* (2001) informa de un crecimiento menor que lineal del número de ejecuciones de la búsqueda tabú con el número de procesadores, debido a la necesidad de interrumpir la aplicación cuando llegan nuevas peticiones al sistema.

Referencia	2	3	4	8	10	16	20
(Adenso-Díaz *et al.*, 2006)					0.38		
(Aiex *et al.*, 2002)	1.04		1.08	0.96		0.69	
(Fiechter, 1994)	1				0.9	0.875	0.85
(García López *et al.*, 2002)	0.99		0.95	0.859			
(Taillard, 1990)	0.96-0.995						
(Taillard, 1991)					0.85		
(Taillard, 1994)	0.75-0.95		0.725-0.825		0.44-0.73		0.22-0.48

Tabla 18: Eficiencia de algunos de los algoritmos estudiados

En la Tabla 18 y la Figura 27 se resumen los valores de eficiencia observados según el número de procesadores de forma que puedan compararse valores.

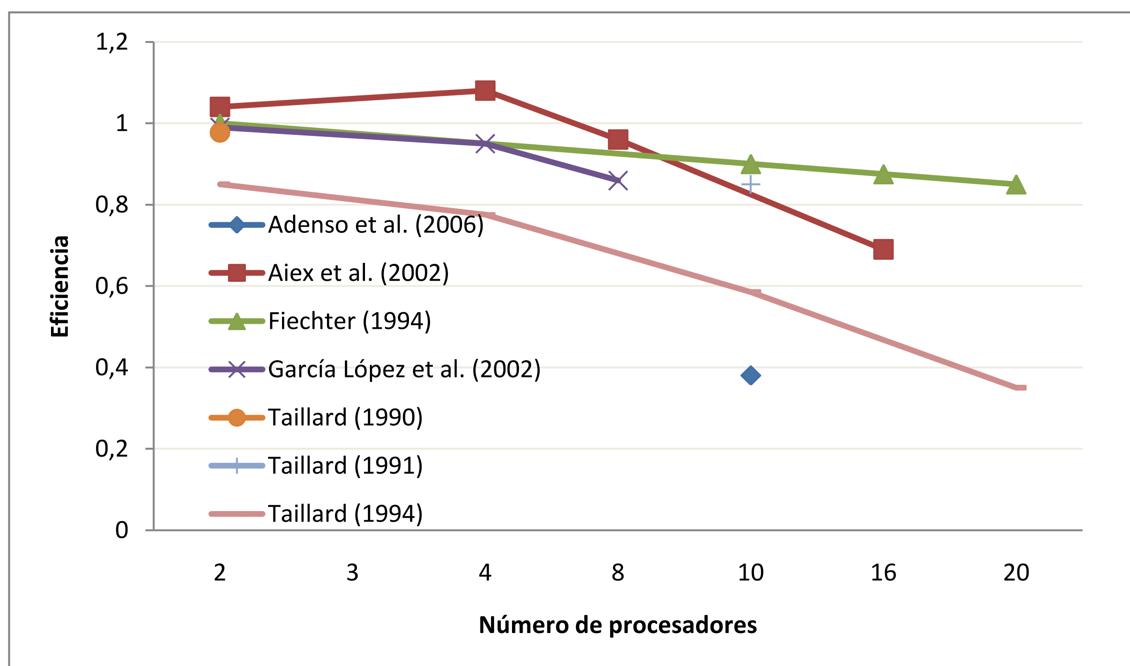


Figura 27: Eficiencia de algunos de los algoritmos estudiados

Hay que tener en cuenta que en gran parte de la literatura consultada sobre la implementación en paralelo de metaheurísticas aplicadas a la resolución de problemas de optimización combinatoria se evita calcular la eficiencia o la aceleración del algoritmo propuesto. En algún caso, esto es debido a no disponer de datos de tiem-

po de ejecución para una calidad dada de las soluciones y en otros casos al empleo de medidas no estándar de la aceleración o de la eficiencia.

3.5 Resumen de aplicaciones en paralelo al problema $F_m/prmu/C_{max}$

En la Tabla 19 se resumen las referencias encontradas sobre aplicación al problema $F_m/prmu/C_{max}$ atendiendo a la taxonomía del capítulo 2.

			B&B		SA			TS			SS		PR		GA
Criterio	**Tipos**	**Subtipos**	**Oc**	**Od**	**WB1g**	**WB1i**	**BW1**	**BW2**	**BW3s**	**BW3a**	**BW4g**	**BW4i**	**BW5g**	**BW5i**	**K**
HW	Memoria compartida		X												
	Memoria distribuida			X	X	X	X	X	X	X	X	X	X	X	X
SW	Hilos				X	X	X	X	X	X	X	X	X	X	
	Sockets														
	Paso de mensajes		MPI	MPI	MPI	MPI	MPI	MPI	MPI	MPI	MPI	MPI	MPI	MPI	MPL*
	HPF														
Estructura del programa	División del trabajo (estrategias)	Iguales	X	X	X	X	X	X							X
		Diferentes							X	X					
	Punto de partida	Único			X	X									
		Diferentes	X	X			X	X	X	X	X	X	X	X	X
	Recorrido (pasadas)	Única	-	-						X	X	X	X	X	X
		Múltiples	-	-	X	X	X	X	X	X					
Comunicación	Papel de los procesos	Maestro/Esclavo		X			X	X	X	X	X	X	X	X	
		Iguales	X		X	X									X
	Colaboración entre procesos	Colaboración	X	X	X		X	X	X	X	X		X		X
		Independientes				X						X		X	
	Sincronización	Síncronos				-		X		X		-		-	X
		Asíncronos	X	X	X	-	X		X	X	X	-	X	-	
	Granularidad	Grano fino													X
		Grano grueso	X	X	X	X	X	X	X	X	X	X	X	X	

Tabla 19: Referencias sobre aplicaciones en paralelo para el problema $F_m|prmu|C_{max}$

*MPL es una extensión del lenguaje C (ver Schwehm, 1993) consistente en objetos para emplear variables distribuidas entre la memoria de los procesadores que componen un sistema masivamente paralelo, además de las funciones necesarias para manejarlas. En B&B no tiene sentido hablar de una o varias pasadas del algoritmo. En el caso de procesos independientes, no tiene sentido hablar de sincronización entre procesos.

La nomenclatura empleada se muestra en la Tabla 20.

	Referencias		Algoritmos
Oc	(Okamoto *et al.*, 1994), mem. compartida	B&B	*Branch & Bound*
Od	(Okamoto *et al.*, 1994), mem. distribuida	SA	*Simulated Annealing*
WB1g	(Wodecki y Bożejko, 2002) global	TS	*Tabu Search*
WB1i	(Wodecki y Bożejko, 2002) independiente	SS	*Scatter Search*
BW1	(Bożejko y Wodecki , 2004a)	PR	*Path Relinking*
BW2	(Bożejko y Wodecki, 2002)	GA	*Genetic Algorithms*
BW3s	(Bożejko y Wodecki, 2004b) síncrona		
BW3b	(Bożejko y Wodecki, 2004b) asíncrona		
BW4g	(Bożejko y Wodecki, 2008a) global		
BW4i	(Bożejko y Wodecki, 2008a) independiente		
BW5g	(Bożejko y Wodecki, 2008b) global		
BW5i	(Bożejko y Wodecki, 2008b) independiente		
K	(Kohlmorgen *et al.*, 1999)		

Tabla 20: Abreviaturas empleadas en la Tabla 19

Sobre las aplicaciones en paralelo del recocido simulado, las que han obtenido mejores resultados son las que emplean procesos que intercambian información en lugar de emplear procesos independientes. Se debe tener en cuenta que un excesivo número de intercambios de información entre procesos puede empeorar la eficiencia del algoritmo cuando se trata de algoritmos de grano grueso, que suelen ser la mayor parte de los revisados.

En casi todas las referencias, incluso en Roussel-Ragot y Dreyfus (1990) que emplea una red de *transputers* – una arquitectura masivamente paralela – se emplean esquemas de comunicación maestro-esclavo, siendo el proceso maestro el encargado de coordinar la función de enfriamiento y las temperaturas a las que funcionará cada uno de los procesos esclavos. Se ha observado que, además de mantener una lista con soluciones y su valor de la función objetivo, el proceso principal o maestro suele guardar también la temperatura a la que se obtuvo dicha solución.

En Roussel-Ragot y Dreyfus (1990) y en Steinhöfel *et al.* (2002) se realizan estimaciones sobre la aceleración que se puede conseguir con el algoritmo paralelo basándose en una estimación de la distribución de probabilidad del valor la función objetivo de las soluciones encontradas.

La mayor parte de las referencias revisadas sobre búsqueda tabú emplean una arquitectura maestro-esclavo. En este caso, se encuentran referencias de intercambios de información tanto síncronos como asíncronos y se vuelve a insistir que una disminución en el número de puntos de sincronización entre procesos suele mejorar la eficiencia del algoritmo paralelo. En el caso del trabajo de Schulze y Fahle (1999) se recuerda este extremo empleando, a diferencia del resto, una arquitectura de procesos iguales e independientes. También cabe destacar que, en el caso de arquitecturas maestro-esclavo, el proceso maestro no sólo guarda la lista tabú sino también la mejor solución encontrada hasta el momento o un conjunto con cardinalidad limitada con las mejores soluciones encontradas.

En los algoritmos voraces revisados se ha mencionado de nuevo, que una reducción del número de comunicaciones entre procesos mejora la eficiencia del algoritmo. En este caso es más acusado ya que este tipo de algoritmo sólo requiere el intercambio de la mejor solución encontrada hasta el momento.

En el caso de algoritmos genéticos en paralelo nos volvemos a encontrar con únicamente una referencia de su aplicación al problema $F_m/prmu/C_{max}$, (Kohlmorgen *et al.,* 1999). En este trabajo se presenta un estudio de varias aplicaciones de los algoritmos genéticos paralelos, entre ellas al problema de secuenciación en flujo uniforme, destacando las ventajas del algoritmo paralelo sobre el secuencial en cuanto a la aceleración del mismo y en cuanto a la calidad de las soluciones al diversificar la población en tantas subpoblaciones como procesadores disponibles en el sistema. En la literatura existe una enorme cantidad de trabajos sobre aplicaciones diversas de este tipo de algoritmos paralelos a diferentes problemas, incluyendo otras variantes de problemas de secuenciación. Algunos autores han destacado la facilidad de la implementación en paralelo de algunas fases de los algoritmos genéticos. La técnica más empleada es la de enviar diferentes poblaciones a diferentes procesadores, para que evolucionen por separado intercambiando los resul-

tados de la evolución cada cierto número de iteraciones, bien de forma síncrona o asíncrona.

Sobre las versiones en paralelo de VNS revisadas, se observa más diversidad de opciones de paralelización que en los apartados anteriores, donde dominaban los esquemas tipo maestro-esclavo. También se han visto mejoras a la implementación de VNS cuando se emplea en paralelo combinada con GRASP. Un aspecto a destacar es que, en este tipo de algoritmo, predominan las aplicaciones basadas en procesos iguales frente a aplicaciones tipo maestro-esclavo.

3.6 Conclusiones

Se ha presentado en este capítulo una revisión bibliográfica de los procedimientos aplicados a problemas de optimización para su resolución exacta y aproximada. De entre ellos se ha prestado especial interés a aquellos que se han aplicado con éxito al problema $F_m/prmu/C_{max}$ o a problemas similares en lo que respecta a su codificación o con objetivos distintos.

Es necesario destacar que son escasas las referencias existentes sobre el problema objeto de estudio. Hasta el momento de redactar este documento, sólo conocemos siete referencias sobre aplicaciones en paralelo para la resolución del problema $F_m/prmu/C_{max}$, la mayoría de ellas debidas a dos autores. Sin embargo, sí se han encontrado aplicaciones al problema de secuenciación en flujo uniforme con otras funciones objetivo como la minimización de las sumas de los tiempos de proceso o variantes con fecha de entrega o máquinas no disponibles. Otras referencias incluyen aplicaciones a entornos de flujo híbrido o también de flujo taller.

La implementación en paralelo de procedimientos de búsqueda local es más compleja que B&B, SA o GA por las dificultades que plantea la división del espacio de soluciones (Roucairol, 1996) por lo que en la literatura se encuentran muchos menos desarrollos en paralelo de este tipo de métodos. En algunas de las variantes analizadas no se han encontrado referencias de su aplicación en paralelo al problema $F_m/prmu/C_{max}$. Es el caso de los algoritmos voraces, la búsqueda de vecindad variable, la búsqueda local iterativa, la optimización mediante colonias de hormigas o los algoritmos de estimación de la distribución.

Otra de las dificultades es la comparación de las prestaciones de los algoritmos revisados. En general, no se aplican los mismos criterios para exponer sus prestaciones. Incluso cuando se habla de los valores de aceleración o de eficiencia logrados con el algoritmo paralelo (los criterios más usuales) se han calculado siguiendo diferentes criterios de evaluación. Mientras en todos los trabajos se está de acuerdo en que el valor de aceleración debe medir la relación entre el tiempo de ejecución del algoritmo paralelo frente al del algoritmo secuencial para lograr soluciones con el mismo valor de la función objetivo, en muchos de ellos se recurre únicamente a presentar el valor de la función objetivo alcanzado empleando el mismo número total de iteraciones con el algoritmo secuencial y con el algoritmo paralelo (ver Wodecki y Bożejko, 2002, Bożejko y Wodecki, 2002, Bożejko y Wodecki, 2008a, Bożejko y Wodecki, 2008b). Esto implica una reducción en el número de iteraciones que realiza cada uno de los procesos que componen el algoritmo paralelo. Aun así, es interesante observar que para el mismo número total de iteraciones, la calidad de las soluciones obtenidas mediante el algoritmo paralelo empleando procesos independientes es mejor que la obtenida mediante el algoritmo secuencial en la mayoría de los trabajos revisados. En cambio en otro trabajo de los mismos autores, se muestra el efecto esperado, soluciones de peor calidad al emplear procesos independientes que realizan cada uno menos iteraciones que el algoritmo secuencial. Por tanto, se puede deducir que el empleo de diferentes puntos de partida en un algoritmo paralelo da mejor resultado en cuanto a calidad de las soluciones obtenidas que el número de iteraciones.

Sobre los procedimientos empleados, los más comunes en la actualidad emplean arquitecturas con memoria distribuida (ver Wodecki y Bożejko, 2002, Bożejko y Wodecki, 2008a, Bożejko y Wodecki, 2008b). En algunos de los artículos se advierte de que un excesivo número de puntos de sincronización entre los procesos puede degradar la aceleración del algoritmo. Aunque esto daría ventaja al uso de procesos independientes, los algoritmos con procesos en cooperación suelen obtener mejores resultados en cuanto a tiempo de ejecución y calidad de soluciones.

Algunos de los procedimientos descritos se han implementado sobre sistemas paralelos ideales, como en el caso del sistema PRAM, (Steinhöfel *et al.,* 2002), o en un solo ordenador monoprocesador con objeto de abstraer la arquitectura física del siste-

ma de la arquitectura lógica (Adenso-Díaz *et al.*, 2006). Otros estudios presentan estimaciones de la aceleración que es posible alcanzar con un algoritmo paralelo (Steinhöfel *et al.*, 2002).

En general, la mayoría de los trabajos recurren a una arquitectura de comunicación maestro-esclavo (ver Okamoto *et al.*, 1994, Okamoto *et al.*, 1995, Bożejko y Wodecki, 2002, Bożejko y Wodecki, 2004b, Bożejko y Wodecki, 2008a, Bożejko y Wodecki, 2008b), sobre todo porque alguno de los procedimientos requiere de información común para el desarrollo de la búsqueda de soluciones. Al disponer de un proceso que contiene información sobre la búsqueda, se suele incluir en el mismo, bien la mejor solución encontrada hasta el momento o bien una lista de soluciones visitadas durante la búsqueda o incluso alguna de las características del proceso. Por ejemplo, en algunas versiones de SA se guarda un conjunto con las mejores soluciones encontradas hasta el momento junto con la temperatura a la que se encontraron (ver Wodecki y Bożejko, 2002). También se han empleado con éxito arquitecturas de procesos iguales o *peer-to-peer*, fundamentalmente en sistemas masivamente paralelos o en aquellos que emplean *transputers* (Kohlmorgen *et al.*, 1999). Las redes *transputers* tienen escasa difusión, por lo que las conclusiones sobre su funcionamiento son más complejas de extrapolar a los sistemas informáticos actuales que se pueden encontrar en las empresas de nuestro entorno.

Una vez estudiadas las estrategias más empleadas al problema objeto de estudio, como son la arquitectura de la comunicación tipo maestro esclavo, el empleo de librerías de paso de mensajes en equipos con memoria distribuida, la sincronización entre procesos o el empleo de diferentes puntos de partida en la exploración del espacio de soluciones, se tratarán de implementar en forma de algoritmo paralelo tratando de encontrar la máxima eficiencia del mismo.

4 CLM-paralelo (CLMp)

4.1 Introducción

En este capítulo se proponen dos versiones de un algoritmo paralelo basado en la metaheurística CLM de Ghosh y Sierksma (2002) para el problema objeto de estudio. Las características de adaptabilidad de esta heurística hacen que las conclusiones que pueden extraerse del estudio de su comportamiento en paralelo sean extrapolables a otras heurísticas de búsqueda local. Esto es debido a que, variando los parámetros y condiciones empleadas en CLM, se puede tener un algoritmo de búsqueda local exhaustiva, de máximo descenso, de descenso aleatorio o recocido simulado. En este capítulo nos centramos en el diseño de algoritmos paralelos basados en CLM, dejando para el capítulo 5 el ajuste de sus parámetros.

La primera de las versiones del algoritmo paralelo basado en CLM es de grano grueso y ha servido para ajustar algunas de las características del algoritmo CLM original de Ghosh y Sierksma (2002) tanto al problema objeto de estudio como a su implementación en paralelo. Para esta versión, se ha llevado a cabo una primera experimentación ajustando un parámetro cada vez. Llamamos a esta versión CLMpg.

La segunda es una versión que hemos denominado de grano fino, que se diferencia de la anterior en que se incrementan el número de veces que intercambian información entre sí los procesos en cada iteración. Se ha aprovechado la experiencia de la versión anterior para un mejor ajuste de parámetros mediante una experimentación más elaborada. Llamamos a esta versión CLMpf.

Se verificará si estos mayores intercambios de información mejoran o empeoran la eficiencia del algoritmo paralelo, ya que la mayor comunicación entre procesos lleva a un mayor conocimiento de la exploración que están realizando el resto de procesos y por tanto puede llevar a incrementar la aceleración del algoritmo y consecuentemente su eficiencia. Por otro lado el aumento de la comunicación lleva a mayores tiempos de espera y por tanto a una disminución de la eficiencia.

4.2 Descripción de la metaheurística CLM

Complete Local Search with Memory, de Ghosh y Sierksma (2002), es una metaheurística de búsqueda local que mantiene similitudes con la búsqueda tabú (TS) y con el recocido simulado (SA). Dentro de la clasificación vista en el capítulo 3, esta metaheurística entra dentro de los métodos de trayectoria. CLM presenta ventajas sobre otros métodos de trayectoria, ya que, salvo la búsqueda tabú o sus híbridos, ningún otro algoritmo de búsqueda mantiene una memoria de las soluciones exploradas. Los métodos con memoria ofrecen normalmente más calidad que los métodos sin memoria en un amplio rango de aplicaciones.

En el caso de CLM se mantienen tres conjuntos de soluciones con lo que evita volver a examinar soluciones en la exploración del espacio de soluciones. En su trabajo, Ghosh y Sierksma (2002) aluden a que ningún método de búsqueda tiene en cuenta el hecho de que es posible volver a examinar una solución por un camino diferente, partiendo de la misma solución, debido a las estructuras de vecindad que se crean en la exploración del espacio de soluciones.

En el pseudocódigo mostrado en la Figura 28, S es una solución del problema, S_0 es la solución de partida, S_{neig} una solución vecina de S, $chosen$ es un contador del número de soluciones de la lista LIVE seleccionadas en cada iteración, k es el número máximo de soluciones que se seleccionan de LIVE en cada iteración y τ es el umbral de aceptación de soluciones.

CLM:
Paso 1.Almacenar S_0 en la lista LIVE
Paso 2.Vaciar la lista DEAD
Paso 3.**Mientras** (condición de parada = NO)
  3.1. Vaciar el conjunto NEWGEN
  3.2. $chosen$:= 0
  3.3. **Mientras** ($chosen < k$ y LIVE $\neq \emptyset$)
    3.3.1. Tomar S de LIVE
    3.3.2. $chosen$:= $chosen$ +1
    3.3.3. LIVE := LIVE $\setminus \{ S \}$, DEAD := DEAD $\cup \{ S \}$
    3.3.4. Generar soluciones vecinas S_{neig} de S tales que su función objetivo sea mejor que τ
    3.3.5. **Para cada** vecina S_{neig} de S
      3.3.5.1. **Si** ($S_{neig} \notin$ {LIVE, DEAD, NEWGEN})
        3.3.5.1.1. **Si** (hay suficiente memoria) NEWGEN := NEWGEN $\cup \{ S_{neig} \}$
        3.3.5.1.2. **Si no**
          3.3.5.1.2.1. $\forall S \in$ NEWGEN, NEWGEN:= NEWGEN $\setminus \{ S \}$, LIVE := LIVE $\cup \{ S \}$
          3.3.5.1.2.2. Postprocesar
  3.4. $\forall S \in$ NEWGEN, NEWGEN:= NEWGEN $\setminus \{ S \}$, LIVE := LIVE $\cup \{ S \}$
Paso 4.Devolver la mejor solución de DEAD

Función Postprocesar:
Paso 1.$\forall S \in$ LIVE
  1.1. LIVE := LIVE $\setminus \{ S \}$
  1.2. Obtener un óptimo local S_l mediante búsqueda local a partir de S
  1.3. Almacenar S_l en la lista DEAD

Figura 28: Pseudocódigo de CLM (Ghosh y Sierksma, 2002)

EL algoritmo emplea varias listas con las soluciones que se van obteniendo: LIVE, NEWGEN y DEAD (ver Figura 29) de forma que se evita repetir zonas del espacio de soluciones que ya han sido exploradas y, además, se tiene una memoria de las zonas del espacio con mejores los valores de la función objetivo.

La lista LIVE almacena las soluciones cuya vecindad se va a explorar en sucesivas iteraciones. La lista DEAD almacena las soluciones que ya han sido exploradas. La lista NEWGEN es un almacenamiento temporal para las nuevas soluciones que están siendo generadas por la heurística durante la iteración actual.

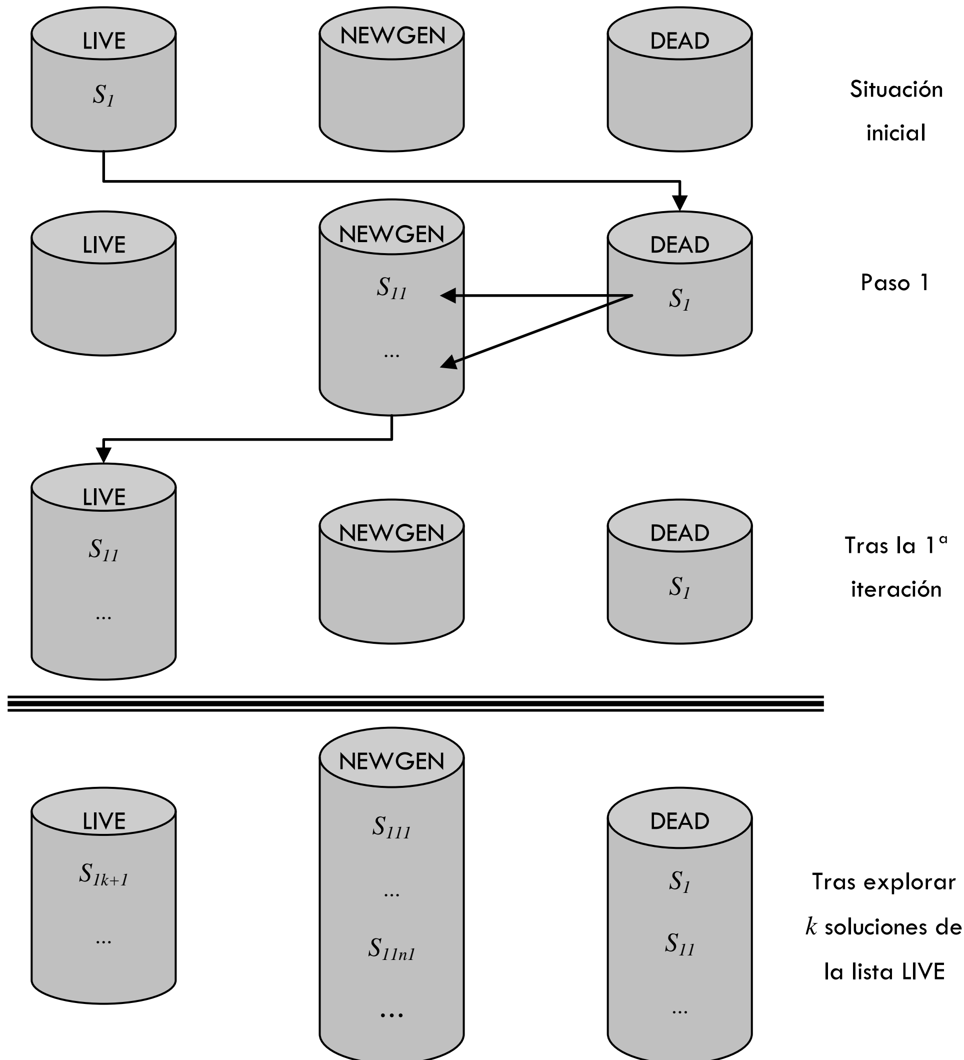


Figura 29: Esquema de funcionamiento interno del algoritmo CLM

La metaheurística comienza con una solución de entrada S_1 almacenada en LIVE. Las listas DEAD y NEWGEN están vacías inicialmente. En cada iteración de CLM, se toma una solución de LIVE, se generan todas sus soluciones vecinas y la solución explorada se pasa a DEAD. Aquellas soluciones recién generadas cuya función objetivo es mejor que un umbral τ, si no están en ninguna de las tres listas, se pasan a NEWGEN. Cuando termina la iteración, se pasan todas las soluciones de NEWGEN a LIVE para poder explorarlas en la siguiente iteración. A partir de la segunda iteración, en la que ya existe más de una solución en LIVE, se toman para explorar sus vecinas hasta K soluciones del conjunto LIVE, o hasta que éste esté vacío. El

algoritmo continúa hasta que se cumpla la condición de parada o se haya explorado completamente el espacio de soluciones.

Como ocurre en el recocido simulado, se emplea un parámetro umbral τ para admitir soluciones de peor calidad que las mejores encontradas hasta el momento, con lo que se trata de evitar el estancamiento de la solución en mínimos locales, en una fase de diversificación. Este umbral τ va disminuyendo con el número de iteraciones para no aceptar soluciones que están muy alejadas de la mejor solución cuando ésta ya ha alcanzado una calidad suficiente, por ejemplo, cuando disminuye el ritmo de mejora o cuando se han realizado más de cierto número de iteraciones.

CLM es realmente una plantilla con varias cuestiones por decidir, lo que hará que el método tenga similitudes con una u otra metaheurística. Estas cuestiones son la condición de parada, el total de soluciones almacenadas o tamaño máximo de la memoria, la estructura de vecindad, cómo decidir cuál de las soluciones de la lista LIVE es la siguiente en ser explorada, el valor de K y el valor del umbral de aceptación τ. Cada uno de estos aspectos es discutido a continuación.

a) Condición de parada.

Ésta puede ser:

- Cuando la lista LIVE está vacía al comienzo de una iteración.
- Cuando se alcanza un número de iteraciones predefinido.
- Cuando no se alcanza mejora en el valor de la función objetivo tras un número de iteraciones predefinido.
- Cuando se alcanza un óptimo local.

Tanto el tiempo de ejecución como la calidad de las soluciones son mayores en el primer caso y menores en el último.

b) Número total de soluciones almacenadas

Según Ghosh y Sierksma (2002), éste es el tamaño total de la memoria sumando todas las listas de soluciones. Cuanto mayor sea este valor, el algoritmo continuará

explorando durante más tiempo y serán mayores las posibilidades de encontrar soluciones de mayor calidad.

c) Estructura de la vecindad

Esta es la regla mediante la que se decide si una solución está cerca de otra de alguna forma y un método para generar una solución vecina de otra. Esta elección tendrá también influencia en el tiempo de exploración y en la cantidad de memoria utilizada.

d) Estrategia de selección de soluciones

Debe decidirse cómo se eligen las soluciones de LIVE cuya vecindad va a ser explorada. Ghosh y Sierksma (2002) proponen tres alternativas:

- Elegir la solución con la mejor función objetivo.
- Elegir la solución que permita la mayor mejora en la exploración en cada paso, lo que implica evaluar al menos alguna solución vecina de cada una de las que compone la lista LIVE en cada paso para seleccionar aquellas que llevan a un mejor avance en la función objetivo.
- Elegir la solución al azar.

e) Valor de K

K es el número de soluciones que se toman de LIVE en cada iteración. Conforme aumenta este valor, habrá más tiempo de exploración y, por tanto, mayor posibilidad de mejora en cada iteración, pero también se incrementará el esfuerzo computacional en cada iteración.

f) Valor de τ

τ es el valor del umbral que permite que se almacenen soluciones exploradas que no implican una mejora. Puede ser alto o bajo, lejos o cerca del mejor valor de la función objetivo, permitiendo que se admitan muchas o pocas soluciones. También puede usarse un valor dependiente del número de iteraciones, a la manera del recocido simulado.

Como ya se ha comentado, en función de la elección de los anteriores, el CLM se asemeja a otros existentes, como se muestra en la Tabla 21.

Método	Condición de parada	Selección de soluciones	K	Umbral τ
Búsqueda exhaustiva	LIVE vacío			Infinito
Búsqueda local con máximo descenso	Óptimo local	Mejor función objetivo (f.o.)	1	f.o. de la solución explorada
Búsqueda local con descenso aleatorio	Óptimo local	Mejor f.o.		f.o. de la solución explorada
Recocido simulado	Nº de iteraciones prefijado o nº de iteraciones sin mejora de la f.o.	Al azar entre las generadas en la última iteración	1	En función del nº de iteraciones y del mejor valor de la f.o. en ese instante

Tabla 21: Resumen de similitudes de CLM con otras metaheurísticas en función de las decisiones sobre los parámetros

La primera regla de parada (LIVE vacío al comienzo de una iteración y un valor infinito del umbral) da como resultado una búsqueda exhaustiva, siempre que el grafo de soluciones adyacentes sea conexo.

La última regla de parada (cuando se alcanza un mínimo local) eligiendo las soluciones con mejor función objetivo, $K = 1$ y el valor del umbral τ como el de la función objetivo de la solución que está siendo explorada da como resultado un algoritmo genérico de búsqueda local con el máximo descenso.

Con la cuarta regla de parada (cuando se alcanza un mínimo local) y la tercera regla de exploración (selección de soluciones al azar de la lista LIVE), la solución con la mayor mejora en la función objetivo, τ como el valor de la función objetivo de la solución que está siendo explorada da como resultado un algoritmo genérico de búsqueda local con descenso aleatorio.

Con las reglas de parada segunda o tercera, $K = 1$, el umbral en función del número de iteraciones y del valor de la mejor función objetivo en ese momento, y usando como regla de exploración una que seleccione al azar soluciones entre aquellas

generadas en la última iteración, da como resultado una heurística de recocido simulado.

En cuanto a aplicaciones de CLM a la resolución de problemas de optimización, se encuentran los trabajos de los ya citados Ghosh y Sierksma (2002) y los de Ghosh (2003), Framiñán y Schuster (2006) y Framiñán y Pastor (2008). En Ghosh y Sierksma (2002) se aplica CLM al problema del viajante (TSP) y al de la suma de un subconjunto (SSP). Según los autores, se seleccionaron estos dos problemas para probar su algoritmo porque en el TSP, la búsqueda local no ha dado buenos resultados mientras que en el SSP los resultados al emplear búsqueda local son mejores que con otras metaheurísticas. El método es comparado con TS y, según sus resultados, ofrece mejor calidad de soluciones y tiempo de ejecución en una serie de instancias aleatorias del TSP y soluciones de similar calidad en el caso del SSP, siendo además el tiempo de ejecución inferior al empleado por la búsqueda tabú.

Ghosh (2003) describe una aplicación de CLM a problemas de localización de instalaciones sin restricciones de capacidad. Otro problema afrontado mediante esta heurística ha sido el problema de secuenciación en entorno taller sin esperas, por Framiñán y Schuster (2006). Finalmente, como CLM es un algoritmo que puede asimilarse a otras metaheurísticas mediante el ajuste de parámetros y decisiones, el desarrollo del algoritmo paralelo se podría emplear para implementar versiones en paralelo de otras metaheurísticas o versiones de métodos híbridos, por ejemplo con otros algoritmos exactos de resolución, tal como se propone en Framiñán y Pastor (2008).

En los apartados siguientes se describen las implementaciones de CLM realizadas.

4.3 Algoritmo paralelo de grano grueso: CLMpg

En el primer capítulo se distinguieron una serie de estrategias mediante las que desarrollar versiones en paralelo de algoritmos de resolución de problemas. Dentro de la taxonomía propuesta por Flynn (1966), la primera versión de nuestro algoritmo paralelo se puede clasificar dentro del tipo SIMD, ya que se separa en múltiples procesos que ejecutan las mismas funciones sobre conjuntos de datos (soluciones) diferentes. Se trata además, de un algoritmo de grano grueso, tal como aparece en

la clasificación de Cung *et al.* (2001), ya que la carga de comunicación es muy inferior a la carga de computación.

En nuestro caso, vamos a emplear una versión de algoritmo paralelo cercana a la versión maestro-esclavo en la que llamaremos proceso jefe al encargado de coordinar al resto, de acuerdo con la notación "*master-worker*" empleada en trabajos como el de Sittig *et al.* (1991), puesto que realiza también parte del trabajo de búsqueda, no limitándose únicamente a coordinar al resto de procesos.

Los procesos trabajadores funcionan como procesos independientes, y sólo intercambian información a través de un proceso central, que llamaremos jefe. A continuación se describen las principales tareas del proceso jefe y de los procesos trabajadores.

4.3.1 Proceso Jefe

La misión principal del proceso jefe es la de mantener una lista central de soluciones o *CPOOL*, ya que el sistema no dispone de memoria compartida. Esta lista contiene las mejores soluciones que ha enviado cada proceso esclavo junto con la estrategia de búsqueda empleada. De esta forma, al recibir las soluciones enviadas por los procesos esclavos, puede devolverles tanto una nueva secuencia inicial como una estrategia de búsqueda. Este ha sido el camino seguido por autores como Attanasio *et al.* (2004), Le Bouthillier y Crainic (2005), Sittig *et al.* (1991) o Niar y Freville (1996). Un resumen de las tareas del proceso jefe se muestra en la Figura 30.

La fase de inicialización consiste en la lectura de la instancia del problema y de los parámetros globales {$npool$, MAX_GLOB_IT, B}, los parámetros de búsqueda iniciales {MAX_MEM, MAX_IT, K, α, β} y se obtiene el número de procesos trabajadores según el número total de procesos menos uno.

Para evitar que la disponibilidad de más procesos implique menos iteraciones por parte de cada proceso trabajador, se ha tomado un número máximo de iteraciones globales, MAX_GLOB_IT proporcional al número de procesos trabajadores disponibles.

Proceso jefe:

Paso 1. Inicialización

Paso 2. **Para cada** proceso trabajador

2.1. Generar solución inicial y estrategia de búsqueda

2.2. Almacenar la solución y la estrategia en POOL

2.3. Enviar a cada proceso su solución junto con su estrategia

Paso 3. **Durante** MAX_GLOB_IT iteraciones

3.1. **Cada vez** que un trabajador devuelva sus B mejores soluciones

3.1.1. Almacenar en POOL las soluciones recibidas junto con la estrategia empleada.

3.1.2. Generar nueva solución y nueva estrategia para ese proceso

3.1.3. Enviar a ese proceso la solución generada y la nueva estrategia de búsqueda

Paso 4. Enviar la señal de parada a los procesos trabajadores

Paso 5. Finalizar el proceso y almacenar los resultados en un archivo

Figura 30: Pseudocódigo del proceso jefe

El número de iteraciones que realiza cada proceso en el algoritmo paralelo, al igual que han hecho autores como Attanasio *et al.* (2004) será proporcional al número de procesos empleado, de forma que cada proceso individual en el algoritmo paralelo ejecuta el mismo número de iteraciones que el algoritmo secuencial.

Se describen ahora con más detalle, algunos de los pasos anteriores.

4.3.1.1 Generación de soluciones iniciales

En primer lugar, el proceso jefe genera una solución inicial distinta para cada uno de los procesos esclavos, junto con un conjunto de parámetros de búsqueda.

En la versión actual del algoritmo se generan de forma completamente aleatoria diferentes soluciones de partida para cada proceso trabajador. El empleo del *pool* central de soluciones, permite generar una solución distinta para cada proceso, ya que se comprueba al generar cada solución si está en esa lista. En caso de estar, se descarta y se genera otra.

El hecho de que cada proceso comience el recorrido de exploración desde soluciones distintas junto con el empleo de memoria, evita que durante el proceso de exploración se vuelvan a explorar soluciones ya estudiadas por algún otro proceso. El hecho de que cada proceso trabajador compruebe si la solución recién generada se encuentra entre las que están en las listas de soluciones LIVE, DEAD o NEWGEN, evita la doble exploración de soluciones.

4.3.1.2 Estrategia de búsqueda

Como se indicó en el apartado 4.2, la estrategia de búsqueda de CLM se define por los siguientes elementos:

- La condición de parada
- El total de soluciones almacenadas o tamaño máximo de la memoria
- La estructura de vecindad
- Cómo decidir cuál de las soluciones de la lista LIVE es la siguiente en ser explorada
- El valor de K
- El valor del umbral de aceptación de soluciones τ.

Estos elementos se describen con detalle en el apartado dedicado a los procesos trabajadores.

4.3.1.3 Almacenamiento en la lista *CPOOL*

Las soluciones junto con los parámetros de búsqueda, después de generarse, se guardan ordenadas según *makespan* en la lista C*POOL*, de forma similar a la propuesta por Le Bouthillier y Crainic (2005), como estructuras de datos. A continuación se envía una combinación de solución y parámetros de búsqueda a cada proceso trabajador. Esto implica que, a mayor número de procesos, mayor número de soluciones iniciales. Si se genera una solución que ya está en la lista, se descarta y se vuelve a generar una nueva de forma aleatoria, como se propone en Schulze y Fahle (1999).

Una vez completado el primer envío de soluciones, el proceso jefe queda a la espera de la respuesta por parte de los procesos trabajadores. Cuando éstos reenvían

el resultado de la exploración, las B mejores soluciones encontradas se almacenan en *CPOOL* junto con los parámetros empleados, devolviéndoles una solución nueva junto con nuevos parámetros de búsqueda hasta completar el total de iteraciones globales MAX_GLOB_IT prefijado. En el apartado 4.4.1.5 se explica la selección de nuevas soluciones para reenviar a los procesos trabajadores.

4.3.1.4 Recepción de soluciones

El intercambio de información entre procesos se realiza de forma similar a la propuesta por Toulouse *et al.* (1999). Las soluciones almacenadas en *CPOOL* son las recogidas de los procesos trabajadores, mientras que, al llegar al máximo número de soluciones almacenadas, se van descartando las de peor calidad.

Los procesos trabajadores, una vez realizada su exploración, devuelven las B mejores soluciones encontradas al proceso jefe, de forma similar a como se propone en el trabajo de Niar y Freville (1996). El proceso jefe recibe las B soluciones enviadas por el proceso que termina primero, almacenándolas ordenadas de menor a mayor valor de *makespan* en la lista *CPOOL*. Esta operación se repite durante MAX_GLOB_IT iteraciones globales, como proponen Niar y Freville (1996).

Si se supera la capacidad $npool$, de esta lista, se eliminan las soluciones de peor calidad, como proponen Schulze y Fahle (1999), de forma que no aumente de forma ilimitada el tamaño de esta lista y se vayan descartando poco a poco las soluciones de peor calidad una vez avance el algoritmo.

	Solución						
Tipo de dato	entero	n enteros	entero	doble	entero	entero	entero
Contenido	proceso	secuencia	trabajos	*makespan*	memoria	k	iteraciones

Figura 31: Estructura de las soluciones almacenadas en la memoria

Si las soluciones recibidas no están repetidas en la lista, se guardan ordenadas, incluyendo el número del proceso que las ha generado y los parámetros que empleó dicho proceso en la exploración. La estructura de las soluciones junto con los

parámetros o nodos guardados es, indicando el tipo de dato empleado la que se muestra en la Figura 31.

4.3.1.5 Generación de nuevas soluciones

En el punto 3.1.2. del pseudocódigo anterior (Figura 30), el proceso jefe genera una nueva solución de partida para el proceso trabajador que acaba de enviar sus resultados, junto con sus parámetros de búsqueda. El método empleado para seleccionar una de las soluciones del *pool* central es el método SMobil propuesto por Crainic y Gendreau (2002). La solución se escoge de *CPOOL* con una probabilidad proporcional a su posición en la lista y al número de soluciones que se han añadido a la misma detrás de la solución de que se trate, como otra medida más de su calidad. Siendo S_i una solución almacenada en *CPOOL*, ordenada según el valor de *makespan* correspondiente, la probabilidad de que sea seleccionada de la lista viene dada por la expresión (16):

$$P(S_i) = \frac{m_i + npool + 1 - i}{\sum_i^{npool}(m_i + i)} \quad (16)$$

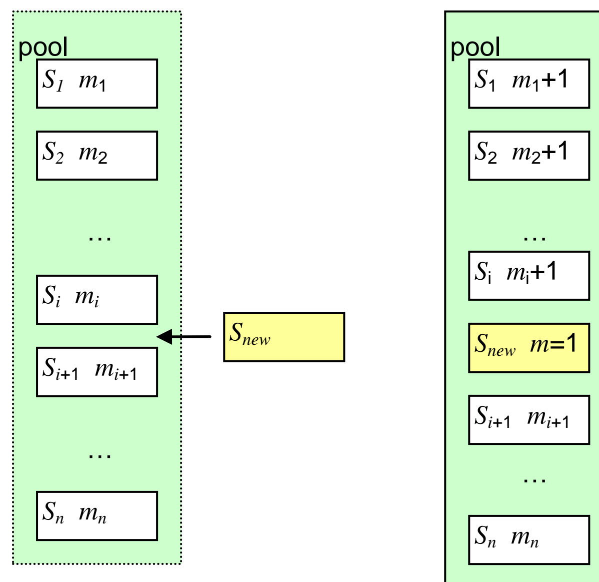


Figura 32: Esquema de inclusión de soluciones en el *pool* central y modificación de la movilidad

La primera solución de la lista es aquella con mejor valor de *makespan*. El número de soluciones almacenadas es $npool$. m_i es la movilidad de la solución, tal como se define en Crainic y Gendreau (2002). El valor inicial de m_i es cero para todas las soluciones. Cuando una solución S_{new} se añade a *CPOOL*, todas las soluciones con mejor valor de *makespan* que S_{new} incrementan en uno su movilidad m_i, ver Figura 32, mientras que el resto permanece sin modificar. De este modo, no solo tienen más probabilidad de ser seleccionadas aquellas soluciones de mejor calidad, sino también aquellas que han permanecido más tiempo en las primeras posiciones de la lista.

Esta política de selección de soluciones es la que ofrece mejor resultado según Crainic y Gendreau (2002) de entre las cinco propuestas por estos autores cuando se incrementa el número de procesos. Las otras propuestas se basan más directamente en la calidad de las soluciones. Estas propuestas son:

- Sbest: Consiste en seleccionar siempre la mejor de las soluciones de la lista central o *CPOOL*. Según Crainic y Gendreau (2002) esta estrategia puede llevar a una convergencia prematura de las soluciones.
- Sprob: Selecciona las soluciones de *CPOOL* según una función de probabilidad basada en la posición en la lista, tal como se muestra en la expresión (17):

$$P(S_i) = \frac{n+1-i}{\sum_1^n i} \qquad (17)$$

- SHhigh y SHlow son diseños más complejos que tratan de tener en cuenta las características ideales de la mejor solución al problema de estudio otorgando más probabilidad de ser seleccionadas a aquellas con mayor número de componentes iguales a los de una solución ideal construida como un promedio de las mejores soluciones.

Los parámetros de búsqueda se adaptan de forma dinámica, tal como proponen Niar y Freville (1996). En nuestro algoritmo paralelo varían el tamaño máximo de

memoria y el número de iteraciones en los procesos trabajadores en función de una métrica basada en la posición. La distancia entre dos soluciones se calcula según la expresión (18):

$$dist = \sum_{j=1}^{n} \left| \sigma_j^1 - \sigma_j^2 \right| \qquad (18)$$

Donde σ_j^1 es la posición que ocupa el trabajo j en la solución 1 y σ_j^2 es la posición que ocupa el trabajo j en la solución 2. n es la dimensión de las soluciones. Se ha empleado esta métrica por la correlación existente entre la proximidad de las soluciones según esta métrica y la función objetivo (Reeves, 1999).

Si la solución seleccionada para enviar a ese trabajador está cerca de la última enviada por dicho proceso, se aumentan los parámetros que controlan la condición de parada, de forma que aumenten las posibilidades de salir de zonas poco prometedoras del espacio de soluciones. Estos parámetros son el tamaño máximo de las listas, MAX_MEM, y el número máximo de iteraciones sin mejora, MAX_IT y el aumento consiste en duplicar su valor.

Si no se ha obtenido mejora en varias iteraciones, se devolverá una solución aleatoria de entre las más prometedoras encontradas hasta la fecha.

4.3.1.6 Envío de soluciones y continuación del proceso

Completado el procedimiento de selección de una solución y, si es el caso, de modificación de los parámetros, se envía al proceso trabajador la solución de partida junto con los parámetros de la búsqueda. Además, el proceso jefe le envía el resto de soluciones de *CPOOL* de forma que el proceso trabajador no vuelva a explorar soluciones de calidad ya examinadas por alguno de los otros procesos. A continuación el proceso jefe queda a la espera de nuevas soluciones, identificando al proceso que las envía y volviendo a iterar, de forma similar a la empleada en Crainic y Gendreau (2002) o en Niar y Freville (1996).

4.3.1.7 Tareas finales del proceso jefe

Una vez completado el número prefijado de iteraciones globales, IT_GLOB, se envía la señal de parada al proceso trabajador cuyas soluciones acaban de recibirse. La salida del bucle de exploración continúa con la recepción de los últimos mensajes del resto de procesos, enviándoles la señal de parada y añadiendo a *CPOOL* las soluciones recibidas en caso necesario. El resultado se presenta en un archivo de salida especificado en la orden de ejecución. Por último se libera la memoria utilizada por el proceso.

La arquitectura descrita se puede clasificar dentro del tipo asíncrono. La información se comparte entre los procesos sólo bajo ciertas condiciones, como el agotamiento de la memoria, MAX_MEM, o el que se haya alcanzado el número máximo de iteraciones sin haber obtenido mejora MAX_IT. En la Figura 33 se muestra gráficamente el esquema propuesto de funcionamiento del algoritmo paralelo.

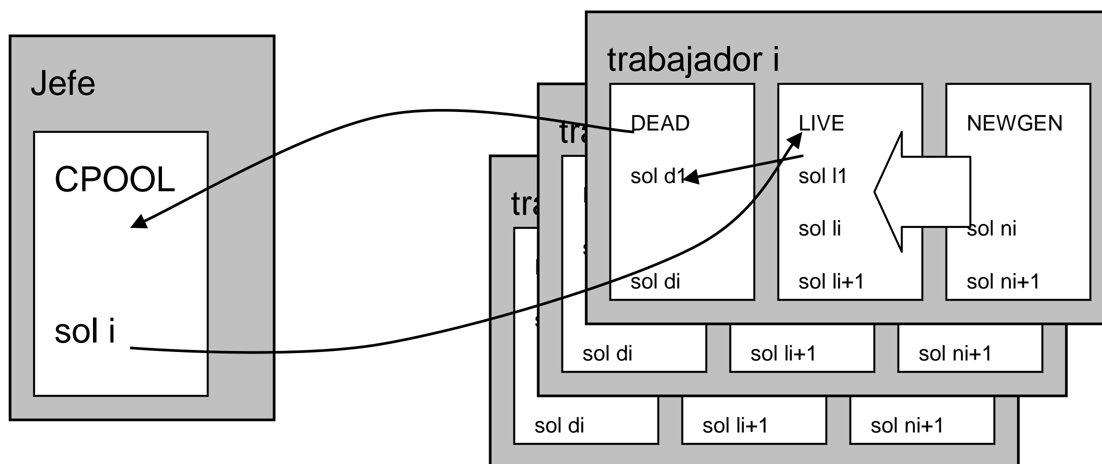


Figura 33: Esquema de funcionamiento en paralelo

4.3.2 Proceso trabajador

Como ya se mencionó, llamaremos procesos trabajadores en lugar de esclavos a los procesos que realizan el trabajo de exploración del espacio de soluciones, de forma similar a la denominación usada por Sittig *et al.* (1991). Los procesos trabajadores ejecutan una variante del algoritmo CLM propuesto por Ghosh y Sierksma

(2002) para el problema del viajante con algunos cambios para adaptarlo mejor al problema $F_m/prmu/C_{max}$. La descomposición del espacio de soluciones para la exploración por parte de los procesos trabajadores se realiza de forma implícita como se describe en Toulouse *et al.* (1999) sin más que hacer partir cada proceso de una solución diferente. Los parámetros que cambian de un proceso a otro son MAX_MEM y MAX_IT, tal como se ha descrito en el proceso jefe.

El pseudocódigo de un proceso trabajador se muestra en la Figura 34.

Antes de comenzar, cada proceso trabajador construye las variables necesarias para la comunicación entre procesos mediante la interfaz MPI. La interacción entre procesos, como la propuesta por Toulouse *et al.* (1999), se basa únicamente en el conocimiento de las exploraciones llevadas a cabo en las iteraciones anteriores, que reside en el proceso jefe y se comunica a cada trabajador al comienzo de cada iteración.

El proceso continúa con la recepción de una solución de partida junto con la estrategia de búsqueda enviada por el proceso jefe. Los parámetros de búsqueda enviados por el proceso jefe no incluyen los que controlan el valor del umbral de aceptación de soluciones, α y β, que se toman, según el trabajo de Ghosh y Sierksma (2002) entre 0 y 1. En nuestro caso, se han observado los mejores resultados con 0.1 para ambos parámetros.

La primera condición de parada es el tamaño de la memoria, MAX_MEM, y el número máximo de iteraciones sin obtener mejora MAX_IT como segunda condición de fin de la exploración. Los parámetros de búsqueda que se envían a cada proceso en la primera iteración son iguales.

El tope de memoria de cada proceso esclavo, para el algoritmo CLM, se elige en función de la dimensión del problema. En el caso del problema $F_m/prmu/C_{max}$ se realiza en función del número de trabajos.

El valor del parámetro K, según los experimentos de Ghosh y Sierksma (2002), se tomará bajo. Aunque estos autores proponen un valor de 1 ó 2, hemos obtenido mejores resultados con un valor de 5. Este parámetro tiene menos influencia sobre la calidad de las soluciones obtenidas que sobre el tamaño de las listas LIVE, DEAD y

NEW manejadas por los procesos trabajadores. Al aumentar el valor de K, se seleccionan más soluciones de la lista LIVE en cada iteración para explorar su vecindad, con lo que se alcanza con más rapidez el tope de memoria disponible.

Proceso trabajador:

Paso 1. Recibir la solución inicial S_0 y la política (MAX_MEM, MAX_IT, α_0, β …) desde el proceso jefe; $mk_{best} = mk_0$; $\alpha = \alpha_0$;

Paso 2. **Comienza** iteración

2.1. LIVE := LIVE $\cup$ { S_0 }

2.2. **Mientras** *condición_de_parada* = NO

2.2.1. **Si** LIVE $\neq$ {∅}

2.2.1.1. **Actualizar_umbral** τ

2.2.1.2. Tomar K soluciones S_i de LIVE

2.2.1.3. Generar soluciones vecinas S_{neig} de cada S_i

2.2.1.4. **Si** $mk_{neig} \leq \tau$

2.2.1.4.1. **Si** $S_{neig} \notin$ {LIVE, NEW, DEAD, POOL} NEW = NEW $\cup$ { S_{neig} }

2.2.1.4.2. **Si** *memoria* $\geq$ MAX_MEM

2.2.1.4.2.1. *condición_de_parada* := SI

2.2.1.4.2.2. Ir a Postprocesar

2.2.1.4.3. **Si** $mk_{neig} < mk_{best}$

2.2.1.4.3.1. *Iteraciones_sin_mejora* = 0

2.2.1.4.3.2. $mk_{best} = mk_{neig}$

2.2.1.5. Pasar cada solución explorada de LIVE a DEAD

2.2.1.6. LIVE := LIVE $\cup$ { NEW } ; NEW := ∅

2.2.1.7. *Iteraciones_sin_mejora* := *Iteraciones_sin_mejora* +1

2.2.1.8. **Si** *Iteraciones_sin_mejora* $\geq$ MAX_IT

condición_de_parada := SI

Postprocesar

2.3. **Postprocesar**

2.4. Enviar al proceso jefe las B mejores soluciones de DEAD

2.5. Recibir del proceso jefe la siguiente solución, política y señal de parada

2.6. **Si** *condición_de_parada* = SI Terminar proceso

2.7. Recibir un subconjunto del pool central de soluciones y almacenarlo en la lista POOL

Paso 3. **Fin** del proceso trabajador

Figura 34: Pseudocódigo del proceso trabajador

En el algoritmo CLM original, el tamaño total de las listas era $3n$ y el máximo número de iteraciones, $30n$, siendo n el tamaño del problema. En nuestro caso, el tamaño máximo de memoria es $n(n\text{-}1)\cdot 0.3$ y el máximo número de iteraciones sin mejora es 200, de forma que se alcance una cota superior de la función objetivo incluso para los problemas más grandes, en un espacio de tiempo razonablemente corto.

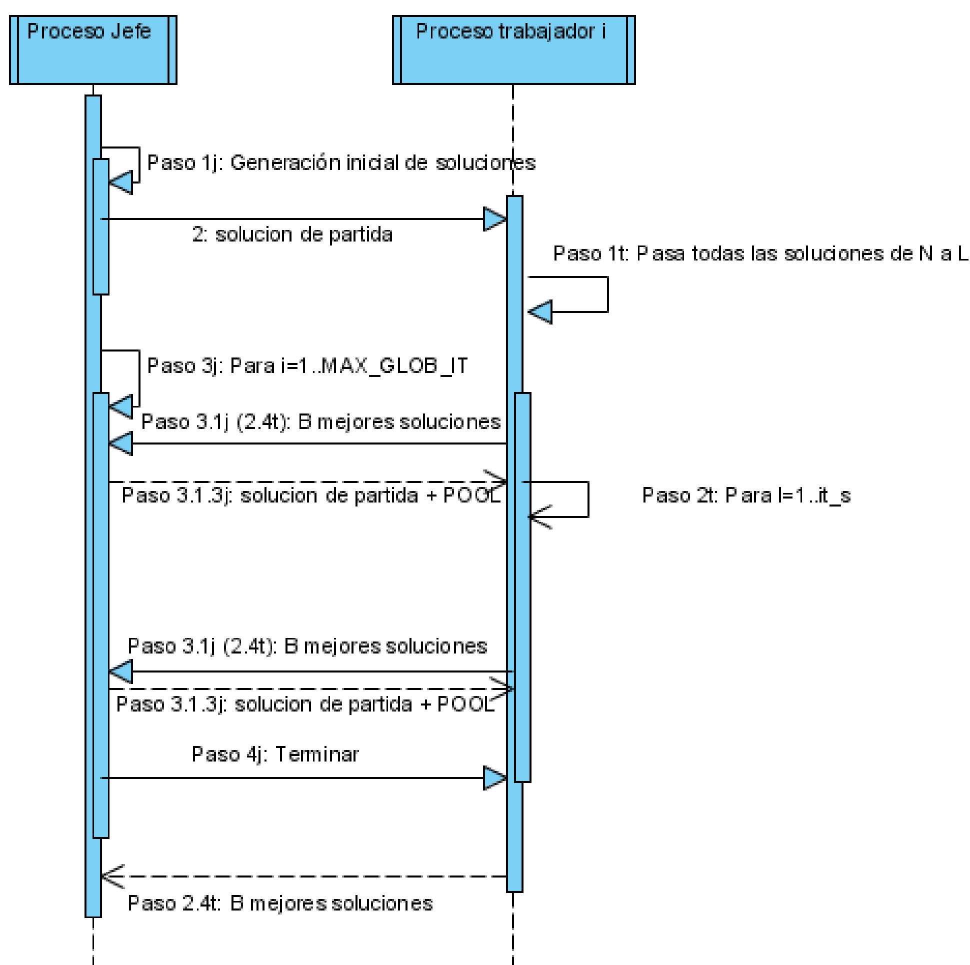


Figura 35: Diagrama de secuencia del algoritmo paralelo de grano grueso

Para terminar, el mensaje recibido contiene, en una etiqueta, la señal enviada por el proceso jefe de continuar la búsqueda o finalizar. En la Figura 35 se muestra un diagrama de secuencia para indicar el proceso de intercambio de mensajes entre el proceso jefe y los procesos trabajadores. Se indican los pasos correspondientes al

pseudocódigo de ambos procesos. Se han etiquetado los intercambios de información tal como aparecen en el pseudocódigo de cada proceso, terminando en j los correspondientes al proceso jefe y terminando en t los correspondientes al proceso trabajador.

Para nuestro algoritmo paralelo, hemos añadido en los procesos trabajadores otra lista, *POOL*, compuesta por un subconjunto del *pool* central, para evitar entrar en caminos de exploración que ya han sido recorridos por otros procesos o incluso por este mismo en una iteración anterior, ya que las otras listas de soluciones se están vaciando cada vez que se comunican los resultados al proceso jefe.

Tal como se explicó en la descripción del algoritmo CLM secuencial, los procesos trabajadores manejan tres conjuntos de soluciones para evitar la exploración repetitiva de las mismas soluciones. En nuestro caso, manejaremos listas de soluciones ordenadas según el valor de la función objetivo, para facilitar la implementación de la heurística. La lista LIVE contiene soluciones, junto con su valor de *makespan*, que van a ser exploradas. La lista DEAD contiene las soluciones que ya han sido exploradas y la lista NEW contiene las soluciones vecinas que acaban de ser generadas a partir de las tomadas de la lista LIVE.

Si la señal recibida del proceso jefe es de continuar la exploración, la solución recibida del proceso jefe junto con su valor de *makespan* forman un nodo que se coloca en la lista LIVE, con lo que comienza la ejecución del algoritmo CLM. En el pseudocódigo de la Figura 36, mk_{best} es el mejor valor de la función objetivo encontrado hasta el momento por este proceso, mk_{neig} es el correspondiente a la solución S_{neig}, vecina recién encontrada de la solución S_i cuya vecindad está siendo explorada.

Función Actualizar_umbral

Paso 1. $\alpha = \alpha \cdot \lambda$
Paso 2. $\boldsymbol{\tau = (1 + \alpha) mk_{best}}$

Figura 36: Pseudocódigo de la función Actualizar umbral

En los procesos trabajadores, para que una solución sea aceptada, tiene que superar un umbral que depende del mejor *makespan* hasta el momento y del número de iteraciones.

En nuestra versión del algoritmo, el procedimiento Postprocesar consiste en pasar todas las soluciones restantes en la lista LIVE cuando se cumple la condición de parada a la lista DEAD. En el algoritmo de Ghosh y Sierksma (2002) se realizaba una búsqueda local a partir de cada una de las soluciones de esta lista. En nuestra implementación, hemos comprobado que esta búsqueda local no encuentra soluciones de calidad y ralentiza el algoritmo enormemente, por lo que se ha tomado la decisión de no continuar con la exploración de estas soluciones.

Los valores MAX_IT y MAX_MEM están relacionados, de manera que elegir un valor grande para la memoria implica elegir un número grande de iteraciones sin mejora, para que la exploración no se detenga de forma prematura por alcanzarse alguno de los dos límites. Ambos parámetros dependen fuertemente del hardware disponible, sobre todo de la velocidad del procesador más que de la cantidad de memoria física disponible. Estos son los parámetros que guían la calidad de las soluciones, luego una mejor selección del resto de parámetros aprovechará mejor la potencia de cálculo disponible.

El proceso trabajador envía cada una de las B mejores soluciones de la lista DEAD junto con su valor de *makespan* al proceso jefe (punto 2.4. de la Figura 35), vacía dicha lista y queda a la espera de una nueva solución de partida para continuar con la búsqueda. Los parámetros como el tamaño de la memoria, el número de soluciones que se toman de LIVE para cada exploración o el número máximo de iteraciones sin mejora son enviados por el proceso jefe en cada iteración, debidamente corregidos. En caso de recibir la señal de parada, el proceso trabajador libera la memoria utilizada y finaliza su ejecución.

Sobre la estructura de vecindad, se pueden generar soluciones vecinas de la actual mediante el operador de búsqueda FSH o *forward shift*, que consiste en insertar el trabajo i después del trabajo j, situado siempre más adelante en la secuencia, como se muestra en la Figura 37. El número de soluciones vecinas que se pueden obtener de una dada es $n(n\text{-}1)/2$.

De esta forma, en los problemas más grandes, cuando se emplean tamaños de memoria inferiores no ya al espacio de soluciones, sino al tamaño de la vecindad, se reduce la búsqueda a una zona del espacio de soluciones cercana a la solución de partida con objeto de reducir el tiempo de exploración.

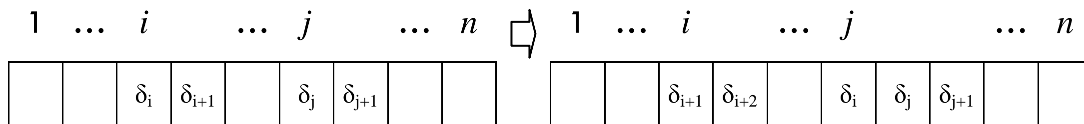


Figura 37: Desplazamiento hacia adelante

Se ha optado por generar las soluciones vecinas mediante intercambio aleatorio, con lo que la distancia entre las soluciones generadas será aleatoria. El intercambio aleatorio consiste en seleccionar dos posiciones cualesquiera del vector solución e intercambiar los valores entre esas dos posiciones (Figura 38). De este modo, aún con tamaños de memoria pequeños, se tienen mayores garantías de explorar una zona significativa de la vecindad.

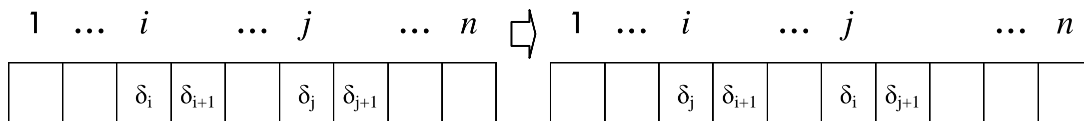


Figura 38: Intercambio aleatorio

4.3.3 Experimentación

Para la experimentación inicial con esta versión del algoritmo, se han ejecutado seis réplicas de cada uno de los 120 problemas de la conocida batería de Taillard (1993), empleando de uno a doce procesadores para comprobar tanto la escalabilidad como la robustez del algoritmo ante distintas instancias del problema. La batería contiene problemas de secuenciación desde 20 a 200 trabajos y de 5 a 20 máquinas generados de forma aleatoria. De esta batería se dispone, para diferentes configuraciones del problema de secuenciación de trabajos en flujo uniforme, de

los valores más cercanos al óptimo obtenidos hasta la fecha por diferentes equipos de investigadores (INA 2007), ver Tabla 77 en el Anexo 1.

Mediante la experimentación desarrollada, se ha buscado poner de manifiesto las prestaciones del algoritmo, sobre todo en cuanto a eficiencia y aceleración con respecto al número de procesadores empleados. De esta forma, se pretende conseguir dos objetivos.

- Reconocer si los resultados que se obtienen con el algoritmo desarrollado son mejores o comparables a los obtenidos con otros algoritmos paralelos descritos en la literatura. Se tomarán como algoritmos de referencia las versiones en paralelo de metaheurísticas similares a la empleada.
- Obtener el número óptimo de procesadores con el que nuestro algoritmo alcanza la máxima eficiencia, de manera que, en un futuro, pueda emplearse en condiciones reales de funcionamiento. Desde un principio, se ha buscado en este trabajo una configuración física de clúster que fuera aplicable en empresas de tamaño mediano, mediante una tecnología que es accesible y de coste contenido. Los resultados obtenidos en cuanto a eficiencia del algoritmo servirán de orientación sobre el número de procesadores a la hora de adquirir el sistema informático sobre el que se implemente el algoritmo.

A continuación, se describen las condiciones en las que se ha desarrollado la experimentación con el algoritmo secuencial y el resumen de los resultados de partida en cuanto a calidad de las soluciones obtenidas. Más tarde, se muestran los resultados de la experimentación seguida con el algoritmo CLMpg en cuanto a aceleración y eficiencia.

La aceleración se ha calculado comparando el tiempo de un proceso jefe y un trabajador corriendo en un solo procesador contra el tiempo usando más procesadores. Este es el algoritmo secuencial más parecido al paralelo, aunque no es realmente un proceso corriendo en un procesador. En el apartado siguiente, con el objeto de observar detalladamente el comportamiento del algoritmo CLMpg cuando aumenta el número de procesadores, se presentan los resultados en cuanto a eficiencia incremental generalizada.

Parámetro	Proceso	Descripción	Valor
α	Trabajador	Parámetro del umbral	0.1
β	Trabajador	Valor inicial del parámetro del umbral	0.1
K	Trabajador	Número de soluciones seleccionadas de la lista LIVE antes de actualizarla	2
MAX_IT	Trabajador	Número máximo de iteraciones sin mejora	200
TOP_MEM	Trabajador	Número máximo de soluciones almacenadas en las listas	0.2
TOP_*POOL*	Jefe	Número máximo de soluciones en la lista global	0.2
MAX_GLOB_IT	Jefe	Número de iteraciones	20
B	Jefe	Número de soluciones que se pasan de los trabajadores al proceso jefe	5

Tabla 22: Parámetros del algoritmo paralelo

Para comprobar si el algoritmo paralelo tiene un buen comportamiento conforme aumenta la potencia de cálculo disponible, se ha ejecutado el algoritmo en un solo procesador un número prefijado de iteraciones globales. De este modo, se tiene una cota del tiempo necesario para alcanzar una solución y se conoce su valor de *makespan*, con el proceso jefe y un proceso trabajador corriendo en el mismo procesador. Ya se han comentado anteriormente los valores de los parámetrosy se resumen en la Tabla 22.

El valor de α decrece rápidamente con el número de iteraciones luego, los valores iniciales de los parámetros α y β tienen una influencia limitada en la calidad de las soluciones. De hecho, según Ghosh y Sierksma (2002), el algoritmo es sensible al valor de α pero no al de β. La calidad de las soluciones mejora, en su caso, con valores más altos de α, pero también se incrementa el tiempo de resolución.

El tamaño máximo de memoria se elige proporcional al tamaño de la estructura de vecindad empleada en la fase de búsqueda local. El tamaño de ésta, si se emplea desplazamiento hacia delante o *forward shift* es $n(n\text{-}1)/2$, siendo n el tamaño de las soluciones, en nuestro caso, el número de trabajos. Aunque en el caso del intercambio aleatorio elegido en esta versión del algoritmo es mayor, se ha limitado a este tamaño reduciendo el número de intercambios. De esta forma se consigue limitar la

sensibilidad del algoritmo a variaciones del orden del tamaño de las soluciones. Por tanto, TOP_MEM = 0.2 significa en problemas de tamaño 100 que el número máximo de soluciones almacenadas en todas las listas es 0.2·100·99/2=990.

El número de soluciones almacenadas en los procesos trabajadores tiene gran influencia en el tiempo de computación, de forma que se ha elegido intencionadamente un valor bajo para alcanzar resultados globales en un tiempo razonable de experimentación. El tamaño máximo del *pool* central de soluciones se ha seleccionado con el mismo criterio: TOP_*POOL* · $n(n\text{-}1)/2$, siendo n el número de trabajos.

Las iteraciones globales especifican las veces que se envían soluciones de partida desde el proceso jefe a los trabajadores. Este número se multiplica por el número de procesos trabajadores para que éstos ejecuten el algoritmo el mismo número de veces, independientemente del número de procesos disponibles.

Dimensión	ARPD
20x5	0.06
20x10	0.05
20x20	0.03
50x5	0.03
50x10	0.06
50x20	0.05
100x5	0.02
100x10	0.06
100x20	0.10
200x10	0.06
200x20	0.12
500x20	0.10
Promedio	**0.06**

Tabla 23: ARPD promediados según la dimensión del problema con un solo procesador

La Tabla 23 muestra una comparación entre el valor de *makespan*, mk, obtenido mediante el algoritmo CLMpg ejecutado en un solo procesador con dos procesos, el jefe y un trabajador, y las cotas de Taillard (1993), mediante el porcentaje de desviación relativa (*Average Relative Percentage Deviation* – ARPD). Estos resultados se han obtenido después de seis ejecuciones de la batería completa con los paráme-

tros propuestos y partiendo de soluciones generadas al azar. La dimensión del problema se expresa como $n \times m$, siendo n el número de trabajos y m el número de máquinas.

Los promedios obtenidos por cada tamaño de problema son los que se muestran en la Tabla 23. Se observa que los resultados obtenidos son del mismo orden para distintos tamaños de problema. Por un lado, el tamaño máximo de memoria, una de las condiciones de parada, es proporcional al tamaño del problema, por lo que eran previsibles resultados no demasiado dispares en cuanto a calidad de las soluciones con diferentes tamaños de problema. Por otro lado, para no ralentizar demasiado la experimentación, sobre todo en el caso de los problemas mayores, se han tomado valores bajos tanto de tamaño máximo de memoria como del número de iteraciones sin mejora.

Para observar el comportamiento del algoritmo cuando se añaden más procesadores, se ha medido el tiempo necesario para encontrar una solución de la misma calidad que las encontradas con un solo procesador, empleando esta vez distinto número de procesadores. Se comienza con dos procesos, el jefe y un trabajador, corriendo ahora en dos procesadores y se termina con el jefe y once trabajadores corriendo en doce procesadores. Los parámetros elegidos son los mismos que en nuestra versión secuencial, excepto el número de iteraciones globales, ya que ahora la condición global de parada es el valor de la función objetivo de la versión con un solo procesador.

Los valores que se toman como referencia para cada instancia del problema son los valores promedio de *makespan* obtenidos después de ejecutar seis réplicas de cada instancia en un solo procesador. Cada uno de los problemas de la batería se ha ejecutado en su versión paralela cinco veces con cada una de las instancias del problema, lo que resulta en un total de 3600 experimentos. La Tabla 24 muestra los tiempos promedio en segundos para alcanzar una solución con el mismo valor de *makespan* que con un solo procesador, agrupados según la dimensión del problema, desde el jefe y un trabajador en un solo procesador, 2(1), nuestra versión secuencial, hasta un jefe y once trabajadores en doce procesadores.

	2(1)	**2**	**4**	**6**	**8**	**10**	**12**
20x5	0.03	0.05	0.02	0.01	0.01	0.01	0.06
20x10	0.18	0.29	0.08	0.05	0.04	0.04	0.02
20x20	0.24	0.26	0.08	0.11	0.06	0.08	0.05
50x5	0.20	0.61	0.26	0.14	0.11	0.24	0.14
50x10	2.88	4.93	2.80	0.74	1.31	0.87	0.94
50x20	88.55	57.34	25.96	25.10	13.64	6.72	9.74
100x5	3.62	7.09	1.94	2.02	1.05	0.94	1.05
100x10	6.14	3.65	1.69	1.44	1.20	1.24	1.14
100x20	16.33	6.39	3.15	2.67	2.33	2.40	2.29
200x10	70.69	72.88	19.99	15.57	14.27	14.03	12.81
200x20	122.78	86.25	23.60	19.76	17.74	17.40	16.13
500x20	10773.80	7310.50	2235.68	1635.43	1418.91	1374.24	1340.47
Promedio	**968.67**	**672.42**	**244.50**	**175.25**	**152.83**	**152.92**	**147.08**

Tabla 24: Tiempo promediado según la dimensión del problema y el número de procesos

En general, el algoritmo paralelo mejora al secuencial, especialmente en los problemas mayores. En los problemas pequeños, se han obtenido resultados variables, ya que el tamaño de memoria e iteraciones sin mejora se han tomado pequeños para no ralentizar en exceso la exploración en los problemas mayores. Así, se han destacado en sombreado en la Tabla 24 tiempos de ejecución que han resultado peores con más procesadores. En los problemas de menor dimensión, de menos de 100 trabajos, el algoritmo paralelo no ha llegado a obtener soluciones de la misma calidad que el algoritmo secuencial en 165 ocasiones de un total de 3000 experimentos. Estos resultados son similares a los obtenidos por Crainic y Gendreau (2002), 6 sobre 180 ejecuciones. En problemas de dimensión mayor, el algoritmo paralelo ha superado siempre al algoritmo secuencial, lo que hace pensar de nuevo que los parámetros de búsqueda seleccionados son más adecuados para problemas difíciles.

La Tabla 25 muestra la eficiencia (ver apartado 2.3.1.2) del algoritmo CLMpg propuesto para cada tamaño de problema en cuanto a trabajos y máquinas disponibles, de dos (columna 2) hasta doce (columna 12) procesadores:

En algunos casos, se observan resultados cercanos a la unidad mostrándose sombreados los superiores a la unidad. Estos resultados son mejores en el caso de los problemas con mayor número de trabajos y corroboran la observación anterior de que los parámetros seleccionados son más apropiados para los problemas de ma-

yor dimensión. Se puede concluir por tanto, que el algoritmo es lo suficientemente robusto en un amplio rango de instancias del problema de secuenciación de trabajos en flujo uniforme.

	2	4	6	8	10	12
20x5	0.26	0.45	0.38	0.50	0.38	0.04
20x10	0.31	0.56	0.64	0.53	0.43	0.63
20x20	0.46	0.73	0.36	0.47	0.30	0.41
50x5	0.16	0.19	0.24	0.22	0.08	0.12
50x10	0.29	0.26	0.65	0.27	0.33	0.26
50x20	0.77	0.85	0.59	0.81	1.32	0.76
100x5	0.26	0.47	0.30	0.43	0.38	0.29
100x10	0.84	0.91	0.71	0.64	0.49	0.45
100x20	1.28	1.30	1.02	0.88	0.68	0.59
200x10	0.49	0.88	0.76	0.62	0.50	0.46
200x20	0.71	1.30	1.04	0.87	0.71	0.63
500x20	0.74	1.20	1.10	0.95	0.78	0.67
Promedio	0.55	0.76	0.65	0.60	0.53	0.44

Tabla 25: Eficiencia del algoritmo paralelo según tamaño de problema y número de procesos

Los resultados de la experimentación anterior medidos ahora en cuanto a la eficiencia incremental generalizada (ver apartado 2.3.1), en el caso de los problemas de mayor dimensión (de 100 y 200 trabajos) son en promedio los que se muestran en la Tabla 26 y siguientes. Así, para los problemas de 200 trabajos en 20 máquinas, la eficiencia incremental generalizada cuando se pasa de ocho a doce procesadores es $gie(8, 12) = 0.73$.

Se observan resultados similares a los mostrados en las tablas de eficiencia. Cuando se añaden más procesadores, la eficiencia decae lentamente como observaron otros autores como Taillard (1991) o Fiechter (1994), debido fundamentalmente a los tiempos de espera entre jefe y trabajadores y por la fracción secuencial que introduce el proceso jefe, según la ley de Amdahl.

Por otro lado, se han obtenido aceleraciones superiores a la lineal cuando se pasa de dos a cuatro o seis procesadores. Este efecto muestra la mejora en la calidad de las soluciones obtenida al repetir el algoritmo desde puntos de partida diferentes y a la colaboración entre procesos.

100x5	4	6	8	10	12
2	1.83	1.17	1.69	1.50	1.13
4		0.64	0.92	0.82	0.62
6			1.44	1.29	0.97
8				0.80	0.67
10					0.75

100x10	4	6	8	10	12
2	1.08	0.85	0.76	0.59	0.54
4		0.78	0.71	0.55	0.50
6			0.90	0.69	0.63
8				0.84	0.70
10					0.91

Tabla 26: Eficiencia incremental generalizada del algoritmo paralelo para problemas con 100 trabajos y 5 o 10 máquinas según la variación en el número de procesos

100x20	4	6	8	10	12
2	1.01	0.80	0.68	0.53	0.46
4		0.79	0.68	0.53	0.46
6			0.86	0.67	0.58
8				0.81	0.68
10					0.87

200x10	4	6	8	10	12
2	1.82	1.56	1.28	1.04	0.95
4		0.86	0.70	0.57	0.52
6			0.82	0.67	0.61
8				0.89	0.74
10					0.91

Tabla 27: Eficiencia incremental generalizada del algoritmo paralelo para problemas con 100 trabajos y 20 máquinas o 200 trabajos y 10 máquinas según la variación en el número de procesos

200x20	4	6	8	10	12
2	1.82	1.44	1.20	0.99	0.89
4		0.80	0.68	0.54	0.49
6			0.85	0.68	0.61
8				0.81	0.73
10					0.91

500x20	4	6	8	10	12
2	1.72	1.52	1.32	1.07	0.92
4		0.90	0.79	0.64	0.55
6			0.87	0.71	0.61
8				0.83	0.71
10					0.86

Tabla 28: Eficiencia incremental generalizada del algoritmo paralelo para problemas con 200 trabajos y 20 máquinas o 500 trabajos y 20 máquinas según la variación en el número de procesos

Dado que uno de los motivos por los que decae la eficiencia con el número de procesadores es el tiempo de espera introducido por el proceso jefe tratando de atender las peticiones de los trabajadores, una mejora pasará por ir disminuyendo el procesamiento en el proceso jefe hasta llegar a un esquema de comunicación entre iguales o *peer-to-peer*, en el que no existe un proceso jefe, para lo que será necesario implementar mecanismos diferentes para compartir información entre todos los procesos. Otra mejora podría producirse al incrementar el número de iteraciones en los trabajadores, de forma que se disminuya la proporción de tiempo con respecto al total empleada en el proceso jefe. Esta mejora se vería contrarrestada por el menor número de intercambios de información entre procesos, por lo que sería necesario un estudio más amplio para considerarla.

El comportamiento del algoritmo propuesto se aproxima al observado por Aiex *et al*. (2002), con la eficiencia disminuyendo conforme aumenta el número de procesadores, teniendo su valor máximo con cuatro procesadores. No obstante, su eficiencia es inferior a la obtenida mediante redes de *transputers*, por ejemplo por Taillard (1990) o Taillard (1991), aunque más cercana a la de Taillard (1994). Sin embargo, se observa un mejor rendimiento con el algoritmo CLMpg que en el caso de algoritmos paralelos funcionando en un sistema paralelo virtual, como en el caso de Adenso-Díaz *et al*. (2006) o redes de PCs, como Brüngger *et al*. (1999).

4.4 Algoritmo paralelo de grano fino CLMpf

En la versión anterior del algoritmo, CLMpg, se observó que el incremento en el número de iteraciones globales disminuyendo el tiempo de exploración de cada proceso daba como resultado mejor calidad de las soluciones disminuyendo a la vez el tiempo total de exploración. Para reducir el tiempo de exploración en cada proceso, se disminuyó tanto la memoria máxima como el número de iteraciones sin mejora. Este resultado también lo hemos puesto de manifiesto en León et al. (2006) y en trabajos de otros autores como Bożejko y (2008a). En este apartado se ha rediseñado el algoritmo propuesto con anterioridad, de forma que mejore el tiempo en el que ofrece soluciones de buena calidad. Para ello se ha aumentado el número de veces que los procesos intercambian información entre sí, es decir, incrementando la granularidad del algoritmo paralelo.

Para ello, se va a introducir un punto más de intercambio de información entre los procesos trabajadores. No solo se intercambiará información mediante el proceso jefe, al final de cada serie de iteraciones en los procesos trabajadores, sino que estos intercambiarán entre sí las soluciones exploradas, al final de cada iteración, de forma que se mejore el conocimiento de las zonas del espacio de soluciones que ya han sido visitadas.

Para lograr el intercambio correcto de información entre los procesos trabajadores, se ha creado un comunicador nuevo para ellos, de forma que la información se transmita entre este conjunto de procesos aparte de la que se intercambia entre los procesos trabajadores y el proceso jefe.

4.4.1 Proceso jefe

El proceso jefe continúa siendo igual que en la versión asíncrona de grano grueso (ver Figura 39). Sólo se ha modificado la función "Generar nueva solución y política" para tener en cuenta que ahora la condición de parada en los procesos trabajadores no incluye el número de iteraciones sin mejora ni el tamaño de la memoria (ver Figura 40). Para intensificar la búsqueda únicamente se variará el parámetro K.

Proceso jefe:

Paso 1. Inicialización
Paso 2. **Para cada** proceso trabajador
  2.1. Generar solución inicial S_0 y estrategia de búsqueda
  2.2. Almacenar la solución y la estrategia en POOL
  2.3. Enviar a cada proceso su solución junto con su estrategia
Paso 3. **Durante** MAX_GLOB_IT iteraciones
  3.1. **Cada vez** que un trabajador devuelva sus B mejores soluciones
    3.1.1. Almacenar en POOL las soluciones recibidas junto con la estrategia empleada.
    3.1.2. **Generar nueva solución y política**
    3.1.3. Enviar a ese proceso la solución generada y la nueva estrategia de búsqueda
Paso 4. Enviar la señal de parada a los procesos trabajadores
Paso 5. Finalizar el proceso y almacenar los resultados en un archivo

Figura 39: Pseudocódigo del proceso jefe o master

Función Generar nueva solución y política:

1. Seleccionar una solución de POOL
2. **Si** la solución seleccionada está cerca de la última solución recibida del proceso trabajador Incrementar el valor de K
3. **Si no** Mantener el valor de K

Figura 40: Pseudocódigo de la función Generar nueva solución y política del proceso jefe

4.4.2 Proceso Trabajador

Los procesos trabajadores disponen de más información sobre la búsqueda que realizan el resto de procesos, frente a la que disponían en el algoritmo CLMpg, gracias a mayor número de puntos de intercambio de la información. El nuevo proceso trabajador se detalla en la Figura 41.

Proceso trabajador:

Paso 1. Recibir la solución inicial S_0 y la política (MAX_MEM, MAX_IT, α_0, β ...) desde el proceso jefe, $mk_{best} = mk_0$, $\alpha = \alpha_0$;

Paso 2. **Mientras** $master_stop \neq 1$

2.1. LIVE :=LIVE $\cup$ $\{S_0\}$

2.2. **Para** (***it_s***)

2.2.1. **Actualizar** umbral τ

2.2.2. **Para** $1..K$

2.2.2.1. **Si** LIVE $\neq \emptyset$

2.2.2.1.1. Tomar solución Si de LIVE

2.2.2.1.2. **Para** $(1..jobs)$x$(1..jobs/2)$

2.2.2.1.2.1. Generar solución vecina S_{neig} de S_i

2.2.2.1.2.2. **Si** $S_{neig} \in$ POOL

2.2.2.1.2.2.1. Generar solución vecina S_{neig} de S_i

2.2.2.1.2.2.2. **Si** $mk_{neig} \leq \tau$

Si $S_{neig} \notin$ {LIVE, NEW, DEAD} LIVE :=LIVE $\cup$ $\{ S_{neig} \}$

Si $mk_{neig} < mk_{best}$ $mk_{best} := mk_{neig}$

2.2.3. Envío/recepción del número de soluciones vecinas encontradas

2.2.4. **Si** $B <$ mín vecinas encontradas

2.2.4.1. Envío/recepción de las B mejores de NEW

2.2.4.2. **Si** las recibidas $\notin$ {POOL, NEW} Se añaden a NEW

2.2.5. LIVE := LIVE $\cup$ NEW; NEW := $\emptyset$

2.3. **Postprocesar**

2.4. Devolver al proceso jefe las B mejores soluciones de DEAD

2.5. Recibir del proceso jefe la siguiente solución, política y señal de parada

2.6. **Si** Condición de parada = SÍ, Terminar proceso

2.7. Recibir un subconjunto del pool central de soluciones y almacenarlo en la lista POOL local

Paso 3. **Fin** del proceso trabajador

Figura 41: Pseudocódigo del proceso trabajador o esclavo

Para describir la comunicación de una forma más detallada, en la Figura 42 se presenta un esquema de la misma mediante un diagrama de secuencia UML. Se han etiquetado los intercambios de información tal como aparecen en el pseudocódigo de cada proceso, terminando en j los correspondientes al proceso jefe, terminando en ti los correspondientes al proceso trabajador i y terminando en tj los correspondientes al trabajador j.

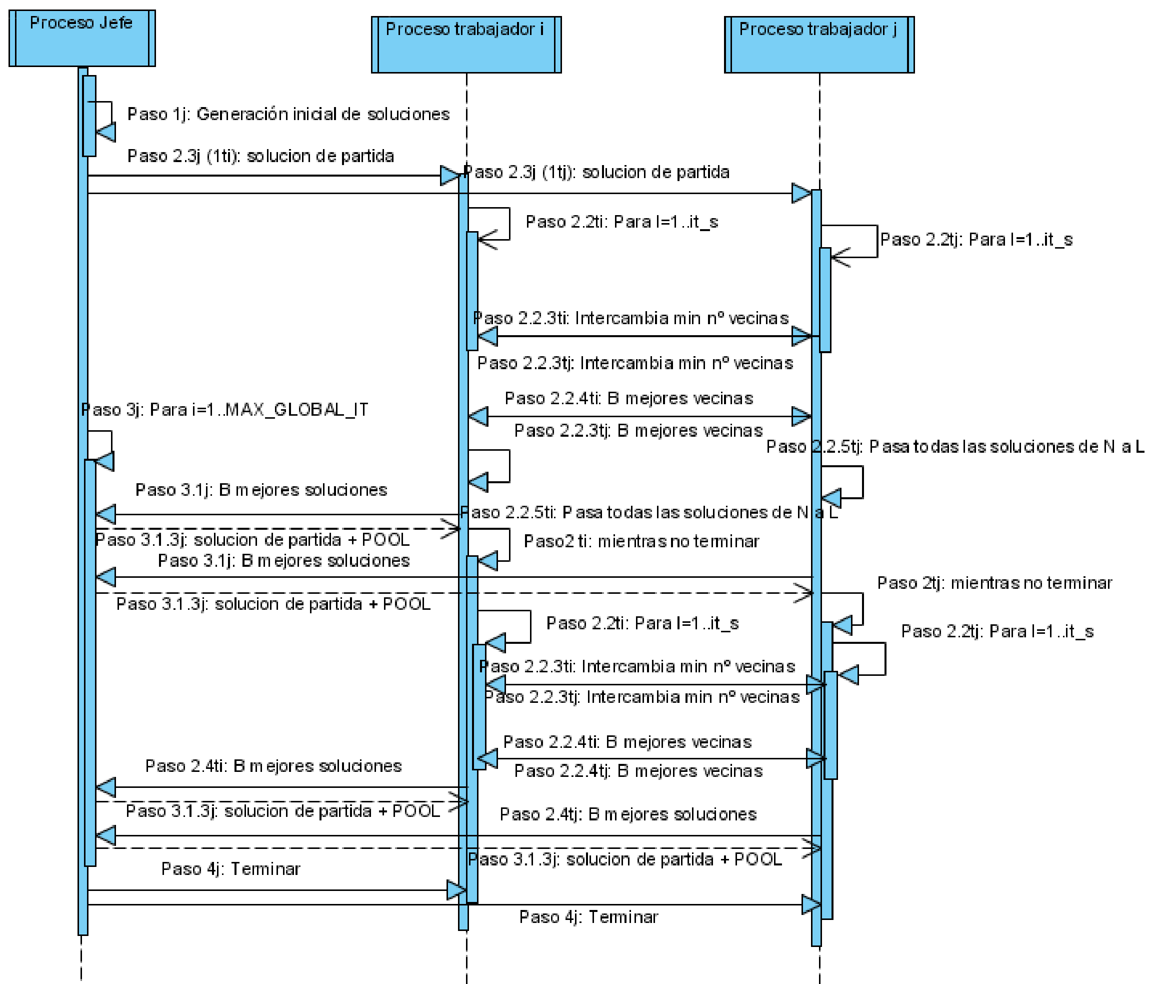


Figura 42: Diagrama de secuencia de la comunicación entre procesos

En esta situación, los procesos trabajadores deben sincronizar el instante en el que intercambian la información entre ellos, por lo que en esta versión del algoritmo se emplea como condición de parada un número de iteraciones que habrá que prefi-

jar, en lugar de emplear como condición de parada un número de iteraciones sin mejora o un tope de memoria.

Esto trae como consecuencia una disminución en la eficiencia del algoritmo paralelo frente a la versión asíncrona del mismo. Recordemos que la eficiencia del algoritmo paralelo relaciona la ganancia de tiempo de ejecución con el incremento en el número de procesadores disponible. Para lograr una eficiencia próxima a la unidad, o al 100%, el tiempo de ejecución debería disminuir de forma proporcional al número de procesadores. En esta versión síncrona del algoritmo, los procesos trabajadores deben esperar a que todos los demás hayan concluido su iteración de búsqueda para posteriormente intercambiar las soluciones encontradas. Estos tiempos de espera hacen que la eficiencia del algoritmo disminuya con el número de procesadores de forma más acusada a como lo hacía en el caso del algoritmo asíncrono. Por este motivo, se va a proceder desde el principio a un ajuste de los parámetros del algoritmo de forma que se minimice la pérdida de eficiencia del mismo.

4.4.3 Diseño experimental

4.4.3.1 Descripción de los factores

El interés del ajuste de los parámetros en cualquier algoritmo de búsqueda se basa en la sensibilidad de este tipo de algoritmos a pequeñas variaciones en los valores seleccionados para dichos parámetros. Esta sensibilidad se manifiesta en tiempos de ejecución muy variables y en variaciones en la calidad de las soluciones encontradas por el algoritmo, como han confirmado nuestras observaciones.

En nuestro caso, el tiempo de ejecución del algoritmo secuencial se incrementa enormemente cuando se intensifica la búsqueda. Esta intensificación se da cuando se aumenta el número de soluciones exploradas o bien cuando se incrementa el número de iteraciones que realiza el algoritmo. Al mismo tiempo, esta intensificación hace posible encontrar soluciones de mejor calidad, por lo que la combinación de parámetros que se escoja implicará un compromiso entre tiempo de ejecución y calidad de las soluciones encontradas.

A esta dificultad se añade la de trabajar en un entorno paralelo, con lo que se hace necesario ajustar otro conjunto de parámetros propios de estos algoritmos, como son

el número de soluciones intercambiadas entre procesos, el número de procesos empleados o el número de iteraciones totales del algoritmo. Autores como Gupta *et al.* (2000) y también Adenso-Díaz *et al.* (2006) proponen metodologías para seleccionar incluso la mejor arquitectura posible para el algoritmo.

Se han propuesto numerosas estrategias para mejorar la calidad de las soluciones obtenidas y el tiempo de ejecución de algoritmos de búsqueda mediante su implementación en paralelo. La mayoría coincide en afirmar que las estrategias de múltiples pasadas en colaboración superan las prestaciones combinadas de tiempo de ejecución y calidad de soluciones de estrategias de una sola pasada o de múltiples pasadas independiente. Se puede consultar Adenso-Díaz *et al.* (2006) para una revisión cuantitativa de diferentes estrategias. Otras revisiones se pueden consultar en Cung *et al.* (2001), o las anteriores, de Flynn (1966) o la de Verhoeven y Aarts (1995).

Sobre el ajuste de los parámetros relacionados con el algoritmo paralelo, la mayoría de autores prefiere experimentar con el número de procesadores desde 1 hasta el total de procesadores disponibles, con lo que se trata de justificar que la mejor elección suele ser la que implica mayor número de procesadores, siempre teniendo en cuenta que el algoritmo disminuye su eficiencia conforme este número se incrementa. Normalmente, no se precisa de mayores ajustes, ya que se opta por no añadir parámetros adicionales a los del algoritmo secuencial, cuyos valores se toman de los empleados para dicho algoritmo.

Hay autores como Bożejko y Wodecki (2004a), o bien, Attanasio *et al.* (2004), que reducen el número de iteraciones en cada proceso de forma proporcional al número de procesos, para reducir así el tiempo de exploración, manteniendo el tiempo total.

Autores como Cordeau y Laporte (2004), tras estudiar varias versiones de búsqueda tabú en paralelo aplicadas al problema de asignación de rutas de vehículos, proponen reducir el número de parámetros aun a costa de perder algo de calidad en la solución final.

La cooperación entre procesos ha sido confirmada como superior en prestaciones a la búsqueda secuencial o al empleo de procesos independientes. Crainic y Gendre-

au (2002) proponen el intercambio de información entre procesos mediante un *pool* de soluciones de donde tomarán los procesos la información necesaria, de forma que se evite la convergencia prematura hacia mínimos locales que presentan los algoritmos paralelos. Para evitar esta convergencia prematura, proponen diferentes estrategias de selección de soluciones del *pool* central. Tras diferentes experimentos, llegaron a la conclusión de que la mejor estrategia debía tener en cuenta tanto la bondad de las soluciones almacenadas en el *pool* como el número de iteraciones que llevan permaneciendo en los primeros lugares en cuanto a calidad. Para ello asignan a cada solución i un coeficiente de movilidad m_i. Cada vez que se añade al *pool* central una solución nueva, el coeficiente de movilidad de todas las soluciones mejores que la introducida, se incrementa en una unidad. De esta forma, las soluciones con mayor coeficiente de movilidad, son las que llevan más tiempo entre las mejores. Así, se seleccionan soluciones del *pool* central con una probabilidad dada por la expresión 19, siendo esta estrategia es la que hemos seguido en esta versión asíncrona de grano fino de nuestro algoritmo paralelo.

$$P(S_i) = \frac{m_i + n - i + 1}{\sum_1^n (m_i + 1)} \qquad (19)$$

Para llegar a un compromiso entre intercambiar información y no interferir demasiado en la exploración de cada proceso, de forma que pudiera alcanzarse una convergencia demasiado prematura hacia óptimos locales, Crainic y Gendreau (2002) proponen intercambios de información que den lugar a menos comunicación. Básicamente, se evitará el intercambio de información durante la fase de exploración de soluciones vecinas de una dada, centrándose en intercambios cuando se va a pasar a otra solución o también, en su caso, búsqueda tabú, antes de proceder a una fase de diversificación. La selección de parámetros relativa a la búsqueda local, se basa, al igual que otros autores, en la versión secuencial del mismo algoritmo. Así aseguramos no recorrer los mismos caminos de exploración con procesos diferentes.

A diferencia de los trabajos citados con anterioridad, en los que se prefería el intercambio de una sola solución en cada fase de comunicación, hemos preferido intercambiar las B mejores soluciones encontradas por cada proceso trabajador, tal como proponen Niar y Freville (1996). El rango de valores para el número de soluciones intercambiadas se ha elegido después de una experimentación previa basada en los valores propuestos por Niar y Freville (1996).

El tamaño del *pool* central se ha seleccionado, tal como hacen Ghosh y Sierksma (2002) con el tamaño de los conjuntos de soluciones almacenadas para la búsqueda local, proporcional al tamaño del problema que se está tratando de resolver. En nuestro caso, tiene que ver con el tamaño de la vecindad que implica emplear el método FSH para generar soluciones vecinas:

$$num_sol = pool\frac{n(n-1)}{2} \quad (20)$$

El factor $pool$ nos permite experimentar con diferentes valores próximos al propuesto.

Por último, se han seguido consideraciones relacionadas con el tiempo de ejecución de la búsqueda para seleccionar el rango de valores para los parámetros que gobiernan el final de la exploración. En nuestro caso, el número de iteraciones globales y el número de iteraciones en los procesos trabajadores, ya que ahora no sirven los criterios de parada empleados en la versión anterior del algoritmo, basados en el número total de soluciones almacenadas en las listas por los procesos trabajadores ni el número de iteraciones sin alcanzar mejora. Es necesaria la sincronización entre procesos.

Como se ha comentado anteriormente, los parámetros relacionados con el proceso de búsqueda local, que se realiza en los procesos trabajadores, son los parámetros heredados del algoritmo secuencial, en el que se basa este algoritmo paralelo. Los relacionados con el algoritmo paralelo son el número de iteraciones globales, el número de procesos, el número de soluciones intercambiadas y el máximo de soluciones almacenadas en el proceso jefe. Estos parámetros se resumen en la Tabla 29.

	Parám.	Descripción	Niveles
secuencial	α	Componente del umbral de aceptación de soluciones	0.2; 0.3; 0.4
	β	Componente del umbral de aceptación de soluciones	0.2; 0.3; 0.4
	It_s	Condición de parada: número de iteraciones que realizan los procesos trabajadores antes de detener la búsqueda	3; 5
	K	Número de soluciones que se toman del conjunto disponible para generar soluciones vecinas en cada iteración de los procesos trabajadores	2; 3; 4
paralelo	It_m	Condición de parada: Número de iteraciones que se realizan en el proceso jefe por cada proceso trabajador disponible	5; 10; 15
	p	Número total de procesos que se van a emplear en el algoritmo paralelo. Está limitado por el total de procesadores de los que consta el clúster donde se realiza la experimentación	4; 8; 12
	B	Número de soluciones que se intercambian en cada iteración entre los procesos trabajadores y entre estos y el proceso jefe	2; 5; 8
	$pool$	Número de soluciones que se almacenan en el proceso jefe. El total de soluciones guardadas en el proceso jefe está gobernado por la expresión 20.	0.3; 0.5; 0.7

Tabla 29: Parámetros de la búsqueda local

Con el objeto de poder realizar un estudio inicial sobre la influencia (ajuste) de los parámetros que intervienen en las respuestas, hemos tenido en cuenta los siguientes aspectos:

4.4.3.2 Número de niveles

Parece interesante estudiar la no linealidad de los parámetros, por lo que para la mayoría de ellos se han establecido 3 niveles. Sus valores extremos están tomados de la literatura o teniendo en cuenta la disponibilidad del sistema informático, tal como se ha mostrado en Tabla 29.

En concreto, los parámetros que tienen que ver con el algoritmo CLM secuencial, son los propuestos por sus autores Ghosh y Sierksma (2002), mientras que el resto se han tomado de algoritmos paralelos de búsqueda local.

El valor del parámetro B está tomado del algoritmo propuesto en Niar y Freville (1996). La propuesta de un *pool* central para imitar el comportamiento se ha propuesto en gran variedad de algoritmos paralelos, como por ejemplo el de Le Bouthillier y Crainic (2005).

El número de procesos empleado se ha tomado en función del número de procesadores disponibles, de forma que cada proceso pueda ejecutarse en un solo procesador. De esta forma se evitan interferencias en la ejecución que podrían falsear los resultados de tiempo de ejecución. Hay que notar que autores como Taillard (1990) hacían funcionar un proceso master y un proceso esclavo en un solo procesador, aprovechando que el trabajo del proceso master era muy inferior al del proceso esclavo. Esta política no es aplicable en nuestro caso, ya que el trabajo del proceso jefe al generar nuevas soluciones para otros procesos trabajadores podría interferir ralentizando el proceso de búsqueda de un proceso trabajador que estuviera corriendo también en ese procesador.

El número de iteraciones en los procesos trabajadores se ha elegido de forma que no supongan excesivo tiempo de exploración en los procesos trabajadores. En el caso del proceso jefe, se ha seleccionado el número de iteraciones globales de forma que se detenga la exploración cuando la mejora en cada iteración no es significativa.

4.4.3.3 Selección del tipo de diseño experimental (DOE – *Design of Experiments*)

Dado que se trata de una experimentación inicial, solamente estamos interesados en la posible influencia de los factores principales que hemos considerado en las respuestas del sistema. Por ello empleamos en primer lugar un diseño del tipo *screening* saturado. Este tipo de diseños se caracteriza por contener una gran confusión entre los diferentes parámetros, pero trataremos de seleccionar un diseño que no tenga confundidos efectos principales entre sí, sino que cada efecto principal esté confun-

dido con al menos interacciones de dos o más parámetros. Entre las alternativas existentes optamos por un diseño factorial fraccional mixto de 1 factor a 2 niveles y 7 factores a 3 niveles, $2x3^{7-5}$, también denominado diseño L18 de Taguchi (ver, por ejemplo, Montgomery, 2002). El empleo de este tipo de diseños aporta una serie de ventajas:

- Se basan en una matriz ortogonal, con lo que simplifica el análisis y la interpretación de los resultados.

- Se obtiene una gran cantidad de información sobre los efectos principales con un reducido número de experimentos (en nuestro caso bastan 18 experimentos).

- Se obtiene información sobre la no linealidad de los términos (parámetros de 3 niveles)

4.4.3.4 Datos

Se ha seguido una experimentación mediante la conocida batería de Taillard (1993) para tratar de ajustar el valor de los parámetros del algoritmo paralelo de grano fino presentado.

Atendiendo al número de parámetros que se van a ajustar y al número de niveles para cada uno de ellos, se han realizado 3 réplicas para cada una de las 10 instancias de cada tamaño de problema, de forma que pueda extraerse información sobre el error (son necesarias al menos dos réplicas): 50x5, 50x10, 50x20, 100x5, 100x10 y 100x20 (trabajos x máquinas).

En total, 60 experimentos para cada uno de los 18 tratamientos propuestos.

4.4.3.5 Respuestas

Las respuestas de interés en nuestro estudio son el tiempo y la calidad de las soluciones obtenidas. Para medir la calidad de las soluciones se ha empleado el promedio del porcentaje de desviación relativa, ARPD (Average Relative Percentage Deviation) con respecto a las mejores soluciones conocidas en la literatura para el problema de secuenciación de trabajos en flujo uniforme sin restricciones de capacidad. Para el tiempo de resolución, se ha medido el tiempo total de ejecución del algoritmo y se ha tomado, como en el caso del ARPD, el promedio en segundos.

4.4.3.6 Número de replicados

El algoritmo puede obtener resultados diferentes dependiendo de la semilla pseudoaleatoria. Este es el motivo de haber realizado 3 réplicas de cada tratamiento, la cual nos permitirá estudiar la variabilidad en cada tratamiento. El diseño resultante se muestra en la Tabla 30.

Run	α	β	*it_s*	*it_m*	*k*	*p*	*B*	*pool*
1	0.2	0.2	3	5	2	4	2	0.3
2	0.2	0.3	3	10	3	8	5	0.5
3	0.2	0.4	3	15	4	12	8	0.7
4	0.3	0.2	3	5	3	8	8	0.7
5	0.3	0.3	3	10	4	12	2	0.3
6	0.3	0.4	3	15	2	4	5	0.5
7	0.4	0.2	3	10	2	12	5	0.7
8	0.4	0.3	3	15	3	4	8	0.3
9	0.4	0.4	3	5	4	8	2	0.5
10	0.2	0.2	5	15	4	8	5	0.3
11	0.2	0.3	5	5	2	12	8	0.5
12	0.2	0.4	5	10	3	4	2	0.7
13	0.3	0.2	5	10	4	4	8	0.5
14	0.3	0.3	5	15	2	8	2	0.7
15	0.3	0.4	5	5	3	12	5	0.3
16	0.4	0.2	5	15	3	12	2	0.5
17	0.4	0.3	5	5	4	4	5	0.7
18	0.4	0.4	5	10	2	8	8	0.3

Tabla 30: Matriz de Diseño experimental L18 de Taguchi

Tras ejecutar la batería de Taillard (1993) para problemas de 50 y 100 trabajos con cada uno de los 18 tratamientos anteriores, se obtuvieron, en promedio, los siguientes resultados en cuanto a ARPD y tiempo de ejecución, en promedio, en segundos mostrados en la Tabla 31:

	ARPD (%)			Tiempo (s)		
	Rep #1	**Rep #2**	**Rep #3**	**Rep #1**	**Rep #2**	**Rep #3**
1	2.17	2.22	2.05	14.03	14.17	16.60
2	1.65	1.72	1.69	58.46	65.47	59.19
3	1.50	1.51	1.60	160.66	143.13	141.94
4	1.77	1.81	1.83	35.37	38.74	37.50
5	1.62	1.74	1.48	105.52	99.02	105.25
6	1.88	1.95	1.94	57.24	57.21	48.05
7	1.72	1.58	1.68	66.30	66.10	65.84
8	1.92	1.92	1.80	118.13	115.98	110.04
9	1.92	1.88	1.83	60.06	63.87	59.62
10	1.52	1.59	1.59	449.60	426.02	419.69
11	1.61	1.62	1.69	56.48	57.82	54.83
12	1.97	1.79	1.84	146.64	142.22	145.86
13	1.80	1.80	1.74	293.31	271.89	242.77
14	1.57	1.62	1.55	161.38	151.29	160.97
15	1.57	1.63	1.60	104.85	105.37	99.28
16	1.45	1.45	1.49	343.01	350.54	356.23
17	1.92	1.90	1.82	151.56	165.71	145.00
18	1.65	1.65	1.68	130.24	120.71	130.12

Tabla 31: Resultados de los experimentos

4.4.4 Resultados: ARPD

4.4.4.1 Análisis de los resultados

En la Tabla 32, correspondiente a un ANOVA de una vía, se puede observar que existe una diferencia significativa entre la varianza entre tratamientos y la varianza intra tratamientos, por lo que las variaciones entre las respuestas obtenidas para cada tratamiento parecen ser significativamente diferentes, ya que no están confundidas con el término de error. En la práctica podemos ver que el ratio F es mucho mayor que 1. Sin embargo con el ANOVA de una vía no conocemos la/s fuente/s de variación.

	SS	DF	MS	Ratio F	Valor P
Entre	1.536	17	0.09035	26.05	<0.001
Intra	0.1249	36	0.003469		
TOTAL	1.661	53			

Tabla 32: ANOVA de una vía para promedios de la respuesta ARPD, nivel significación: 0.1

Con el objeto de determinar las fuentes de variación hemos realizado una ANOVA de 1 grado de libertad. Los resultados se muestran en la siguiente tabla:

Factor	SS	DF	MS	ratio F	l-hat	Valor P	% c.
P (4 v 12)	0.9637	1	0.9637	277.8	-0.3272	<0.001	60.7
it_m (5 v 15)	0.2483	1	0.2483	71.6	-0.1661	<0.001	15.6
it_s (3 v 5)	0.198	1	0.198	57.09	-0.1211	<0.001	12.5
K (2 v 4)	0.0318	1	0.0318	9.169	-0.05944	0.005	2.0
B (2 v 8)	0.0152	1	0.01521	4.385	-0.04111	0.043	1.0
$pool$ (0.3 v 0.7)	0.0049	1	0.0049	1.413	-0.02333	0.242	0.3
β (0.2 v 0.4)	0.0004	1	0.0004694	0.1353	0.007222	0.715	0.0
α (0.2 v 0.4)	0.0001	1	0.0001361	0.03924	-0.003889	0.844	0.0
Within (error)	0.1249	36	0.003469				
TOTAL	**1.587**	**44**					

Tabla 33: ANOVA de un grado de libertad para ARPD

Nota: Las columnas de la tabla ANOVA de un grado de libertad definen:

- El factor que se varía en el tratamiento. Entre paréntesis los niveles de dichos factores.
- SS: Suma de cuadrados.
- DF: Grados de libertad.
- MS: Media cuadrática.
- Ratio F: Relación entre las variaciones observadas en las distintas réplicas y las que se observan entre tratamientos diferentes. Cuanto mayor sea, más significativas son las diferencias entre tratamientos.

- l-hat: Efecto estimado del contraste. Es la variación en la respuesta media de los niveles que están siendo comparados.
- Valor P: Representa el nivel de significación crítico.
- % c.: Porcentaje de contribución de cada factor a la suma de cuadrados.

Por tanto, los efectos significativos para la respuesta estudiada, ARPD, son por orden: P (número de procesos), it_m (iteraciones en el proceso jefe), it_s (iteraciones en los procesos trabajadores), K (número de soluciones para generar soluciones vecinas) y B (número de soluciones intercambiadas en cada mensaje), mientras que los no significativos son: $pool$ (número de soluciones almacenadas en la memoria central), β y α (parámetros que definen el umbral de aceptación de soluciones cercanas a la mejor).

Por otra parte hay que puntualizar que las hipótesis de partida del ANOVA se cumplen en este caso, cumpliéndose las hipótesis de normalidad y homocedasticidad e independencia de las observaciones. Dichos resultados han sido omitidos al carecer de mayor interés.

Se observa que los parámetros que influyen en la calidad de las soluciones obtenidas, según ARPD, son el número de procesadores empleado y en mucho menor medida, el número de iteraciones, tanto en el proceso jefe como en los procesos trabajadores. Se observa, además, que en estos valores, tanto los parámetros del umbral, alfa y beta, como el número de soluciones almacenadas en la memoria central o *Pool* no tienen influencia sobre la calidad de las soluciones.

4.4.4.2 Optimización

Teniendo en cuenta que el diseño empleado no es un diseño factorial completo, la elección del nivel adecuado para cada parámetro puede no corresponderse con ninguno de los tratamientos que hemos realizado. Adicionalmente parece interesante explotar la información obtenida de los experimentos, concretamente sobre la linealidad de la respuesta en función del nivel de los parámetros.

Para obtener dicha información vamos a mostrar los resultados obtenidos a los gráficos de los efectos principales del análisis de las medias (ANOM).

En la Figura 43 y siguientes se muestra el gráfico de efectos principales para los parámetros no significativos: α, β y $pool$.

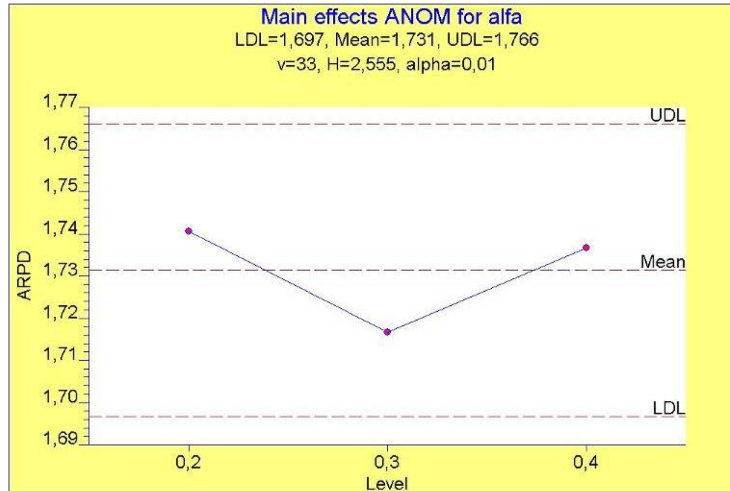


Figura 43: Gráfico de efectos principales para α

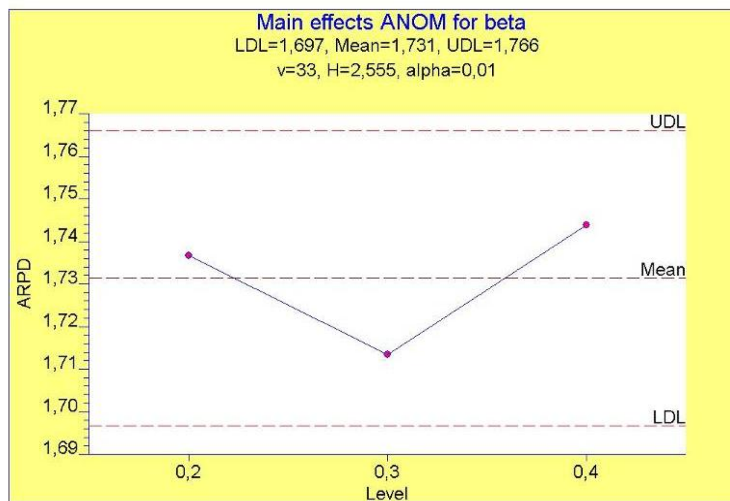


Figura 44: Gráfico de efectos principales para β

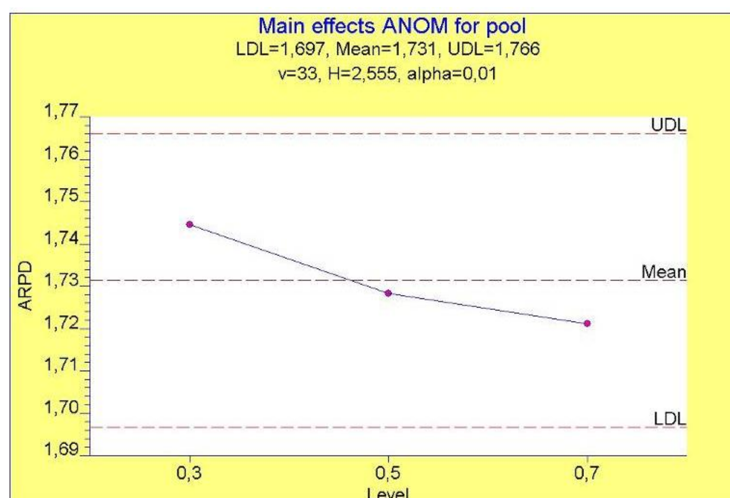


Figura 45: Gráfico de efectos principales para *pool*

Como se derivó del ANOVA anterior, dichos parámetros están dentro de los límites de control, por lo que su efecto no es significativo. Por ello seleccionaremos los valores que más convengan. Desde el punto de vista de ARPD, los niveles más favorables son los niveles intermedios 0.3 y 0.3 para los dos primeros factores (α, β) y el nivel superior, 0.7, para *pool*.

Para el resto de parámetros los resultados se muestran en la Figura 46 y siguientes.

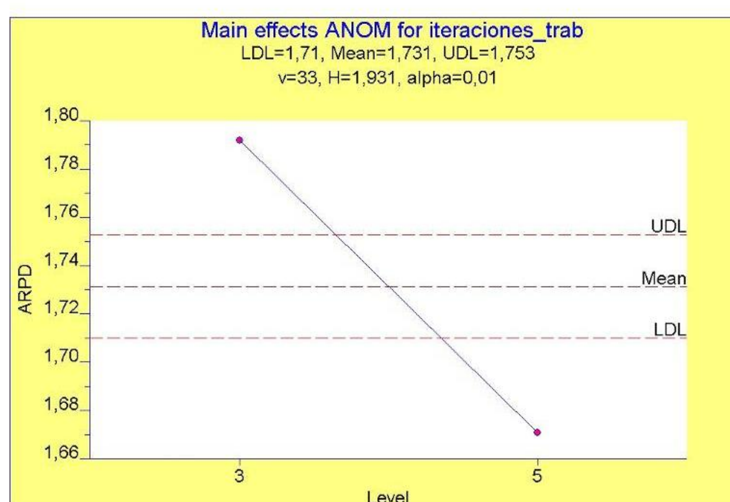


Figura 46: Gráfico de efectos principales para las iteraciones en los trabajadores

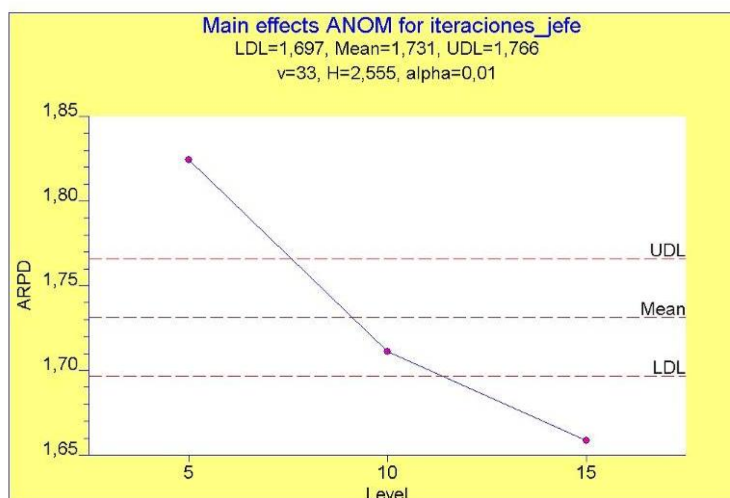


Figura 47: Gráfico de efectos principales para las iteraciones en el proceso jefe

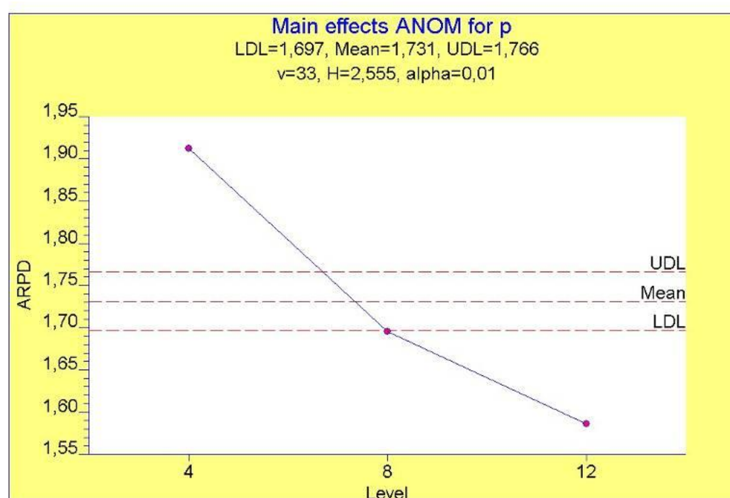


Figura 48: Gráfico de efectos principales para el número de procesos

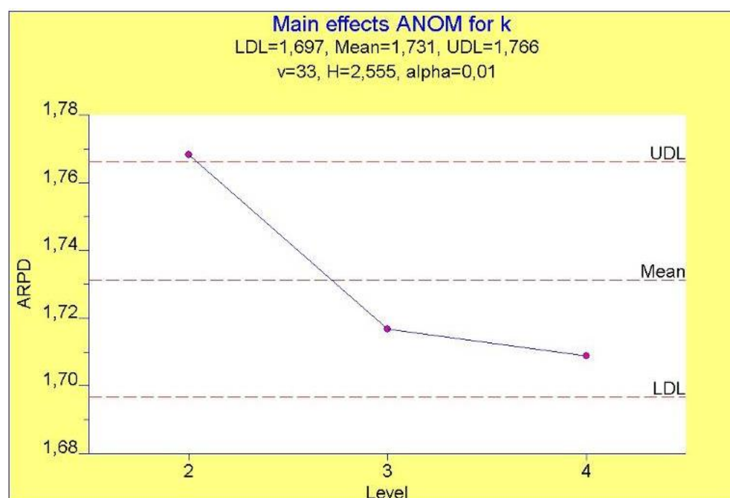


Figura 49: Gráfico de efectos principales para K

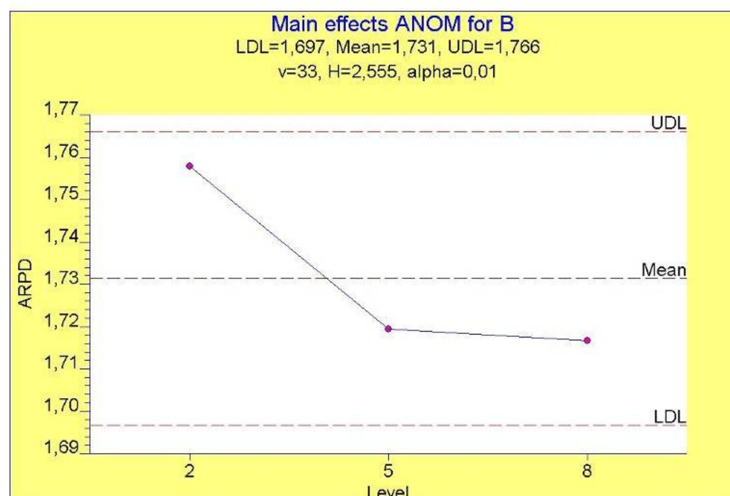


Figura 50: Gráfico de efectos principales para B

Como era de esperar, el análisis es muy parecido al ANOVA. Sin embargo podemos ver cómo el ANOM para el factor B resulta dentro de los límites de control, por lo que su efecto puede confundirse con el error. Según los efectos medios, el parámetro B está dentro de los límites de control, sin embargo es significativo en el análisis ANOVA (debido al efecto de la variabilidad).

Atendiendo a los valores medios, seleccionamos aquellos que minimicen el ARPD. Por lo que el nivel alto para todos estos factores parece ser el más adecuado.

A modo de resumen se muestran los resultados en la siguiente tabla:

	α	β	it_s	it_m	K	p	B	$Pool$
Niveles	0.2	0.2	3	5	2	4	2	0.3
	0.3	0.3	5	10	3	8	5	0.5
	0.4	0.4		15	4	12	8	0.7
Nivel óptimo	0.3	0.3	5	15	4	12	8	0.7
Significativo	NO	NO	SI	SI	SI	SI	SI	NO

Tabla 34: Resumen de ANOVA para ARPD

Estos resultados son completamente coherentes con el diseño de nuestro algoritmo paralelo. Una mayor intensificación y diversificación de la búsqueda mejorará la calidad de las soluciones obtenidas.

El aumento del número de procesos mejorará la diversificación de la búsqueda, debido a que cada uno de ellos realiza una exploración partiendo de una solución diferente, y además está diseñado para no explorar soluciones que están siendo exploradas por otros procesos. De este modo, se aumenta la región de espacio de soluciones que está siendo explorada por el conjunto de procesos y se evita el estancamiento en mínimos locales.

El aumento en el parámetro K también incrementa la zona del espacio explorada por cada proceso, lo que lleva, como en el caso del aumento del número de procesos a un aumento de la región total explorada del espacio.

Por otro lado, la intensificación de la búsqueda, que lleva a mejorar la calidad de las mejores soluciones encontradas hasta el momento, se mejora incrementando el número de iteraciones tanto globales como las realizadas por cada proceso. El resultado es similar al comentado para la diversificación de la búsqueda de soluciones. El incremento del número de iteraciones en los procesos trabajadores, incrementa la intensificación de la búsqueda por parte de cada uno de los procesos, mientras que la intensificación a nivel global se consigue incrementando el número total de iteraciones realizadas por el proceso jefe o master.

Los parámetros B, número de soluciones intercambiadas entre los procesos en cada mensaje y $pool$, número de soluciones almacenadas en la memoria central, están relacionadas con el conocimiento que pueden tener los procesos de la exploración en otras zonas del espacio de soluciones. Su influencia es algo más difusa sobre la intensificación que sobre la diversificación de la búsqueda. Sin embargo, es claro que un aumento de este conocimiento, mejorará la calidad de las soluciones en el sentido de diversificar la búsqueda, ya que se evita que los procesos atraviesen zonas del espacio que están explorando otros procesos. Por otro lado, hay que advertir que, tras la primera iteración, la memoria central sirve para tomar nuevas soluciones de partida, luego cuanto mayor sea la cantidad de soluciones almacena-

das, hay más posibilidades de continuar la búsqueda hacia zonas alejadas del mínimo actual.

Los parámetros que rigen el umbral de aceptación de soluciones tienen una importancia menor que los anteriores. Hay que recordar que el umbral de aceptación de soluciones alejadas del mínimo local, τ_i, va disminuyendo en cada iteración según la expresión (21):

$$\lambda = \frac{\alpha}{1+\beta};\ \alpha_i = \lambda\alpha_{i-1}; \tau_i = (1+\alpha_i)mk_{best} \quad (21)$$

Siendo mk_{best} el mejor valor de la función objetivo que tratamos de minimizar, *makespan*, encontrado por este proceso en esta iteración.

En este caso, el diseño experimental ha ayudado a seleccionar unos valores iniciales apropiados para α y para β, y a descubrir que la importancia de estos dos parámetros es muy inferior a la del resto, bajo el objetivo de minimizar el ARPD.

4.4.5 Resultados: Tiempo de ejecución

4.4.5.1 Análisis de los resultados

Se ha realizado el mismo análisis para el tiempo de ejecución del algoritmo que el mostrado en el apartado anterior para la calidad de las soluciones. En primer lugar, el ANOVA de una vía.

En el caso del tiempo, el ANOVA de una vía, resumido en la Tabla 35, muestra diferencias aun mayores entre la varianza entre tratamientos y la varianza en los tratamientos con un ratio F todavía mayor que en el caso del ARPD.

Source	SS	DF	MS	F-ratio		P-value
Between	624900	17	36760	499.7	*	<0.001
Within (error)	2648	36	73.56			
TOTAL	627600	53				

Tabla 35: ANOVA de una vía para el tiempo con nivel alpha = 0.1

Para estudiar la influencia de los diferentes niveles de los factores, se ha realizado un ANOVA de un grado de libertad (ver Tabla 36).

Source	SS	DF	MS	Ratio F		l-hat	P-value	% c.
it_s (3 v 5)	214100	1	214100	2910	*	125.9	<0.001	36.3
it_m (5 v 15)	172300	1	172300	2342	*	138.3	<0.001	29.2
K (2 v 4)	119600	1	119600	1626	*	115.3	<0.001	20.3
β (0.2 v 0.4)	70280	1	70280	955.5	*	-88.37	<0.001	11.9
$pool$ (0.3 v 0.7)	8662	1	8662	117.8	*	-31.02	<0.001	1.5
B (2 v 8)	1555	1	1555	21.14	*	-13.15	<0.001	0.3
P (4 v 12)	1416	1	1416	19.25	*	12.54	<0.001	0.2
α (0.2 v 0.4)	59.42	1	59.42	0.8078		2.569	0.375	0
Within (error)	2648	36	73.56					
TOTAL	590600	44						

Tabla 36: ANOVA de un grado de libertad para el tiempo de ejecución del algoritmo

En este caso, únicamente el parámetro α se muestra como no significativo, luego en el caso del tiempo de ejecución, pequeños cambios en los parámetros introducen grandes variaciones en la variable de respuesta. El estudio de residuales muestra, igual que en el caso de ARPD, que se cumplen las hipótesis de partida del ANOVA en cuanto a normalidad, variación de los residuos con respecto al tiempo y frente a los valores ajustados.

4.4.5.2 Interpretación de los resultados

Los resultados obtenidos en cuanto al tiempo de búsqueda indican que este es más sensible a pequeñas variaciones en el valor de los parámetros y que estos valores son, en general opuestos a los que ofrecen el mejor valor de ARPD. Esto coincide con lo esperado en la práctica: una exploración más larga en el tiempo conduce, en general, a encontrar soluciones de mayor calidad.

Únicamente habrá de tenerse en cuenta que la mejora de la calidad de las soluciones empieza a ser cada vez menor transcurrido un tiempo de exploración, por lo que conviene llegar a un compromiso entre este y la calidad de las soluciones.

4.4.5.3 Optimización

Nuevamente, se presentan los resultados obtenidos con los gráficos de efectos principales para analizar la linealidad de la variación de la respuesta con los parámetros y el valor más adecuado de los mismos.

El análisis ANOM para el parámetro α se muestra en la Figura 51. Dicho parámetro aparece fuera de los límites de control, aunque en el ANOVA aparecía como no significativo. Aun así, el valor más favorable es el intermedio, 0.3. Aparte de esto, el parámetro se muestra fuertemente no lineal.

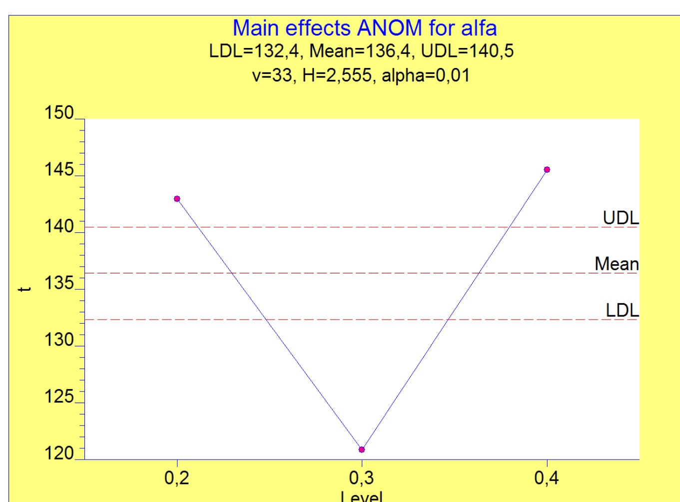


Figura 51: Gráfico de efectos principales para α

El análisis ANOM para el resto de parámetros se muestra en las figuras siguientes:

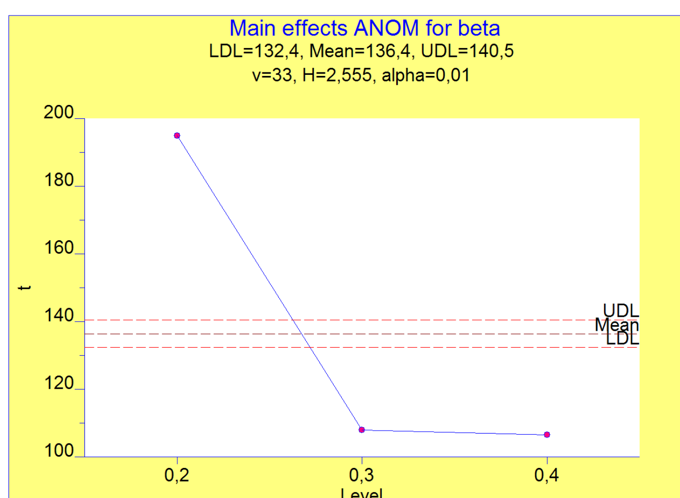


Figura 52: Gráfico de efectos principales para β

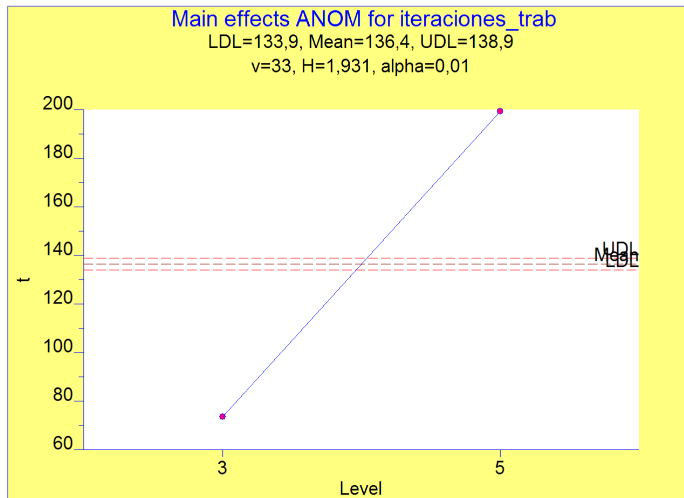


Figura 53: Gráfico de efectos principales para iteraciones en trabajadores

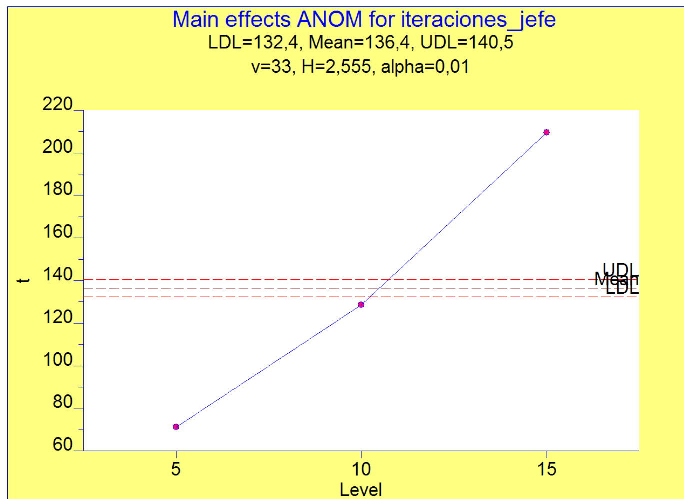


Figura 54: Gráfico de efectos principales para iteraciones en proceso jefe

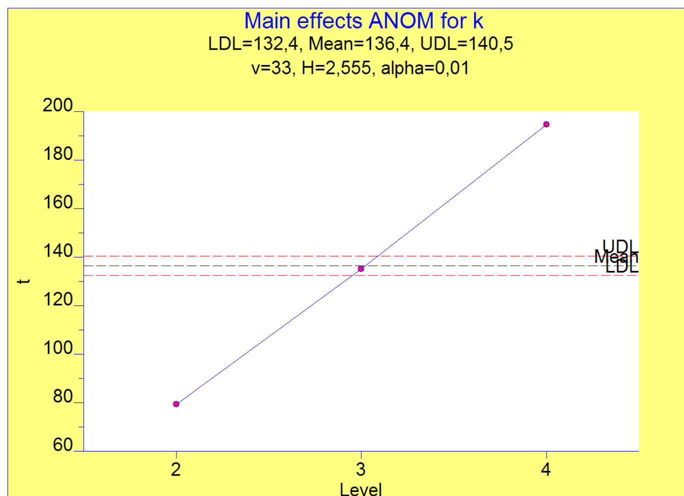


Figura 55: Gráfico de efectos principales para K

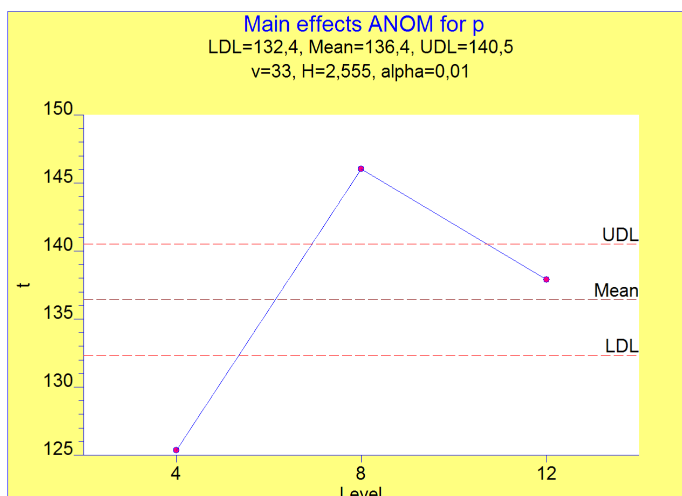


Figura 56: Gráfico de efectos principales para número total de procesos

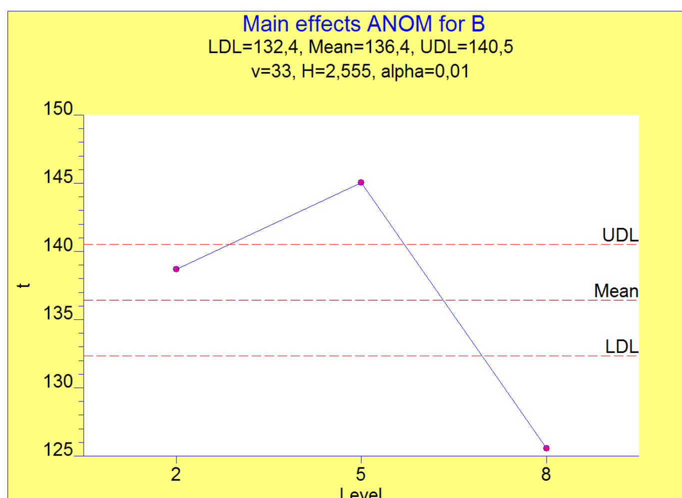


Figura 57: Gráfico de efectos principales para B

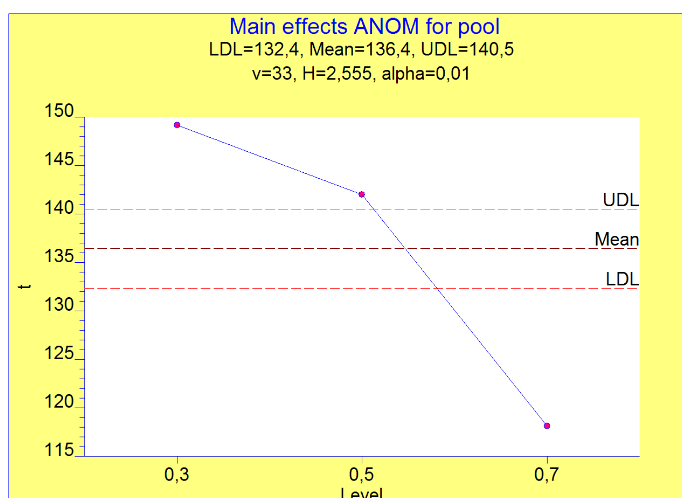


Figura 58: Gráfico de efectos principales para el tamaño de la memoria central

Se observan todos los parámetros fuera de los límites de control, lo que confirma que todos, salvo α, tienen un efecto importante sobre el tiempo de ejecución del algoritmo.

Por otro lado, los valores óptimos para el tiempo de ejecución son, en general, los opuestos a los obtenidos para ARPD. Esto era esperable en el caso de parámetros como el número de iteraciones, tanto en el proceso jefe como en los trabajadores, pero no así en el caso del parámetro K o del número de procesadores empleado.

En el caso de K, el número de soluciones que se toma en cada iteración para generar soluciones vecinas, no es un resultado tan evidente como los anteriores. Confirma que este es un parámetro que tiene que ver, como apuntaron los autores del algoritmo CLM, con la agresividad o intensidad de la búsqueda.

En el caso del número de procesadores, tiene que ver con dos efectos: el conocimiento de zonas nuevas del espacio de soluciones al disponer de más puntos iniciales de partida para cada uno de los procesos y una pérdida de eficiencia del algoritmo paralelo, que se produce debido a los tiempos de espera necesarios para sincronizar el intercambio de soluciones entre un número cada vez mayor de procesadores.

Como se hizo en el caso del ARPD, se han tomado los mejores valores según el tiempo, que se muestran en la Tabla 37, resumidos con los resultados del análisis ANOVA:

	α	β	*it_s*	*it_m*	*K*	*p*	*B*	*pool*
Niveles	0.2	0.2	3	5	2	4	2	0.3
	0.3	0.3	5	10	3	8	5	0.5
	0.4	0.4		15	4	12	8	0.7
Valor óptimo	0.3	0.4 ó 0.3	3	5	2	4	8	0.7
Significativo	NO	SI	SI	SI	SI	SI	SI	SI

Tabla 37: Resumen del análisis de varianza para el tiempo.

En la Tabla 38, mostramos los niveles óptimos de cada parámetro para tiempo y calidad de soluciones:

	α	β	it_s	it_m	K	p	B	$pool$
Nivel óptimo para calidad	0.3	0.3	5	15	4	12	8	0.7
Nivel óptimo para tiempo	0.3	0.3 ó 0.4	3	5	2	4	8	0.7

Tabla 38: Niveles óptimos de los parámetros

Estos resultados nos llevan a elegir para un análisis más detallado los parámetros iteraciones, K y número de procesos para averiguar cuáles de ellos son los más favorables. Sus efectos principales para tiempo y ARPD se muestran en la Tabla 40.

Según esto, los valores que alcanzan un mejor compromiso entre tiempo de ejecución y calidad de soluciones son los que se muestran en la Tabla 39:

Parámetro	Iteraciones Trabajadores	Iteraciones Jefe	Procesos	K
Valor	3	10	12	3

Tabla 39: Valores de compromiso ARPD/tiempo para iteraciones, número de procesos y K

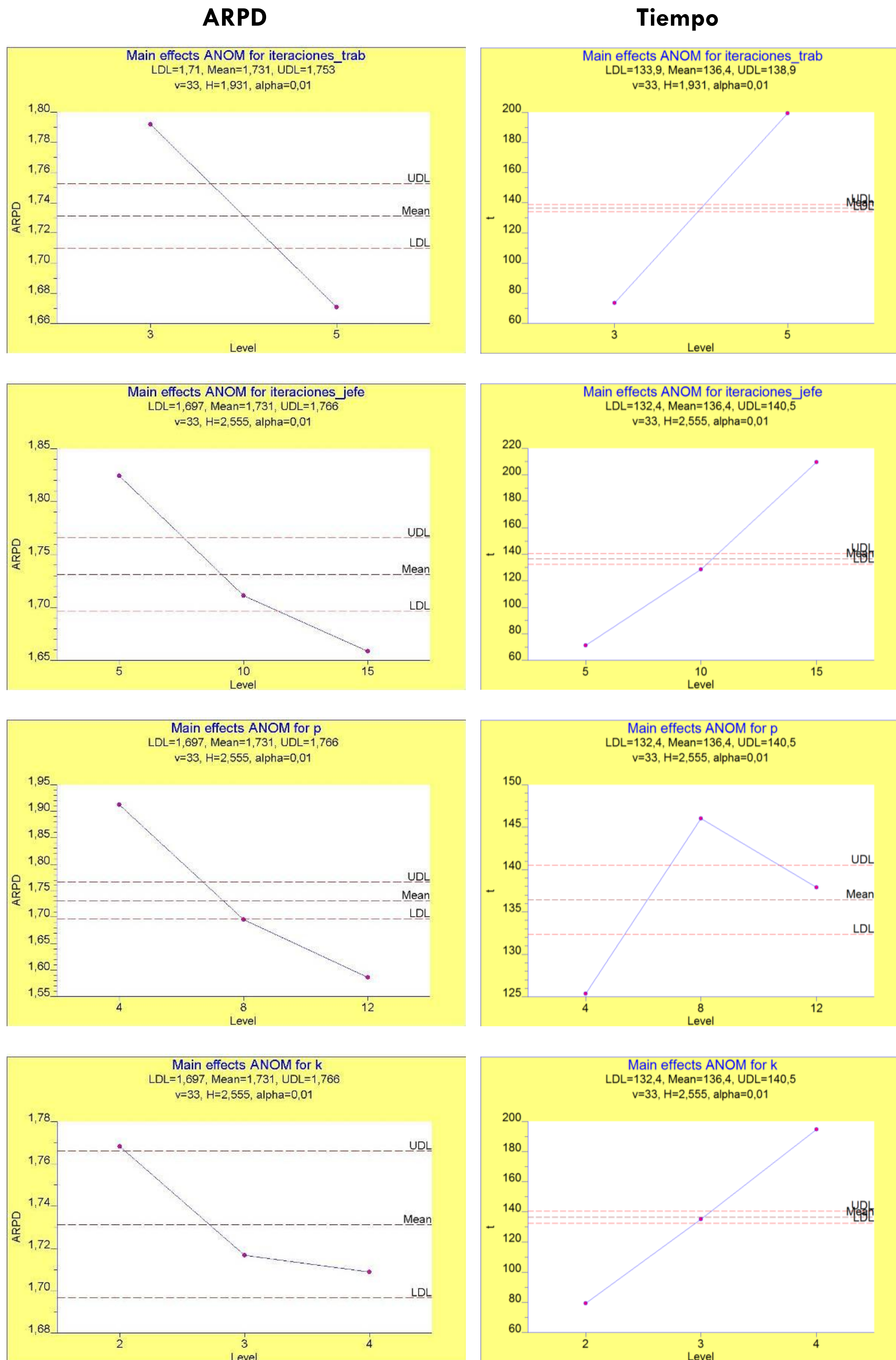


Tabla 40: Efectos principales para iteraciones, número de procesos y K en cuanto a ARPD y tiempo

4.5 Conclusiones

En este capítulo se han diseñado dos versiones de un algoritmo paralelo de búsqueda local basado en el algoritmo CLM propuesto por Ghosh y Sierksma (2002). Para el diseño de los algoritmos se han tenido en cuenta las aportaciones de numerosos autores en el campo del diseño de algoritmos paralelos. Estas aportaciones han influido en el diseño de la arquitectura de los algoritmos, en cuanto al modelo maestro-esclavo modificado, la forma en la que se almacenan las soluciones encontradas (para superar el problema de no disponer de un sistema con una memoria central, sino un sistema con memoria distribuida) y la forma en la que el proceso central o jefe coordina la exploración que realiza el resto.

Se ha desarrollado sobre estos algoritmos una experimentación de la que se han obtenido resultados sobre un gran número de mediciones. Por un lado sobre la calidad de las soluciones en un tiempo de exploración razonable. Por otro lado, sobre la eficiencia del algoritmo paralelo comparando el tiempo de ejecución con varios procesadores con el tiempo de ejecución con un solo procesador. Por último, se ha estudiado la eficiencia incremental generalizada, donde se observan los mejores resultados de los algoritmos.

La comparación de los tiempos de ejecución de cada algoritmo paralelo frente al correspondiente secuencial se ha hecho fijando para el algoritmo secuencial la condición de parada. La calidad de las soluciones obtenidas mediante el algoritmo secuencial con ese criterio de parada ha servido como condición de parada para el algoritmo paralelo. Además, cada algoritmo secuencial, para hacerlo lo más similar posible al correspondiente algoritmo paralelo, se ha ejecutado mediante un proceso jefe y un proceso trabajador funcionando ambos en el mismo procesador.

Se ha observado que las medidas de la eficiencia son menos fiables que las medidas de la eficiencia incremental generalizada, dado que, en esta última, la comparación de los tiempos de ejecución se realiza entre algoritmos que son exactamente iguales, con la misma condición de parada y funcionando cada vez en más procesadores. Por tanto, se puede proponer la eficiencia incremental generalizada como la mejor medida de la escalabilidad de un algoritmo paralelo.

Se ha revisado la literatura en cuanto a las prestaciones de algoritmos para proceso en paralelo completando así a la realizada en el capítulo 3 de este trabajo. Algunos algoritmos están más cercanos a los diseñados en este capítulo y otros son implementaciones más adecuadas a otros tipos de sistema informático. En algunos se consiguen mejores resultados que en los diseñados en este capítulo, mientras que otros han sido claramente superados por las prestaciones de éstos. También se ha observado que muchas de las propuestas de otros autores evitan comparar las prestaciones de su algoritmo en cuanto a eficiencia, limitándose únicamente a constatar su superioridad frente al algoritmo secuencial.

Se ha rediseñado el algoritmo paralelo de grano grueso o *coarse grained* para incrementar la frecuencia de la comunicación entre procesos, para obtener un algoritmo paralelo de grano fino o *fine grained*. Con esta modificación se ha buscado mejorar el tiempo en el que dicho algoritmo obtiene soluciones de buena calidad al incrementar el conocimiento que adquieren los distintos procesos sobre la exploración que llevan a cabo el resto de procesos.

El rediseño del algoritmo paralelo junto con la experimentación desarrollada a continuación para su ajuste ha dado como resultado un algoritmo paralelo más rápido y más eficiente que el anterior, aunque ahora la eficiencia decae más rápidamente por efecto de las esperas que impone la sincronización entre procesos.

De todas formas, la eficiencia del algoritmo CLMpf sigue siendo ligeramente inferior a la del algoritmo CLMpg para cada número de procesadores y para cada dimensión del problema. En cambio, la calidad de soluciones y el tiempo de ejecución con el algoritmo CLMpf es mejor que con el algoritmo CLMpg, luego en este caso resulta ventajoso renunciar mínimamente a la escalabilidad del algoritmo a cambio de obtener soluciones de calidad en un corto espacio de tiempo. Esta es una situación ideal cuando la plataforma de la que se dispone tiene un pequeño número de procesadores.

5 Ajuste de parámetros en algoritmos paralelos

5.1 Introducción

En el capítulo anterior, se han diseñado dos versiones en paralelo de un algoritmo de búsqueda local basado en el CLM de Ghosh y Sierksma (2002). En el primero de ellos, de grano grueso, se ha puesto de manifiesto que una mayor colaboración entre procesos obtiene mejores resultados en cuanto a tiempo de exploración que una mayor intensificación de la búsqueda realizada por cada proceso. Por este motivo, se diseñó una segunda versión con mayor comunicación entre procesos, denominado CLMpf.

Ambos diseños han puesto de manifiesto la dificultad de un ajuste óptimo de los parámetros del algoritmo y la necesidad de una metodología para este ajuste de parámetros orientada a un algoritmo paralelo. Hay que tener en cuenta que, al conjunto de parámetros correspondiente a la metaheurística en paralelo que se está diseñando, deben sumarse el conjunto de parámetros que son propios del algoritmo paralelo, como por ejemplo, el número de procesadores, el instante en el que se intercambian datos, o el número de datos intercambiados, lo que hace que el número total de parámetros a considerar sea relativamente elevado. Por ello parece que una metodología adecuada a este tipo de problemas sea necesaria.

Los objetivos que se buscan con el ajuste de parámetros son:

- Reducción del tiempo de exploración del espacio de soluciones necesario para encontrar soluciones de calidad suficiente.

- Mejora de la calidad de las soluciones encontradas.
- Mejora de la eficiencia del algoritmo paralelo.
- Mejora de la robustez del algoritmo frente a:
    - Variaciones en el tamaño de los problemas afrontados.
    - Variaciones en la complejidad de los problemas.
    - Variaciones en el número de procesadores disponibles en el sistema informático.

En el apartado 2 realizamos una revisión de los métodos existentes que pudieran ser aplicables a la resolución del propio problema de la determinación de los parámetros más adecuados para nuestro algoritmo bajo el criterio de eficiencia.

La metodología que se propone en el apartado 3 trata de obtener la configuración más adecuada de los parámetros para la ejecución de un algoritmo paralelo desde el objetivo de maximización de la eficiencia.

Para maximizar la eficiencia del algoritmo, se incluye en el apartado 4 un diseño experimental, complementado en el apartado 5 con un estudio estadístico de la misma.

El apartado 6 y último antes de las conclusiones, contiene un estudio desde el punto de vista de la robustez del algoritmo, atendiendo a dos enfoques diferentes: el primero atendiendo al tamaño del problema y el segundo atendiendo al número de procesadores disponibles.

5.2 Estado de la cuestión

Las metaheurísticas tratan de encontrar, en un tiempo reducido, soluciones de calidad para problemas de diverso tipo. En algunos casos, pueden existir algunos métodos exactos para encontrar la solución óptima, pero dada su lentitud en problemas de cierta magnitud (como en el caso del problema de secuenciación en flujo uniforme, los cuales requieren un esfuerzo computacional en tiempo que no resulta viable a partir de tres máquinas). En otros problemas no es posible encontrar una solución exacta y debe recurrirse a estos métodos aproximados.

Por otra parte hay que tener en cuenta que el funcionamiento de estos métodos depende de un cierto número de parámetros, los cuales han de ser establecidos para obtener las mejores características del algoritmo empleado. Dichos parámetros pueden agruparse en tres: aquellos relativos al problema, relativos al algoritmo y aquellos relativos al entorno de aplicación o de experimentación (Barr *et al.,* 1995).

Las prestaciones de estos métodos suelen medirse en función de la calidad de las soluciones obtenidas, medida según el valor alcanzado de la función objetivo, comparado con los óptimos o valores de las mejores cotas conocidas, o de la eficacia computacional, medida según el tiempo necesario para encontrar soluciones de calidad. Otras medidas de la calidad del algoritmo son citadas por Barr *et al.* (1995) y son: rapidez del método para encontrar una buena solución, robustez del método, lejanía del óptimo de las soluciones encontradas con facilidad y el compromiso entre admisibilidad y calidad de las soluciones, agrupándolas todas en tres: calidad de la solución, esfuerzo de computación y robustez.

El diseño que estamos afrontando en este capítulo de una metodología de ajuste de parámetros para algoritmos paralelos debe tener en cuenta los objetivos relacionados con el algoritmo paralelo:

- La eficiencia máxima que se puede conseguir con el algoritmo paralelo. Ésta se consigue con el menor número de procesadores.
- El declive de la eficiencia conforme aumenta el número de procesadores. Ésta es debida a la fracción serie del algoritmo y a los tiempos de espera que implica la comunicación entre procesos.
- Y además debe ser aplicable a diferentes instancias de problemas (con diferentes tamaños) y a diferentes tamaños de clúster.

Todo esto debería lograrse con una experimentación que pudiera obtener resultados en un periodo de tiempo razonable.

De manera similar a las observaciones de Barr *et al.* (1995) para algoritmos secuenciales, Belkadi *et al.* en dos estudios, Belkadi *et al.* (2006b) y Belkadi *et al.* (2006c), observaron que tanto la eficiencia del algoritmo paralelo como la calidad de las

soluciones obtenidas dependían fuertemente de los parámetros del algoritmo y de la frecuencia con la que los procesos que lo componen intercambiaban soluciones.

Sin embargo, la experimentación para obtener el mejor balance entre los parámetros que afectan a algoritmos secuenciales se encuentra ampliamente documentada en la literatura. Para una revisión de los trabajos en este campo se puede consultar Adenso-Díaz y Laguna (2006). Los mismos autores propusieron una metodología para el ajuste de parámetros en metaheurísticas basada en una combinación de búsqueda local y diseño de experimentos, ensamblada en la aplicación denominada CALIBRA. Dicha aplicación, diseñada para entornos Windows, trata el algoritmo cuyos parámetros se están ajustando como una caja negra. Los autores consiguen ajustar hasta cinco parámetros con su aplicación. En el caso de la optimización de algoritmos paralelos, en la cual pueden intervenir un número elevado de parámetros, parece una opción limitada. Por otra parte Gupta *et al.* (2000) propusieron un método de selección de la heurística más adecuada.

Además de los estudios descritos anteriormente es posible emplear diferentes estrategias que habitualmente se emplean en la optimización de parámetros. Entre ellas destacan: metodologías OFAT, la optimización global y el diseño experimental, que a continuación se describen.

5.2.1 Metodologías OFAT

El ajuste de parámetros denominado en la literatura como metodología OFAT (*One Factor At a Time*) consiste en llevar a cabo los experimentos fijando el valor de todos los factores menos uno cada vez (Montgomery, 2005). De esta manera, se obtiene el mejor valor para el parámetro no fijado, y se pasa al siguiente factor para repetir ahora el experimento con este factor variable.

La dificultad que plantea este tipo de metodología es, principalmente, que oculta las posibles interacciones entre los valores de los parámetros. De este modo, no es posible conocer todas las combinaciones de valores de los parámetros que conducirían a soluciones de mejor calidad. En definitiva, este tipo de metodologías no puede garantizar que se llegue a la combinación óptima de los parámetros de exploración, sino que sólo garantiza la consecución de un óptimo local.

Por otro lado hay que resaltar que el coste de este tipo de experimentación es demasiado elevado, lo que no lo hace muy recomendable para aplicarlo en la optimización de parámetros de algoritmos paralelos.

5.2.2 Metodologías basadas en la optimización global

Existen herramientas – generalmente comerciales – que permiten tratar el problema del ajuste de parámetros del algoritmo como un problema de optimización en sí mismo. En este caso, las soluciones al problema de optimización serán las diferentes combinaciones de parámetros. El principal inconveniente sobre el empleo de este tipo de métodos es que generalmente se conoce poca información sobre el método de optimización empleado, con lo que suelen ser tratados como una caja negra.

El funcionamiento es similar al de un problema de optimización. En este caso, la herramienta toma como función objetivo el propio algoritmo, como variables los factores que se desean ajustar, y las variables de respuesta las que miden la calidad del ajuste. Normalmente es la misma que mide la calidad de la función objetivo del problema que resuelve el algoritmo.

Algunas herramientas de optimización que pueden emplearse para este fin son, entre otras, Evolver (Palisade, 2007), Genocop (Michalewicz, 1993), GAUL (gaul.sourceforge.net) y Optquest (Opttek Systems, 2007).

Evolver es una herramienta comercial para optimización global, que emplea algoritmos genéticos. Se implementa sobre una hoja de cálculo de Excel. En ella se plantea un modelo de la función cuyo óptimo se desea encontrar. En el caso del ajuste de parámetros, las variables de entrada son los parámetros. La función que se optimiza es el resultado del algoritmo. Al igual que el resto de herramientas comerciales, implica que la función objetivo, es decir, el algoritmo, no debe ser demasiado complejo. En ese caso, cada evaluación del mismo supondría un tiempo de ejecución que haría intratable el problema.

GENOCOP III (*Genetic Algorithm for Constrained Problems*) es una aplicación, en este caso gratuita, para optimización global mediante algoritmos genéticos, orientado a la optimización de una función con cualquier conjunto de restricciones lineales (Mi-

chalewicz, 1993). Los parámetros del sistema se introducen mediante archivos de texto.

GAUL (*Genetic Algorithm Utility Library*) es otra librería basada en algoritmos genéticos, que puede emplearse para el ajuste de parámetros. En nuestro caso, tiene la desventaja de estar muy orientada a aplicaciones que usan algoritmos genéticos o evolutivos pero posee la ventaja de admitir multiproceso mediante MPI.

OptQuest OCL es una librería para optimización global, que emplea búsqueda local y *scatter search*. La herramienta podría ajustar de forma automática los parámetros del algoritmo para el conjunto de problemas de entrenamiento que se le ha presentado.

En la revisión de herramientas software de optimización global de Pintér (1996), algunas de las herramientas que se presentan (GENOCOP III, MAGESTIC Data Fitting by Global Optimization o UFO *Universal Functional Optimization* entre otros) son adecuadas a esta técnica de ajuste de parámetros.

El principal inconveniente de este tipo de técnicas es la necesidad de adaptarlas al ajuste de parámetros en algoritmos paralelos. Algunas sólo se emplean bajo sistemas operativos concretos, tienen muy limitado el número de parámetros o implican la reprogramación del algoritmo paralelo para que interactúe con cada herramienta concreta de optimización.

5.2.3 Metodologías basadas en diseño estadístico de experimentos

Como hemos comentado anteriormente, algunas de las metodologías no son adecuadas principalmente debido al elevado consumo de recursos e ineficiencia en la optimización, como es el caso de la estrategia OFAT descrita anteriormente. Por otra parte otros métodos, como los descritos en el apartado 5.2.2 de optimización global, funcionan como una caja negra y realmente es difícil comprender su funcionamiento. Aunque podría ser una alternativa adecuada, su implementación en entorno de arquitecturas en paralelo puede resultar en ocasiones bastante complicada. Por estas razones, las metodologías basadas en el diseño de experimentos o DOE (*Design of Experiments*) pueden ser la alternativa con soporte científico más general

y adecuada a la optimización de parámetros en algoritmos paralelos. Dicha metodología se describe brevemente a continuación.

De manera general, las razones principales para el empleo de experimentos factoriales son las siguientes (ver Gunst y Mason, 1991):

- Posibilidad de cuantificar las interacciones entre factores.
- Optimización de los recursos empleados.
- Posibilidad de optimización de los factores.
- Se basan en procedimientos estadísticos.

Las metodologías basadas en el diseño estadístico de experimentos tratan de encontrar la mejor combinación de valores de los parámetros de acuerdo a una determinada respuesta del sistema. En primer lugar, podría pensarse en diseños de tipo factorial completo. Es decir, cada uno de los posibles valores de parámetros debería combinarse con cada una de las posibles combinaciones de valores del resto de parámetros. Con ello, se aseguraría poner de manifiesto todas las posibles interacciones entre valores de los parámetros, llegando a la mejor combinación posible de valores. Evidentemente, en caso de que hubiera muchos niveles diferentes para los valores de parámetros, este tipo de experimentación sería inaplicable en la práctica. Este es el caso del ajuste de parámetros en algoritmos paralelos. Para estos casos suelen emplearse, normalmente, diseños de experimentos de tipo factorial fraccional, en los que no sea necesaria la experimentación de todas las posibles combinaciones de valores de los parámetros. En este grupo se encuentran los diseños factoriales fraccionales, diseños de Plackett-Burman o matrices Lx de diseño robusto de Taguchi.

Según Montgomery (2005), las fases que componen el diseño de experimentos son las siguientes:

1. Reconocimiento y declaración del problema
2. Selección de la variable de respuesta
3. Elección de los factores, niveles y rangos.
4. Elección del diseño experimental

5. Realización del experimento
6. Análisis estadístico de los datos.
7. Conclusiones y recomendaciones.

Según Montgomery (2005), en la práctica, la elección de los factores, niveles y rangos y la selección de la variable de respuesta suele hacerse de forma simultánea o incluso al revés de como está indicado. El mismo autor advierte de que se trata de un proceso iterativo, en el que se irá obteniendo información sobre el problema, con lo que se podrán plantear nuevas hipótesis que impliquen rehacer el diseño de experimentos planteado. Todo esto debe hacerse también sin olvidar que los recursos para la experimentación son limitados y por tanto deben establecerse límites a la experimentación basados en los recusos disponibles.

5.2.4 Metodologías de experimentación para ajuste de parámetros en algoritmos paralelos

Según Barr *et al.* (1995) en general, una heurística es valiosa si es rápida, precisa, robusta, simple, de alto impacto, generalizable e innovadora. Teniendo en cuenta estas características, el diseño que estamos afrontando en este capítulo de una metodología de ajuste de parámetros para algoritmos paralelos debe tener en cuenta no solo objetivos relacionados con el algoritmo paralelo:

- La eficiencia máxima que se puede conseguir con el algoritmo paralelo. Ésta se consigue con el menor número de procesadores.
- El declive de la eficiencia conforme aumenta el número de procesadores. Ésta es debida a la fracción serie del algoritmo y a los tiempos de espera que implica la comunicación entre procesos.

Sino también:

- Aplicabilidad a diferentes tamaños de problema.
- Aplicabilidad a diferentes arquitecturas informáticas y a diferentes tamaños de clúster.

Por último, como se ha mencionado anteriormente, todo esto debería lograrse con una experimentación que no se dilatara excesivamente en el tiempo.

Existen escasas referencias en la literatura sobre ajuste de parámetros aplicado a algoritmos paralelos. En Cantú-Paz (2007) se ofrece una revisión sobre el efecto de los distintos parámetros de los algoritmos genéticos paralelos. Entre los parámetros considerados en ese estudio están el tamaño de las poblaciones, el tipo de arquitectura empleada (maestro-esclavo o de múltiples poblaciones), si los procesos intercambian información o no, la frecuencia de la misma o el número de procesadores. No obstante, no hace comentarios sobre una metodología para el ajuste de los parámetros.

5.3 Metodología para el ajuste de parámetros en algoritmos paralelos de búsqueda local para maximizar la eficiencia

5.3.1 Introducción

En el presente trabajo hemos diseñado en primer lugar un algoritmo paralelo de grano grueso adaptado a la arquitectura del sistema propuesto para alcanzar la mejor relación prestaciones-precio (ver apartado 4.3). A continuación, tratando de mejorar sus prestaciones, se ha incrementado el número de intercambios de información entre procesos para mejorar la calidad de las soluciones reduciendo el tiempo de procesamiento mediante un algoritmo paralelo de grano fino (ver apartado 4.4). Aunque el algoritmo de grano fino parece muy prometedor en cuanto a la calidad de soluciones, se produce un empeoramiento de la eficiencia del mismo, debido fundamentalmente a que el mayor número de intercambios de información incrementa los tiempos de espera entre procesos. Por este motivo pensamos que un ajuste adecuado de los parámetros podría obtener resultados más interesantes respecto a la eficiencia del mismo.

5.3.2 Descripción de la metodología

Con la metodología que a continuación se propone, se pretende conseguir un algoritmo paralelo eficiente (ver al respecto el apartado 2.3), teniendo en cuenta

además el compromiso entre criterios de velocidad (tiempo) y calidad de las soluciones. Para ello hemos propuesto una metodología compuesta por dos fases:

- *Fase 1: Optimización de parámetros del algoritmo bajo criterios de tiempo y calidad de soluciones.*

En esta fase inicial se trata de estudiar la respuesta del algoritmo paralelo ante un amplio rango de los diferentes valores de los parámetros. Este estudio inicial nos permite descartar aquellos parámetros que no ejercen ninguna influencia estadísticamente significa sobre las respuestas estudiadas (tiempo y calidad), así como orientar acerca de los valores más adecuados sobre los parámetros. Teniendo en cuenta el elevado número de combinaciones a realizar, pensamos que pudiera ser lo más adecuado realizar un diseño factorial fraccional.

El estudio a realizar en esta fase de la metodología ya se ha realizado en el capítulo anterior de la presente tesis, con lo que el procedimiento detallado, así como las hipótesis tomadas y los datos concretos empleados en la presente fase de la metodología se encuentran detallados en la sección 4.4.

- *Fase 2: Optimización de parámetros del algoritmo bajo criterios de eficiencia*

En la Fase 1 hemos estudiado la influencia de los parámetros del algoritmo paralelo en las respuestas relativas al tiempo y calidad de las soluciones. Además hemos logrado reducir el número de parámetros a tener en cuenta en la siguiente Fase 2 de experimentación. En esta segunda fase nos centramos en un objetivo diferente: la maximización de la eficiencia.

De la misma manera que en la fase anterior nos basamos en las técnicas de diseño experimental para seleccionar aquel más adecuado a nuestro algoritmo. En este caso ya tenemos información acerca de los parámetros que no son relevantes para el tiempo y la calidad de las soluciones, por lo que supondremos que no ejercen influencia sobre la eficiencia y no serán tenidos en cuenta a la hora de optimizar el sistema respecto al criterio de máxima eficiencia. Por otra parte hemos conseguido tener una idea sobre el ámbito más adecuado de los valores de los parámetros, los cuales pueden ser modificados en esta fase para una mejor exploración respecto al

criterio actual. Probablemente sea necesaria la modificación de los niveles de los parámetros así como el número de los mismos.

La descripción detallada de esta fase se encuentra en la siguiente sección, 5.4, mientras que el análisis de sensibilidad sobre los parámetros se encuentra descrito con detalle en la sección 5.6.

5.4 Diseño experimental para la mejora de la eficiencia

5.4.1 Descripción de los factores

En la Tabla 41, se describen brevemente los factores que se han tenido en cuenta en esta fase de la experimentación, ya que todos ellos se han descrito con más detalle en la sección 4.4.3.1.

Parámetro	Descripción
α	Componente del umbral de aceptación de soluciones
β	Componente del umbral de aceptación de soluciones
It_s	Condición de parada: número de iteraciones que realizan los procesos trabajadores antes de detener la búsqueda
K	Número de soluciones que se toman del conjunto disponible para generar soluciones vecinas en cada iteración de los procesos trabajadores
It_m*	Condición de parada: Número de iteraciones que se realizan en el proceso jefe por cada proceso trabajador disponible
np*	Número total de procesos que se van a emplear en el algoritmo paralelo. Está limitado por el total de procesadores de los que consta el clúster donde se realiza la experimentación
B	Número de soluciones que se intercambian en cada iteración entre los procesos trabajadores y entre estos y el proceso jefe
$pool$	Número de soluciones que se almacenan en el proceso jefe. El total de soluciones guardadas en el proceso jefe está gobernado por la expresión 20 (ver apartado 4.4.3.1)

Tabla 41: Parámetros de la búsqueda local

Para el estudio de la eficiencia, se tomarán aparte el número global de iteraciones, It_m y el número de procesadores np.

Para el desarrollo de la nueva experimentación que lleve a obtener los parámetros que hacen máxima la eficiencia del algoritmo paralelo, se hará uso de los resultados de los experimentos llevados a cabo en el capítulo 4. La experimentación orientada a la calidad de soluciones y al tiempo de ejecución ha puesto de manifiesto los valores idóneos de cada uno de los parámetros así como la importancia relativa de los mismos.

En la Tabla 42 se muestran los valores que alcanzaron mejor compromiso para tiempo de ejecución y valor de la función objetivo. Por claridad se presentan los parámetros con nombres descriptivos:

Parámetro	Iteraciones Trabajadores	Iteraciones Jefe	Procesos	K
Valor	3	10	12	3

Tabla 42: Valores de los parámetros para tiempo de ejecución y calidad de soluciones

Debe tenerse en cuenta que la eficiencia de un algoritmo paralelo se va a calcular midiendo el tiempo de ejecución hasta llegar a obtener soluciones con la misma calidad que se obtuvo empleando un solo procesador, tal como se comenta en el capítulo 2. Con objeto de acotar el tiempo de experimentación cuando se ejecutan los problemas de mayor dimensión, con un solo procesador, se ha limitado el número de iteraciones globales, en el proceso jefe, a 10 iteraciones.

En estas condiciones, los parámetros que son significativos para el tiempo de ejecución o para la calidad de las soluciones serán también significativos para la eficiencia del algoritmo paralelo, ya que ésta se ha medido mediante el tiempo de ejecución, manteniendo fijo el valor de la calidad de las soluciones.

5.4.2 Número de niveles

Una vez estudiados los factores significativos para tiempo de ejecución y ARPD, junto con su linealidad, se han reducido el número de niveles de estos factores a 2, siendo los valores seleccionados, aquellos que ofrecieron mejores resultados en cuanto a tiempo y ARPD. Por tanto, haciendo uso de los datos obtenidos en la experimentación anterior, los niveles que se han considerado más adecuado estudiar, son

los mostrados en la Tabla 43. Aparte, se consideran el número de iteraciones en el proceso jefe, $It_m = 10$, y el número de procesadores np, cuyos valores son 3, 6, 9 y 12.

Parámetro	Niveles
α	0.2; 0.3
β	0.3; 0.4
It_s	3; 5
K	2; 4
B	5; 8
$pool$	0.5; 0.9

Tabla 43: Parámetros de la búsqueda local

5.4.3 Selección del tipo de diseño experimental

Aunque existe una gran variedad de alternativas en cuanto a diseños experimentales a aplicar teniendo en cuenta el número de niveles y de factores, se ha seleccionado un diseño Plackett-Burman. Los motivos fundamentales para esta elección son que permite obtener los efectos principales de los factores con un número muy reducido de experimentos, lo cual es de bastante interés ya que el tiempo necesario de computación es una variable que excluye diseños poco saturados o diseños factoriales completos.

Tratamiento	α	β	it_s	K	B	$pool$
1	0.2	0.3	3	2	5	0.5
2	0.2	0.3	3	4	8	0.9
3	0.2	0.4	5	4	8	0.5
4	0.2	0.4	5	2	5	0.9
5	0.3	0.4	3	2	8	0.9
6	0.3	0.4	3	4	5	0.5
7	0.3	0.3	5	4	5	0.9
8	0.3	0.3	5	2	8	0.5

Tabla 44: Conjunto de tratamientos para eficiencia

En la Tabla 44 se muestra la matriz del diseño experimental resultante. it_s se refiere al número de iteraciones en los procesos trabajadores.

5.4.4 Datos

En esta ocasión, los experimentos han incluido también tanto problemas de menor dimensión de la batería de Taillard (1993), que venimos empleando durante toda nuestra experimentación, de 20 trabajos y 20 máquinas, como los de mayor dimensión, 200 y 500 trabajos. Esto forma un total de 10 tipos de instancias diferentes que se estudiarán para cada tratamiento: 20x20, 50x5, 50x10, 50x20, 100x5, 100x10, 100x20, 200x10, 200x20 y 500x20 (trabajos x máquinas).

5.4.5 Respuestas

La respuesta de interés en el estudio actual es la eficiencia del algoritmo paralelo, medida mediante el tiempo de ejecución para una calidad fija de las soluciones obtenidas. Esta medida de calidad se sigue calculando, como en el capítulo anterior, mediante el promedio del porcentaje de desviación relativa, ARPD con respecto a las mejores soluciones conocidas en la literatura para el problema de secuenciación de trabajos en flujo uniforme sin restricciones de capacidad.

5.4.6 Número de replicados

Como en el apartado anterior, el algoritmo comienza desde soluciones aleatorias y diferentes, por lo que se han realizado tres réplicas de cada tratamiento con el objeto de poder estimar la dispersión de los datos. Como en total se consideran 10 tipos de instancias diferentes con 10 instancias cada una, se computarán un total de 100 experimentos para cada uno de los 8 tratamientos propuestos.

5.4.7 Resultados

Con el conjunto de condiciones o tratamientos mostrados en la Tabla 44, se ha procedido a ejecutar el algoritmo propuesto con un proceso jefe y un proceso trabajador funcionando ambos en el mismo equipo con un solo procesador. De esta forma, se dispone de un valor inicial de calidad de las soluciones que nos va a permitir

observar la eficiencia del algoritmo paralelo conforme se vayan añadiendo procesadores.

	Tratamiento							
ARPD(%)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
20x20	2.36	2.10	1.94	2.24	2.28	1.91	1.95	2.33
50x5	0.43	0.35	0.44	0.43	0.51	0.44	0.37	0.40
50x10	3.51	3.29	3.25	3.17	3.54	3.15	3.14	3.13
50x20	4.54	4.30	4.04	4.21	4.50	4.17	4.11	4.11
100x5	0.29	0.23	0.25	0.28	0.26	0.30	0.20	0.23
100x10	1.46	1.44	1.35	1.34	1.44	1.47	1.26	1.47
100x20	4.00	3.76	3.63	3.73	4.00	3.77	3.70	3.68
200x10	0.81	0.89	0.82	0.77	0.82	0.84	0.80	0.78
200x20	3.17	3.09	2.85	2.99	3.24	3.09	2.93	3.00
500x20	1.93	1.68	1.84	1.74	1.85	1.66	1.78	1.77
Promedio	2.25	2.11	2.04	2.09	2.24	2.08	2.02	2.09

Tabla 45: Calidad (ARPD) de las soluciones obtenidas en un procesador con 10 iteraciones globales

En la Tabla 45 se muestran los resultados obtenidos respecto a la calidad de las soluciones.

	Tratamiento							
Tiempo(s)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
20x20	0.07	0.15	0.31	0.11	0.08	0.14	0.25	0.11
50x5	0.41	1.30	2.82	1.28	0.42	1.13	2.50	0.76
50x10	0.85	1.66	2.42	1.65	1.00	1.71	2.68	1.77
50x20	2.59	4.54	8.32	4.03	2.84	4.37	7.40	4.52
100x5	13.95	55.87	116.35	36.30	22.16	35.12	151.07	79.11
100x10	7.96	16.08	37.32	14.54	11.95	21.30	57.16	20.46
100x20	19.64	36.15	58.90	33.93	24.03	41.93	60.89	39.21
200x10	570.76	431.68	1232.22	858.37	539.11	651.21	2060.71	991.96
200x20	455.28	568.17	731.32	487.22	856.13	930.76	1198.93	931.49
500x20	26914.02	45770.33	50092.77	37382.93	87895.36	81570.03	47581.81	98025.51
Total	**27985.52**	**46885.92**	**52282.76**	**38820.36**	**89353.09**	**83257.71**	**51123.40**	**100094.91**

Tabla 46: Tiempo de ejecución del algoritmo en un solo procesador con 10 iteraciones globales

En la Tabla 46 se muestra el tiempo necesario para alcanzar los valores de calidad de las soluciones de la Tabla 45.

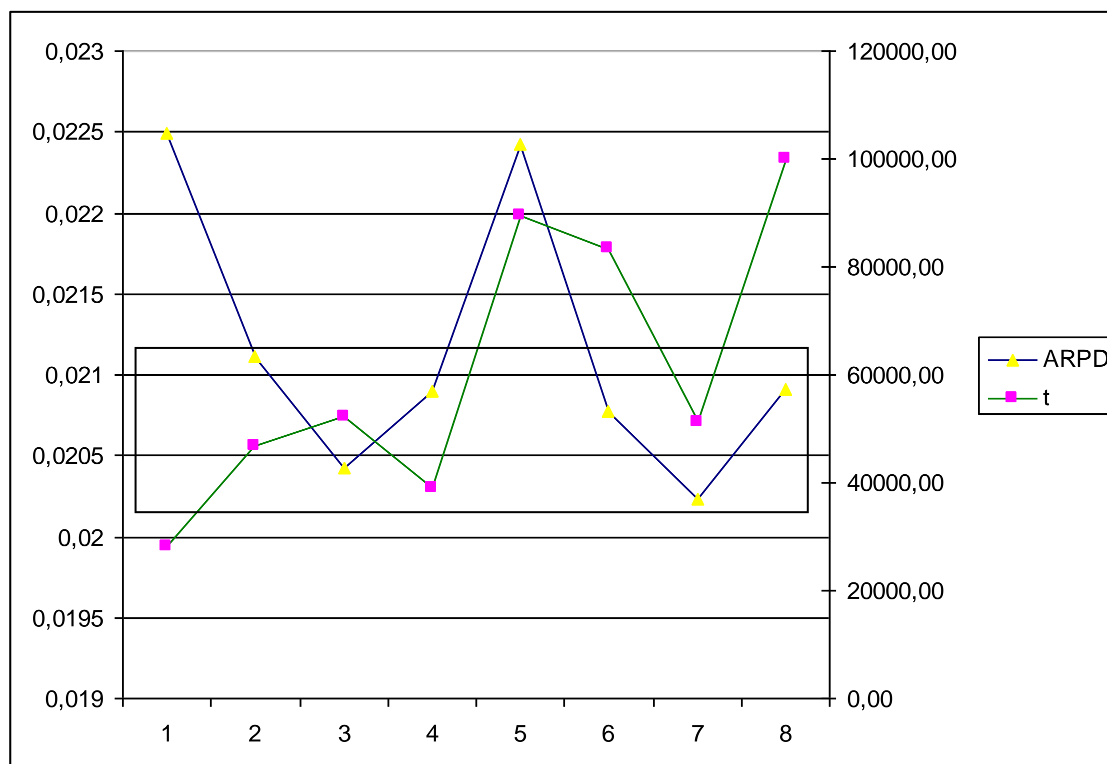


Figura 59: Gráfico de calidad y tiempo de ejecución del algoritmo secuencial

En la Figura 59 se muestran los resultados de calidad y de tiempo para los diferentes tratamientos considerados. En una franja se han resaltado aquellos tratamientos que han mostrado un mejor comportamiento respecto a los objetivos.

Mientras que los tratamientos 1, 5 y 8 producen valores extremos, los tratamientos 2, 3, 4 y 7 son los más prometedores por el compromiso alcanzado entre tiempo de ejecución y calidad de las soluciones obtenidas.

Debido a los enormes tiempos de ejecución, en un solo procesador, para los problemas de 500 trabajos, del orden de 2 semanas, se ha optado por dejarlos aparte en esta fase de diseño de la metodología.

Resumiendo los resultados finales en cuanto a eficiencia después de cada una de las 3 réplicas de cada tratamiento, se obtiene la Tabla 47.

Eficiencia				
Tratamiento	**Rep #1**	**Rep #2**	**Rep #3**	**Promedio**
1	0.43	0.42	0.39	0.42
2	0.61	0.55	0.47	0.55
3	0.51	0.50	0.49	0.50
4	0.47	0.48	0.46	0.47
5	0.49	0.44	0.48	0.47
6	0.55	0.51	0.57	0.54
7	0.55	0.54	0.57	0.55
8	0.49	0.43	0.49	0.47

Tabla 47: Resultados de los experimentos

De manera similar a otros estudios (ver, por ejemplo, Crainic y Gendreau, 2002) se ha observado que una pequeña proporción de los resultados no son útiles en el estudio seguido. Los detalles sobre este comportamiento se encuentran en el anexo 2 – "patologías en las observaciones".

5.4.8 Análisis de los resultados de eficiencia

En la Tabla 48 se muestran los resultados para el análisis ANOVA de una vía. En dicho análisis se puede observar que existe una diferencia significativa entre la varianza entre tratamientos y la varianza en los tratamientos (error), por lo que las variaciones entre las respuestas obtenidas para cada tratamiento parecen ser significativamente diferentes, ya que no están confundidas con el término de error. Como se puede observar, el ratio F es mayor que la unidad. Sin embargo este análisis no es suficiente para determinar las posibles fuentes de variación.

Source	**SS**	**DF**	**MS**	**F-ratio**	**I-hat**	**P-value**
Between	0.04958	7	0.007083	7.011	*	<0.001
Within (error)	0.01616	16	0.00101			
TOTAL	0.06575	23				

Tabla 48: ANOVA de 1 vía para promedios de la eficiencia, nivel Alfa: 0.1

Con objeto de determinar las fuentes de variación, hemos realizado, un análisis ANOVA de un grado de libertad cuyos resultados se muestran en la Tabla 49:

Factor	SS	DF	MS	Ratio F	I-hat	Valor P	% contrib
K	0.03808	1	0.03808	37.69	0.07967	<0.001	57.9
α	0.004396	1	0.004396	4.351	0.02707	0.02707	6.7
$pool$	0.003846	1	0.003846	3.806	0.02532	0.069	5.8
α x $pool$	0.003114	1	0.003114	3.083	0.02278	0.098	4.7
it_s	0.000105	1	0.000105	0.1039	0.004183	0.751	0.2
β	0.000017	1	0.000017	0.01683	-0.001683	0.898	0.0
B	2.243E-05	1	2.243E-05	0.0222	0.001933	0.883	0.0
Within (error)	0.01616	16	0.00101				
TOTAL	0.06575	23					

Tabla 49: ANOVA de un grado de libertad para eficiencia

De este análisis se deriva que las fuentes de variación o efectos significativos para la eficiencia del algoritmo paralelo son α, K, $pool$ y la interacción α x $pool$, mientras que los parámetros no significativos son β, it_s y B. Por otra parte es interesante observar que la mayor contribución porcentual a la respuesta corresponde al parámetro K, con lo que representa el factor que tiene mayor influencia ejerce sobre la eficiencia. Hay que tener en cuenta que hemos partido de un diseño saturado en el que no se tienen grados de libertad suficientes para estimar todas las interacciones. Sin embargo, al ser los parámetros β, it_s y B no significativos, cabe esperar que las interacciones combinadas con algunos de estos parámetros sean a su vez no significativas.

Por otra parte hay que puntualizar que las hipótesis de partida del ANOVA se cumplen en este caso, cumpliéndose las hipótesis de normalidad y homocedasticidad e independencia de las observaciones. Dichos resultados han sido omitidos al carecer de mayor interés.

Como era de esperar (intuitivamente) aquellos factores que resultaron significativos para las respuestas de tiempo y calidad (Fase 1) han resultado significativas para

los experimentos de eficiencia. Sin embargo hay que tener en cuenta que mientras que los parámetros más importantes para variar la calidad de las soluciones obtenidas eran el número de procesadores y con menos importancia el número de iteraciones global y el número de iteraciones dentro de los procesos trabajadores, en cambio, sobre el tiempo de ejecución influyeron todos los parámetros seleccionados, número de iteraciones globales y locales, K, número de procesadores, B y el tamaño relativo del *pool* central, además de los parámetros α y β del umbral de aceptación de soluciones.

5.4.9 Optimización

Teniendo en cuenta que el diseño empleado no es un diseño factorial completo, la elección del nivel adecuado para cada parámetro puede no corresponderse con ninguno de los tratamientos que hemos realizado. De una forma práctica y sencilla, para obtener dicha información vamos a basarnos en las contribuciones individuales de cada factor a la respuesta del sistema. Dicha contribución individual se muestra claramente en los gráficos de los efectos principales del análisis de las medias (ANOM).

En las Figuras 69-71, se muestra los gráficos de efectos principales para los parámetros: K, α y $pool$.

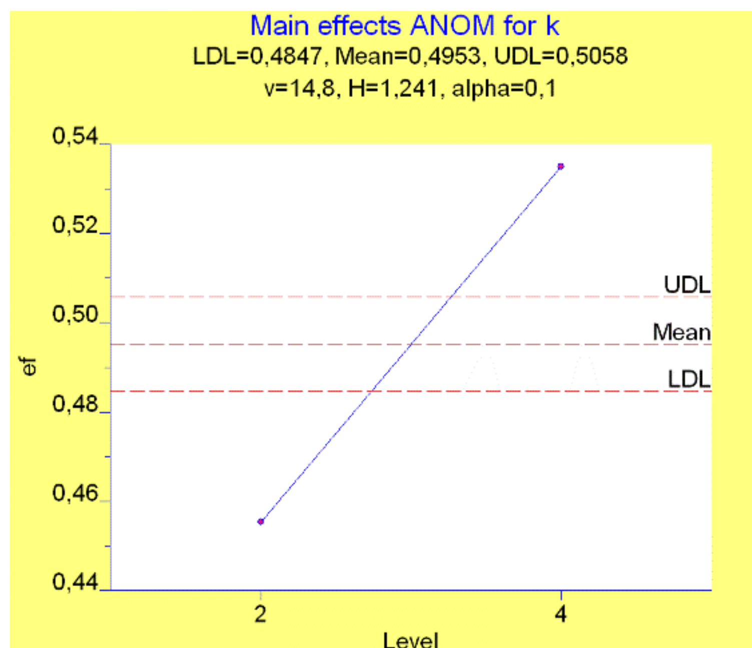


Figura 60: Gráfico de efectos principales para K

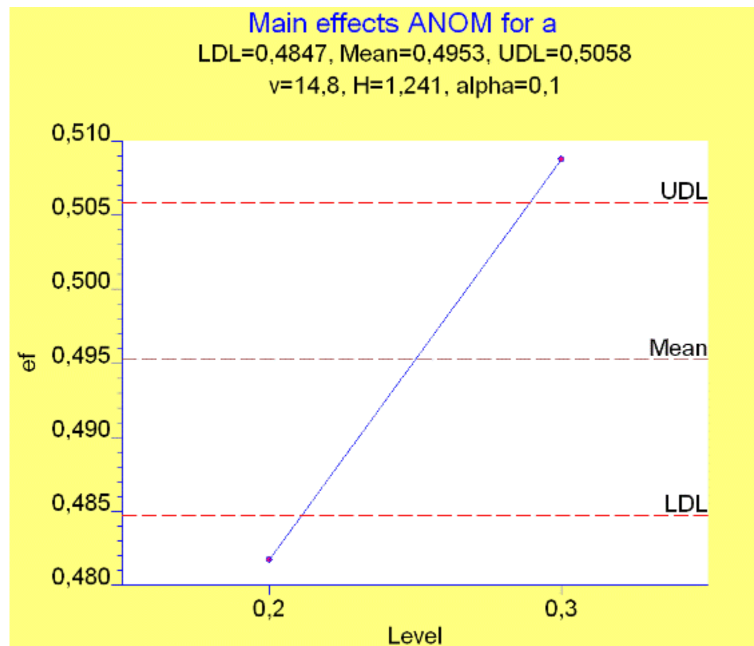


Figura 61: Gráfico de efectos principales sobre la eficiencia para α

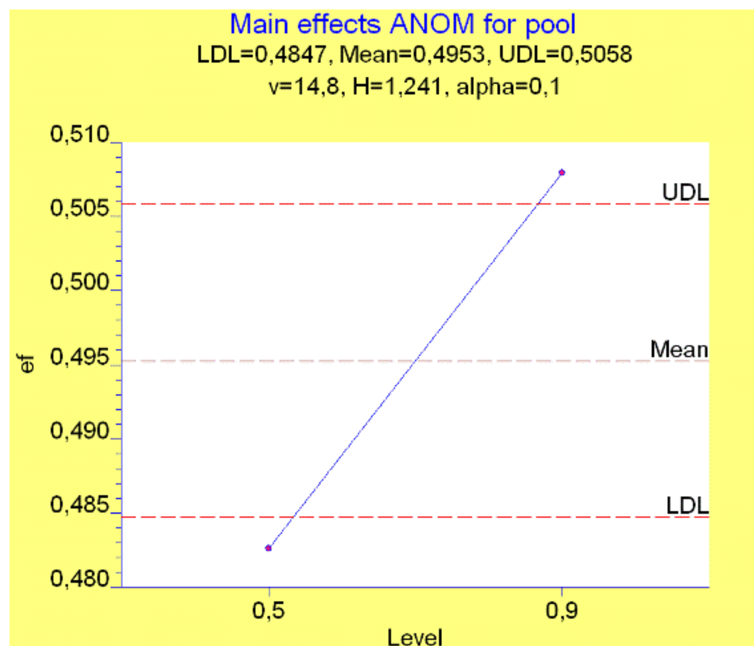


Figura 62: Gráfico de efectos principales sobre la eficiencia para $pool$

De manera coincidente con el análisis ANOVA de una vía anterior, los parámetros que resultaron significativos aparecen fuera de los límites de control en este análisis. Teniendo en cuenta que la interacción α x $pool$ tiene el mismo signo que los parámetros individuales y suponiendo que no existen más interacciones entre parámetros, se pueden estimar los valores óptimos de dichos parámetros para obtener una máxima

eficiencia. Los niveles más favorables son los niveles superiores de los anteriores parámetros.

Para el resto de parámetros los resultados son los siguientes:

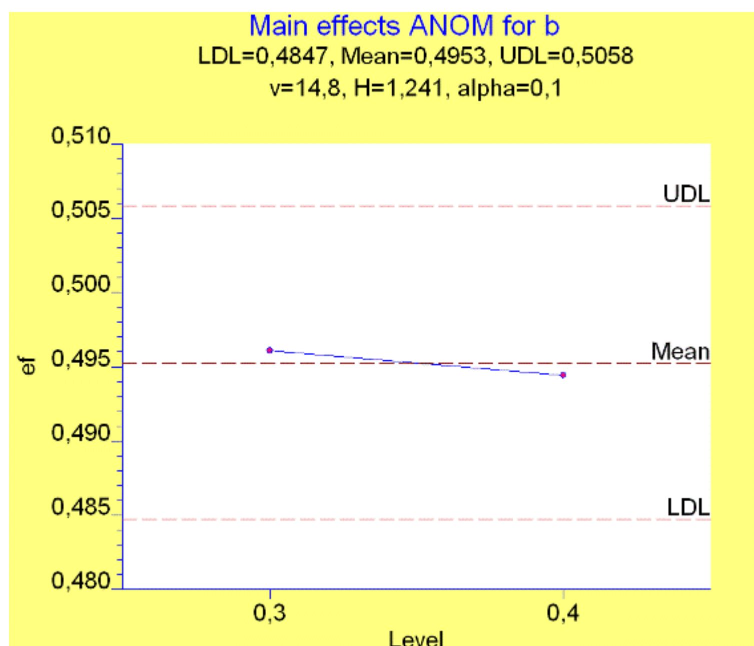


Figura 63: Gráfico de efectos principales sobre la eficiencia para β

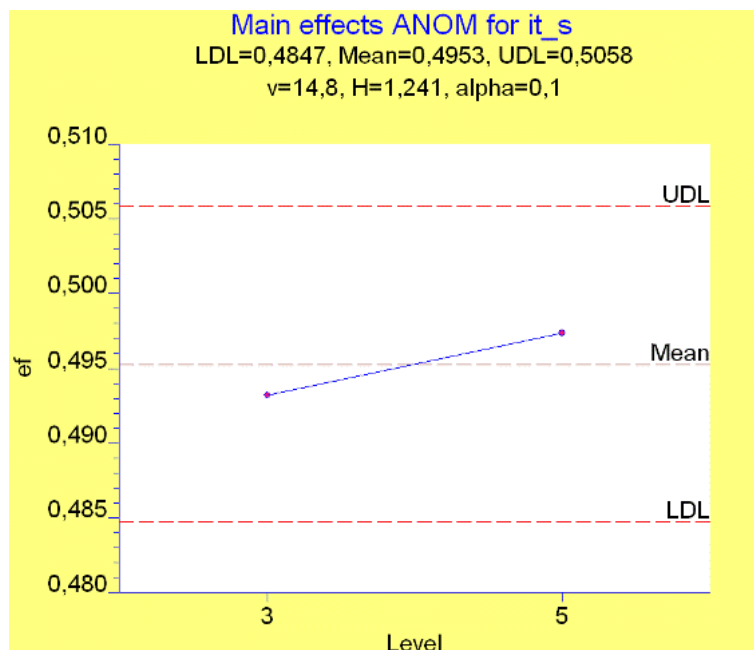


Figura 64: Gráfico de efectos principales sobre la eficiencia para el número de iteraciones en los trabajadores (it_s)

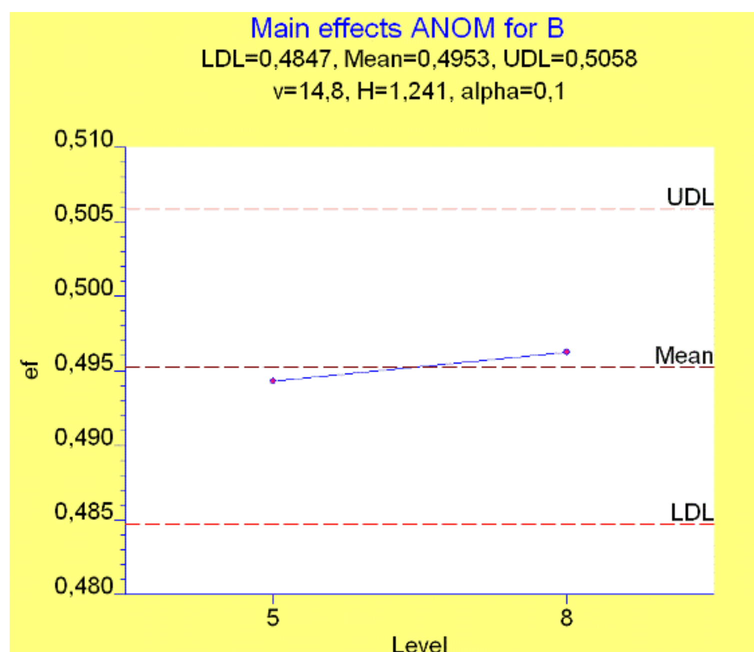


Figura 65: Gráfico de efectos principales para B

Como era de esperar, el análisis es muy parecido al ANOVA, resultando estar dentro de los límites de control los parámetros no significativos derivados del análisis ANOVA de una vía.

Teniendo en cuenta estos resultados, en la Tabla 50 se muestran aquellos valores que maximizan la eficiencia.

	α	β	it_s	K	B	$Pool$
Niveles	0.2	0.3	3	3	5	0.5
	0.3	0.4	5	4	8	0.9
Nivel óptimo	0.3	0.3	5	4	8	0.9
Significativo	SI	NO	NO	SI	NO	SI

Tabla 50: Resumen de ANOVA para ARPD

El análisis de los mejores valores de los parámetros planteados concuerda en cierto modo con lo esperado. En algunos casos, coinciden con valores previstos en la literatura (Ghosh y Sierksma, 2002, para α, β, it_s y K) y en otros coincide con las observaciones de diversos autores (ver Niar y Freville, 1996) sobre la conveniencia del

intercambio de soluciones entre procesos para mejorar el comportamiento del algoritmo paralelo de exploración.

El parámetro β no es significativo y el parámetro α tiene menor contribución a la respuesta que otros parámetros, pero existe una interacción entre el valor del parámetro α y el tamaño relativo del *pool* central de soluciones. La experimentación ha servido para encontrar los valores que ofrecen un mejor comportamiento del algoritmo en cuanto a la eficiencia.

En cuanto al número de iteraciones que realizan los procesos trabajadores antes de intercambiar soluciones de nuevo con el proceso jefe, se ha mostrado que tiene una importancia menor, aunque es lógico pensar en obtener una mejor eficiencia si este número aumenta. Esto implica más exploración antes de cada intercambio con el proceso jefe y, por tanto, menores tiempos de espera y mayor aprovechamiento de la capacidad de cómputo de cada procesador.

El número de soluciones K, que toma cada proceso para explorar su vecindad, o intensidad de la búsqueda es también lógico que haya alcanzado el valor más alto, por las mismas razones que el caso anterior. Además, este cambio entre los valores de K sí tiene una influencia significativa en el valor final de la eficiencia del algoritmo.

El factor B no tiene una influencia significativa en la eficiencia, aunque sería esperable que las variaciones en el conocimiento que unos procesos adquieren sobre la exploración realizada por otros ejercieran algún tipo de influencia. Sin embargo este comportamiento puede justificarse si tenemos en cuenta que este conocimiento implica muchos caminos de exploración que el resto de procesos encuentran cerrados, y quizá suponga un obstáculo para la mejor exploración del espacio de soluciones. Quizá, en posteriores avances de nuestra investigación, podríamos relajar esta condición, de manera que pudieran cruzarse unas sendas con otras y así poder continuar con la exploración. La influencia sobre el tiempo de comunicación del número de soluciones intercambiadas es despreciable con las velocidades de comunicación de la red *gigabit* que se ha empleado para la experimentación. De todos modos, se han recogido los datos de tiempo de comunicación y de computación, con lo que podrá estudiarse más adelante también la influencia de todos los parámetros

sobre la granularidad del algoritmo o balance entre comunicación y computación sin necesidad de experimentos adicionales.

El tamaño relativo de la memoria central o *pool*, también tiene una importancia significativa sobre la eficiencia del algoritmo, siendo el valor de esta mejor cuanto mayor sea el tamaño de la memoria central. Este resultado es menos evidente al pensar en la eficiencia del algoritmo. Por un lado, la calidad de las soluciones exploradas debería mejorar, por la memoria que tiene el algoritmo de las zonas prometedoras exploradas con anterioridad. Por otro lado, esto puede conducir al estancamiento en mínimos locales a pesar de las precauciones tenidas en el diseño del algoritmo en cuanto a selección de las soluciones de esta memoria central. Por último, la gestión de un tamaño de memoria central demasiado grande, también podría conducir a incrementar la fracción serie del algoritmo y por tanto la eficiencia del mismo se degradaría según la ley de Amdahl (1967).

5.5 Estudio estadístico sobre la eficiencia del algoritmo CLMpf

En el ajuste de parámetros de metaheurísticas, se ha buscado siempre la combinación de los parámetros tanto de búsqueda como de funcionamiento en paralelo que dan lugar a un algoritmo lo más robusto posible ante diferentes instancias del problema. En nuestro caso, disponemos de miles de observaciones que podemos emplear sin demasiada dificultad para comprobar la influencia que tienen diferentes factores en la eficiencia del algoritmo, de forma que podamos extraer algún tipo de conclusión.

En el caso de la experimentación para el problema de secuenciación de trabajos en flujo uniforme, hemos estudiado la influencia que ejercen sobre la eficiencia del algoritmo el número de trabajos, el número de máquinas, el número de procesadores y el conjunto de parámetros seleccionado para resolver cada problema.

El diseño estudia la variable dependiente Eficiencia para los casos obtenidos mediante el cruce de los factores y niveles presentados en la Tabla 51. El diseño no es factorial completo (ya que la batería de Taillard no proporciona todos los cruces, y además sólo se han considerado las instancias a partir de 20 trabajos con 20

máquinas y hasta 200 trabajos), por lo que dicha tabla muestra las observaciones realizadas para cada nivel de cada factor.

Factor	Nivel	Observaciones
Trabajos	20	960
	50	2880
	100	2880
	200	1920
Máquinas	5	1920
	10	2880
	20	3840
Número de procesadores	3	2160
	6	2160
	9	2160
	12	2160
Conjunto de parámetros	1	1080
	2	1080
	3	1080
	4	1080
	5	1080
	6	1080
	7	1080
	8	1080

Tabla 51: Factores y niveles

5.5.1 Tablas de resultados medios de eficiencia

Las siguientes tablas muestran los resultados medios de eficiencia para cada nivel de cada uno de los factores analizados.

Así, se pretende analizar si nuestra experimentación confirma los resultados presentes en la literatura en cuanto a la degradación de la eficiencia del algoritmo paralelo en función del número de procesadores. Además se pretende estudiar la in-

fluencia del tamaño del problema en la eficiencia, teniendo en cuenta los resultados medios obtenidos para los distintos valores de trabajos y máquinas.

			Eficiencia	
n	m	np	CLMpf	CLMpg
20	20	3	1.05	
		6	0.47	0.36
		9	0.28	
		12	0.20	0.41
50	5	3	0.79	
		6	0.30	0.24
		9	0.15	
		12	0.11	0.12
	10	3	0.85	
		6	0.59	0.65
		9	0.34	
		12	0.25	0.26
	20	3	1.04	
		6	0.74	0.59
		9	0.49	
		12	0.37	0.76
100	5	3	0.40	
		6	0.17	0.30
		9	0.11	
		12	0.10	0.29
	10	3	0.76	
		6	0.42	0.71
		9	0.28	
		12	0.21	0.45
	20	3	0.76	
		6	0.56	1.02
		9	0.38	
		12	0.30	0.59
200	10	3	0.48	
		6	0.31	0.76
		9	0.20	
		12	0.16	0.46
	20	3	0.63	
		6	0.49	1.04
		9	0.38	
		12	0.28	0.63
			0.43	0.54

Tabla 52: Media de la eficiencia según trabajos (n), máquinas (m) y procesadores (np)

En la Tabla 52 se observa que los valores más altos de eficiencia se presentan cuando se emplean menos procesadores, 3 en nuestro caso, llegando en algunos de

ellos a aceleraciones superlineales, como por ejemplo en las series de 20x20 y 50x20. La eficiencia decae como era de esperar conforme aumenta el número de procesadores, hasta 12 en nuestro caso. El mejor valor se obtiene para la serie de 50x20 mientras que el peor se presenta con 50 trabajos y 5 máquinas, con un valor bajo que indica que los parámetros de exploración están más adaptados en cuanto a eficiencia a problemas de mayor complejidad.

Eficiencia		**Número de procesadores**				
n	m	**3**	**6**	**9**	**12**	**Promedio**
20	20	1.05	0.47	0.28	0.20	**0.50**
50	5	0.79	0.30	0.15	0.11	**0.34**
	10	0.85	0.59	0.34	0.25	**0.51**
	20	1.04	0.74	0.49	0.37	**0.66**
100	5	0.40	0.17	0.11	0.10	**0.19**
	10	0.76	0.42	0.28	0.21	**0.42**
	20	0.76	0.56	0.38	0.30	**0.50**
200	10	0.48	0.31	0.20	0.16	**0.29**
	20	0.63	0.49	0.38	0.28	**0.44**
Promedio		**0.75**	**0.45**	**0.29**	**0.22**	**0.43**

Tabla 53: Resultados medios de eficiencia agrupados según n y m

El algoritmo se ha adaptado muy bien en promedio la serie de problemas de 50 trabajos y 20 máquinas, mientras que los peores resultados en cuanto a eficiencia del algoritmo CLMpf se dan con la serie de 200 trabajos y 10 máquinas.

En la tabla Tabla 54 se observa cómo el tratamiento número 7 es el que ofrece mejor valor de la eficiencia en promedio para la batería de problemas estudiada.

Eficiencia	Número de procesadores				
tratamiento	3	6	9	12	promedio
1	0.67	0.37	0.26	0.18	0.37
2	0.85	0.46	0.30	0.22	0.46
3	0.73	0.50	0.33	0.25	0.45
4	0.75	0.40	0.27	0.21	0.41
5	0.71	0.45	0.29	0.20	0.41
6	0.82	0.48	0.28	0.21	0.45
7	0.84	0.50	0.32	0.26	0.48
8	0.66	0.43	0.28	0.21	0.40
promedio	0.75	0.45	0.29	0.22	0.43

Tabla 54: Promedios de eficiencia agrupados según cada uno de los tratamientos

5.5.2 Gráficas de las medias de la eficiencia

En las gráficas siguientes se representan los valores medios de la eficiencia (en tanto por 1) según los factores número de trabajos, número de máquinas y procesadores.

La dispersión en los valores observados de la eficiencia va disminuyendo conforme aumenta el número de procesadores empleado. Se observa, además un gran declive de los valores medios, debida a los tiempos de espera impuestos por la sincronización entre procesos. Esta observación se hace más patente en la Figura 67.

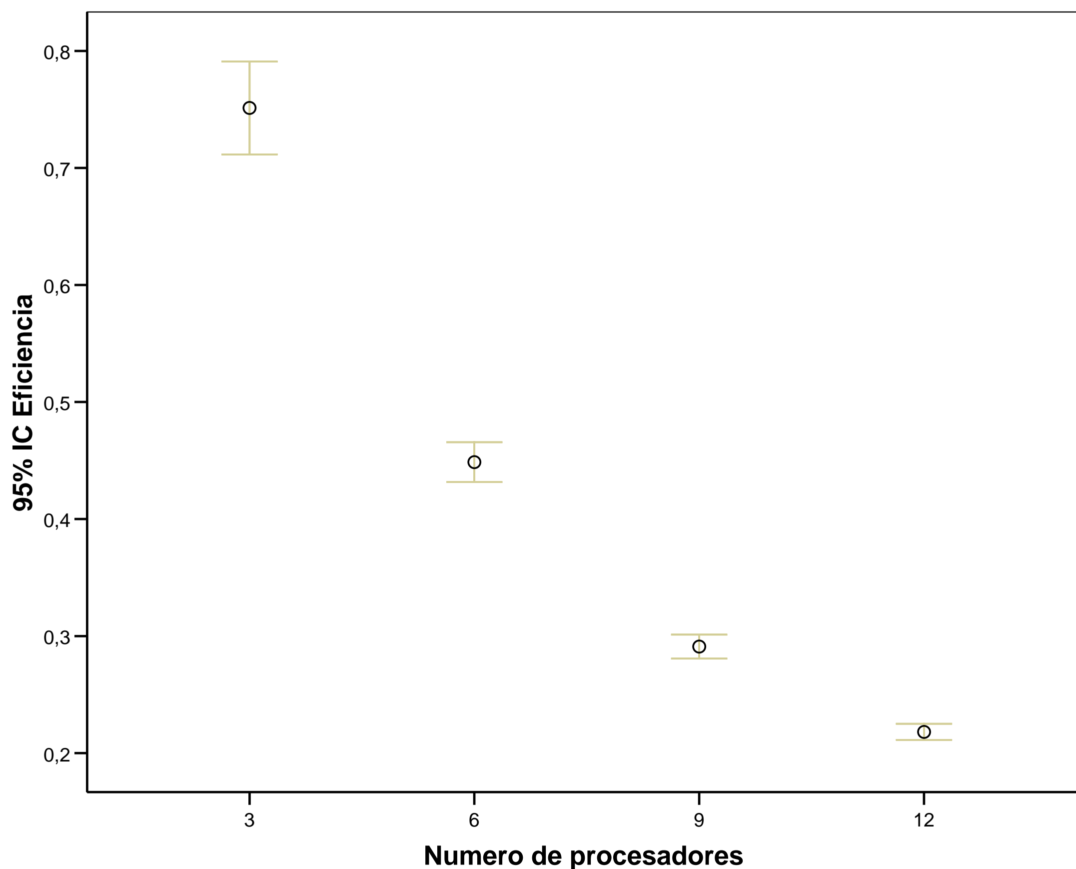


Figura 66: Diagrama de cajas para la eficiencia según el número de procesadores (nivel de confianza del 95%)

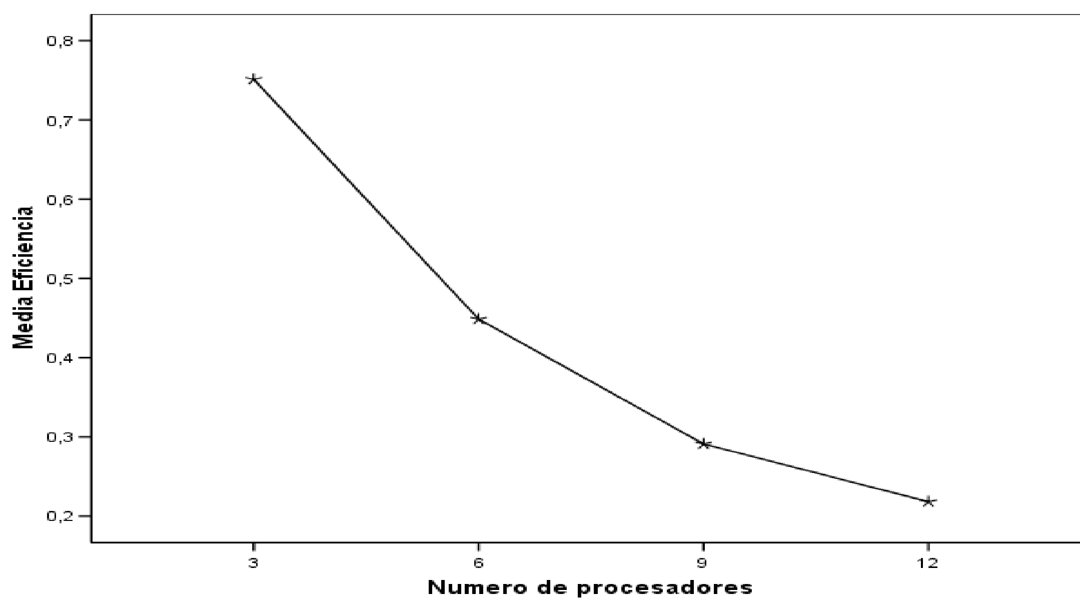


Figura 67: Valor medio de eficiencia según el número de procesadores

La Figura 68 muestra los resultados marginales de eficiencia con respecto al número de trabajos de la instancia del problema para cada nivel del número de procesadores.

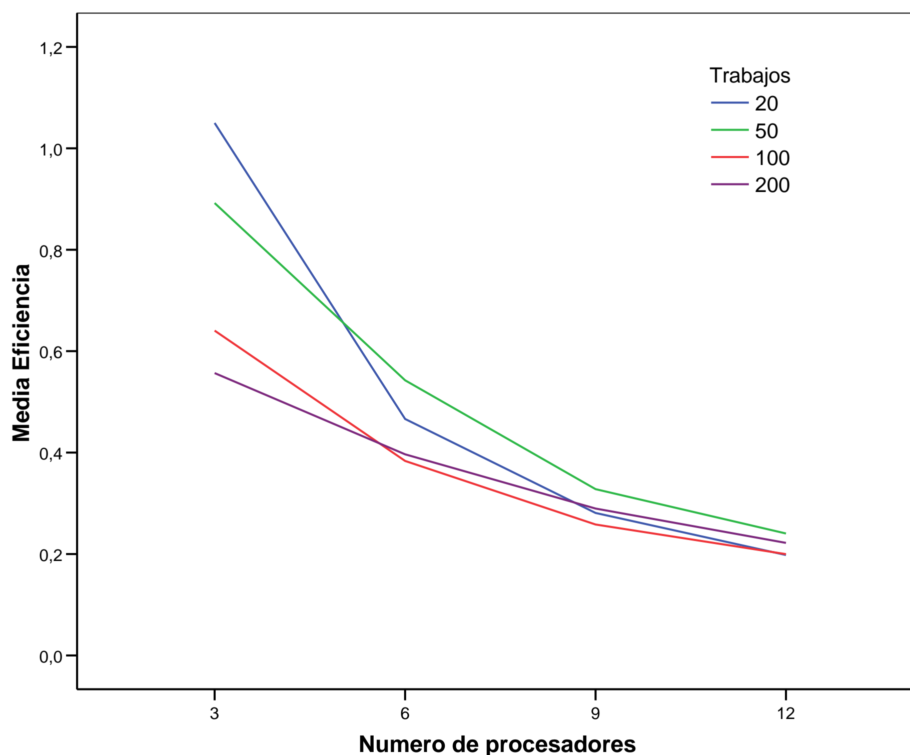


Figura 68: Valores medios de eficiencia agrupados por número de trabajos para cada número de procesadores

Se puede observar la irregularidad que muestran los resultados para los distintos niveles del número de trabajos. En general, los resultados parecen mejores para los casos de 20 y 50 trabajos. Además, los valores de eficiencia se aproximan entre sí conforme aumenta el número de procesadores, por lo que se intuye que, para un número grande de procesadores el número de trabajos no va a ser un factor influyente sobre la eficiencia.

La Figura 69 representa, de la misma forma, los resultados marginales de eficiencia para cada nivel del número de procesadores, pero en este caso con respecto al número de máquinas de la instancia del problema.

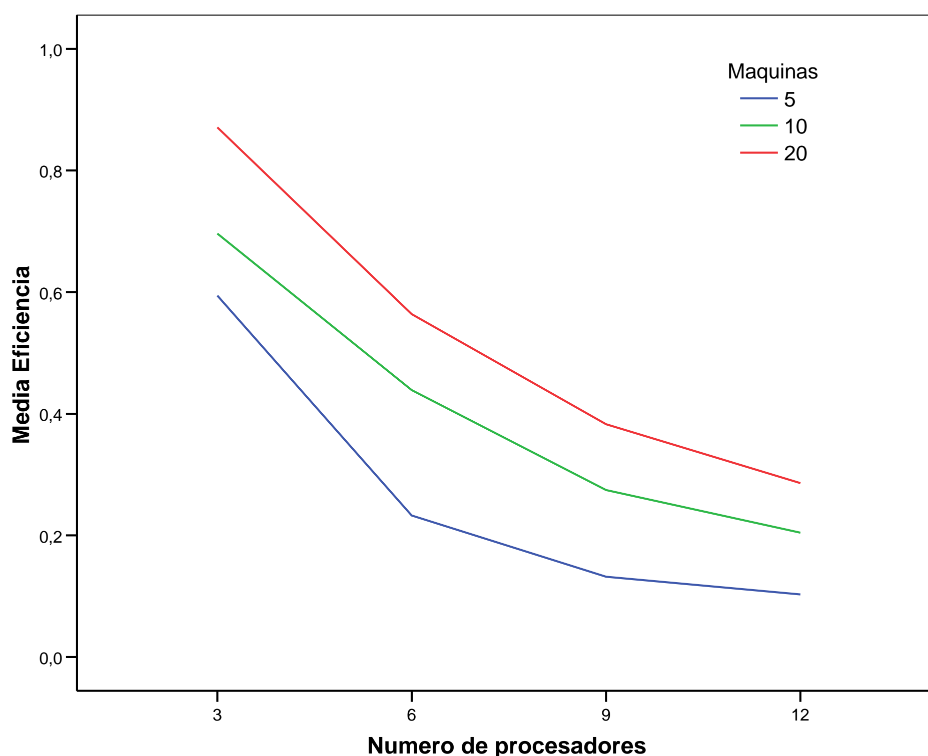


Figura 69: Valores medios de eficiencia agrupados por número de máquinas para cada número de procesadores

Aquí se observa un comportamiento similar para cada nivel del número de máquinas, siendo mayor la eficiencia cuanto menor sea el número de procesadores, e ir aumentando ésta a medida que aumenta el número de máquinas.

5.5.3 Análisis estadístico

Para validar las observaciones llevadas a cabo mediante el estudio anterior con las tablas y gráficas de los valores medios de eficiencia para los niveles de los factores, se va a llevar a cabo un estudio estadístico que afirme o rechace las hipótesis sobre la influencia de los factores del diseño sobre la variable Eficiencia, y la diferencia o no de los resultados de dicha variable para los distintos niveles de cada factor.

Así, con los resultados de los experimentos realizados, la primera intención es realizar un estudio de la varianza o ANOVA, pero al no verificarse todas las hipótesis necesarias para poder aplicar este test (independencia, normalidad y homocedasti-

cidad) se ha optado por un estudio no paramétrico para comprobar la influencia de los factores sobre la eficiencia del algoritmo paralelo. Los estudios se han realizado con el paquete estadístico SPSS. El nivel de significación usado en todos los test realizados es el valor usual de 0.05.

Las pruebas no paramétricas realizadas son la prueba H de Kruskal-Wallis, que contrasta la igualdad entre poblaciones, de forma que nos permite averiguar para qué factores la eficiencia es diferente (Pardo Merino y Ruiz Díaz, 2005).

En caso de que, para un factor en particular, la prueba H dé como resultado que no existen diferencias entre sus niveles, se realizará un estudio más detallado de las diferencias existentes entre las medias de la eficiencia para cada una de los niveles, realizando un análisis de comparaciones por pares. Para ello, se emplea la prueba U de Mann-Whitney para dos muestras junto con la corrección de Bonferroni para controlar la tasa de error. La corrección de Bonferroni consiste en emplear un nivel de significación de $0.05/nc$ siendo nc el número de comparaciones por pares que se desea realizar (Pardo Merino y Ruiz Díaz, 2005).

En los siguientes subapartados se va a llevar a cabo dicho estudio para cada factor respectivamente.

5.5.3.1 Influencia del tamaño del clúster

El objetivo de este apartado es analizar la influencia del número de procesadores en los valores de la eficiencia del algoritmo paralelo.

En primer lugar realizamos la prueba no paramétrica de Kruskal-Wallis para ver si existen diferencias entre los niveles de este factor. Así, el nivel crítico asociado al estadístico H es menor que 0.05, indicando que la eficiencia es significativamente diferente para cada tamaño de clúster. Para estudiar las diferencias entre los niveles del factor, se emplea el procedimiento de Mann-Whitney. La corrección de Bonferroni implica un nivel de significación de $0.05/6 = 0.008$ y los P-valores obtenidos para el estadístico U son todos menores que dicho nivel. Así, la eficiencia media es significativamente diferente para todos los niveles. Los resultados obtenidos representados mediante tabla de subconjuntos homogéneos es la siguiente:

Numero de procesadores	Observaciones	Subconjunto			
		1	2	3	4
12 procs	2160	0.218111			
9 procs	2160		0.290975		
6 procs	2160			0.448615	
3 procs	2160				0.751240
Sig.		1.000	1.000	1.000	1.000

Tabla 55: Subgrupos según el factor número de procesadores

En cada subgrupo se observa cómo la media de la eficiencia del algoritmo paralelo va disminuyendo conforme aumenta el número de procesadores, coincidiendo con la ley de Amdahl (1967), y con los resultados previos presentados en las tablas y las gráficas del apartado anterior.

5.5.3.2 Influencia del número de trabajos

Se pretende estudiar la posibilidad de que no existan diferencias significativas entre el valor de la eficiencia obtenida con diferentes tamaños de problema (en este caso según el número de trabajos) para llegar a alguna conclusión sobre la robustez del algoritmo paralelo propuesto ante diferentes tamaños de problema.

Así, en el caso del número de trabajos, la prueba no paramétrica proporciona el nivel crítico asociado al estadístico H: P-valor = 0.00 < 0.05, indicando que la eficiencia es significativamente diferente para cada valor del número de trabajos (20, 50, 100 y 200).

Ante este resultado, es necesario llevar a cabo la prueba U (ver Tabla 56). Por la corrección de Bonferroni, el nivel de significación es de 0.05/6 = 0.008. Los P-valores obtenidos han sido menores que la significación fijada en todos los casos, excepto para la comparación entre los niveles 20 y 50, donde el P-valor ha sido 0.027. Esto implica que no existen indicios para rechazar la hipótesis de igualdad entre estos niveles del factor Trabajos. Así, la eficiencia media es significativamente diferente para todos los niveles excepto para 20 y 50. La siguiente tabla muestra

estos resultados en forma de subconjuntos, indicando la media de la eficiencia para cada subconjunto, obteniéndose el mejor resultado de eficiencia para los problemas con 20 y 50 trabajos, como se intuía en los resultados previos de las tablas y se observó en la Figura 77.

Trabajos	Observaciones	Subconjunto		
		1	2	3
200	1920	0.366256		
100	2880		0.370520	
20	960			0.498843
50	2880			0.500735

Tabla 56: Subgrupos según el factor número de trabajos

5.5.4 Influencia del número de máquinas

Se ha realizado el mismo estudio para el número de máquinas. El estudio no paramétrico del estadístico H da como resultado un P-valor menor que 0.05, por lo que existen diferencias significativas entre los distintos valores del número de máquinas. Las comparaciones por pares de Mann-Whitney dan como resultados P-valores todos menores que la significación con la corrección de Bonferroni igual a 0.05/3=0.01, por lo que los subconjuntos resultantes son los siguientes.

Máquinas	Observaciones	Subconjunto		
		1	2	3
5	1920	0.265470		
10	2880		0.403560	
20	3840			0.525875

Tabla 57: Subgrupos según el factor número de máquinas

Por lo tanto, se puede afirmar que la eficiencia depende del número de máquinas, aumentando a medida que aumenta el número de éstas. Esto parece deberse a que

las soluciones encontradas con el algoritmo secuencial son de calidad menor en el caso de los problemas más complejos. En este caso, el algoritmo funcionando en paralelo obtiene una mejora más significativa en el tiempo de ejecución que cuando se trata de problemas más sencillos.

5.5.5 Selección del mejor conjunto de parámetros

Para decidir cuál es el conjunto de parámetros con el que se obtiene un mejor resultado para la eficiencia, repetimos el análisis anterior agrupando los resultados según cada uno de los ocho conjuntos de parámetros que se han propuesto.

La prueba de Kruskal-Wallis proporciona un P-valor menor que 0.05, por lo que se han realizado, como anteriormente, las comparaciones por pares de Mann-Whitney. Los resultados no muestran diferencias significativas entre los valores de eficiencia del algoritmo paralelo obtenidas para distintos casos con nivel de significación 0.05/28 = 0.002, distinguiéndose dos posibles subconjuntos, mostrados en la tabla siguiente:

Combinación de parámetros	N	Subconjunto	
		1	2
1	1080	0.367800	
8	1080	0.397076	0.397076
4	1080	0.407103	0.407103
5	1080	0.412095	0.412095
6	1080	0.446664	0.446664
3	1080		0.451542
2	1080		0.456684
7	1080		0.478919
Sig.		0.483	0.126

Tabla 58: Subconjuntos homogéneos según el factor combinación de parámetros

Aunque no existen diferencias estadísticamente significativas entre los casos presentados en el subconjunto 2, el mejor valor es el obtenido con el conjunto número 7, cuyos valores son los siguientes:

Tratamiento	α	β	It_s	K	B	$pool$
7	0.3	0.3	5	4	5	0.9

Tabla 59: Conjunto de valores que ofrece mejor resultado en cuanto a eficiencia

Estos resultados coinciden con los obtenidos cuando se realizó el estudio mediante promedio de observaciones por tamaño de problema, salvo el parámetro B, cuyo nivel óptimo fue en aquella ocasión de 8. Esta es una diferencia menor, ya que también se observó que las variaciones de dicho parámetro no eran significativas para el valor de la eficiencia del algoritmo entre los dos niveles propuestos, 5 y 8.

5.6 Estudio de la robustez del algoritmo

5.6.1 Introducción

Una de las medidas de calidad de los algoritmos, tal como señalan Barr *et al.* (1995), es la robustez. En esta sección se pretende estudiar la robustez del algoritmo CLMpf desde dos puntos de vista diferentes que creemos son de interés en la aplicación práctica:

1.- Robustez respecto el tamaño del problema, para un número fijo de procesadores.

2.- Robustez respecto el número de procesadores, para un tamaño de problema determinado.

El primer caso puede ser de interés para explotar las características de un determinado clúster para un conjunto de instancias de diferentes tipos. En el segundo caso se trata de estudiar la robustez del algoritmo para un determinado problema, cuando éste va a ser ejecutado en un clúster del que en principio desconocemos las características respecto al número de procesadores a emplear.

Para realizar este estudio nos basamos en la metodología de Taguchi (e.g. Taguchi *et al.*, 1989). Taguchi indica que un determinado sistema está influido por variables que podemos tener bajo control – *parámetros de control* – y otras variables que aunque pueden ser conocidas y medidas, escapan de nuestro control. A estos últimos parámetros les denomina *factores de ruido*. La idea de Taguchi era, basándose en técnicas derivadas del DOE clásico, obtener la combinación de parámetros de control más adecuada ante una posible variación de las condiciones en el entorno, obteniendo así un sistema más robusto.

En los estudios de robustez basados en la metodología de Taguchi se suele emplear la conocida expresión de la relación señal/ruido (*Signal to Noise Ratio*), que, para el caso de maximización de una determinada respuesta (en nuestro caso el objetivo es maximizar la eficiencia) es:

$$SN_i = -10log\frac{1}{n}\sum_{j=1}^{n}\frac{1}{y_{ij}^2} \quad (22)$$

Donde SN_i es la relación señal/ruido medida en decibelios para un determinado tratamiento, y_{ij} es la respuesta del sistema (eficiencia) para el tratamiento i y para un determinado valor j de los factores de ruido (con n posibles combinaciones de factores de ruido). Para más información acerca de este ratio o para otros objetivos diferentes al de maximización, ver Taguchi *et al.* (1989).

Esta relación señal/ruido nos da un valor de ganancia en decibelios. De esta manera, la combinación de factores que consiga una mayor ganancia se considerará la más robusta bajo dicha combinación de factores.

Para conducir la experimentación con esta metodología se suelen emplear dos matrices: una para indicar los tratamientos (matriz interna) y otra para designar las combinaciones de los parámetros de ruido (matriz externa). Estas matrices se cruzan para formar la matriz experimental que se muestra en la Tabla 60.

	Tratamiento	1	2	3	4	5	6	7	8
Parámetros de Control	α	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
	β	0.3	0.3	0.4	0.4	0.4	0.4	0.3	0.3
	It_s	3	3	5	5	3	3	5	5
	K	2	4	4	2	2	4	4	2
	B	5	8	8	5	8	5	5	8
	Pool	0.5	0.9	0.5	0.9	0.9	0.5	0.9	0.5
Factores de Ruido	*Factor 1*								
	Factor 1								
	…								
	Factor n								

Tabla 60: Matriz experimental de Taguchi

5.6.2 Enfoque 1: Robustez para un número determinado de procesos y diferentes tamaños de problema

Se busca, con este estudio, encontrar la mejor combinación posible de parámetros cuando se dispone de un clúster para computación en paralelo con un número conocido de procesadores, para obtener un algoritmo robusto ante diferentes tamaños de problema. En este caso, los factores de ruido son los distintos tamaños de problema. Maximizar la relación señal/ruido significa que tendremos un algoritmo que minimiza las variaciones de su eficiencia cuando cambia el tamaño de problema al que se aplica.

	Tratamientos							
s/n (dB)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
3 procs	-12.93	-13.59	-14.41	-11.70	-11.59	-8.75	-14.24	-12.14
6 procs	-21.00	-15.17	-17.94	-17.36	-18.15	-13.44	-16.22	-15.36
9 procs	-18.76	-18.82	-24.10	-17.17	-21.16	-22.19	-19.43	-17.65
12 procs	-22.45	-18.90	-19.44	-17.26	-17.68	-20.84	-20.72	-21.21

Tabla 61: Relación señal/ruido para cada número de procesadores, siendo el tamaño de problema el factor de ruido

El estudio comienza separando los valores de eficiencia para cada número de procesadores. Las tablas con los datos de eficiencia y la relación señal/ruido resultante se muestran en el anexo 3.

Para cada uno de estos estudios se pueden representar gráficamente las medias marginales de la relación señal ruido, para cada uno de los parámetros, tal como se muestra en la Figura 70 y siguientes.

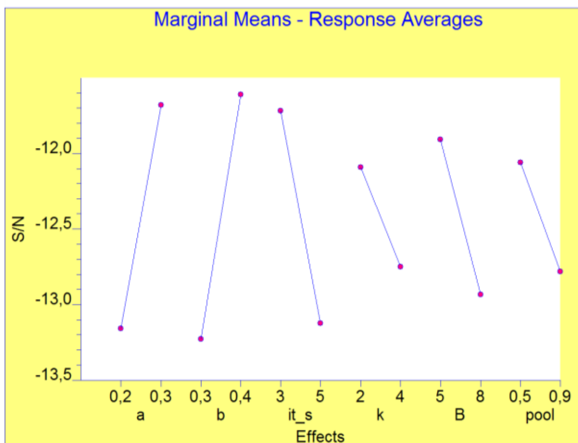


Figura 70: 3 procesadores

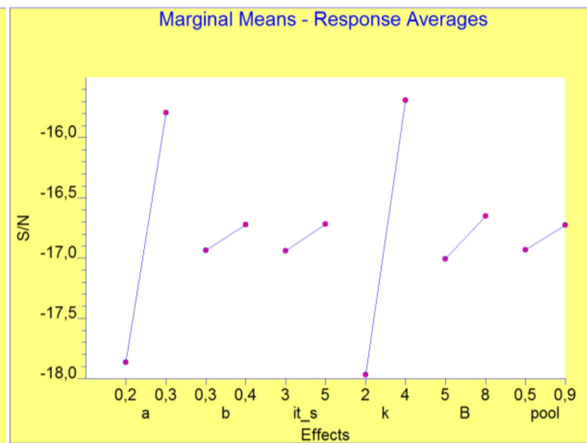


Figura 71: 6 procesadores

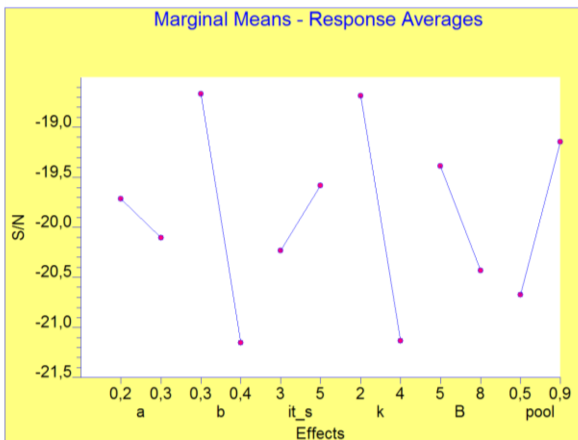


Figura 72: 9 procesadores

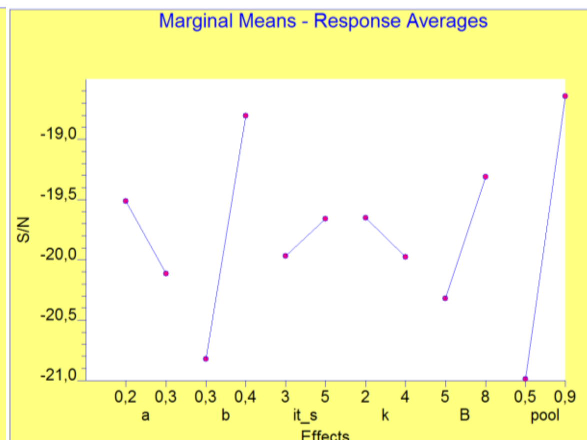


Figura 73: 12 procesadores

De donde, tomando los valores más altos de la relación señal ruido, podemos deducir que la combinación de parámetros que hace al algoritmo más robusto ante diferentes tamaños de problema en función del número de procesadores disponibles es la que se muestra en la Tabla 62:

np	α	β	*It_s*	*K*	*B*	*pool*
3	0.3	0.4	3	2	5	0.5
6	0.3	0.4	5	4	8	0.9
9	0.2	0.3	5	2	5	0.9
12	0.2	0.4	5	2	8	0.9

Tabla 62: Valores de los parámetros que maximizan la relación señal/ruido para cada número de procesadores

Ninguna de estas combinaciones de parámetros se corresponde con ninguno de los tratamientos realizados. Para validar los resultados obtenidos, vamos a realizar un experimento confirmatorio. Este ha sido realizado para el caso de 12 procesadores. Los resultados obtenidos son los siguientes:

Problema	Eficiencia
20x20	0.196
50x5	0.099
50x10	0.233
50x20	0.267
100x5	0.171
100x10	0.143
100x20	0.247
200x10	0.294
200x20	0.196
S/N	**-15.299**

Tabla 63: Relación señal/ruido según tamaño de problema para 12 procesadores

Como puede observarse, el valor de ganancia obtenido es superior a aquellos obtenidos previamente en la experimentación para 12 procesadores (ver Tabla 61), lo que ratifica la mejora de la robustez con los parámetros propuestos.

5.6.3 Enfoque 2: Robustez para un problema determinado y diferente número de procesadores

En este caso se realiza el mismo estudio, agrupando los resultados según el número de procesadores empleados para distintos tamaños de problema. Las tablas con los valores de eficiencia agrupados por tamaño de problema se muestran en el anexo 2.

S/N (dB)	1	2	3	4	5	6	7	8
20x20	-16.91	-14.16	-7.38	-11.34	-17.93	-13.44	-10.32	-16.28
50x5	-24.12	-20.71	-20.99	-15.65	-24.13	-16.78	-23.64	-24.34
50x10	-13.30	-11.19	-8.94	-12.80	-10.74	-10.93	-10.10	-9.65
50x20	-10.88	-8.82	-9.70	-10.90	-11.07	-13.89	-7.91	-9.47
100x5	-25.69	-21.04	-21.96	-22.22	-20.74	-25.71	-22.02	-16.97
100x10	-18.66	-13.12	-17.98	-18.42	-13.54	-18.76	-17.19	-15.53
100x20	-12.04	-11.23	-11.06	-10.74	-13.76	-12.04	-10.15	-11.65
200x10	-18.66	-21.17	-27.65	-17.04	-19.41	-20.88	-20.91	-19.94

Tabla 64: Relación señal/ruido para cada tamaño de problema, siendo el número de procesadores el factor de ruido

Con este enfoque, se pretende encontrar la combinación de parámetros que hace más robusto al algoritmo paralelo, en cuanto a la eficiencia cuando varía el número de procesadores. Con esto conseguimos un algoritmo paralelo cuyo comportamiento, medido por la eficiencia relativa, no se deteriora cuando varía el número de procesadores. Podremos emplearlo entonces sin importar el número de procesadores de los que disponga nuestro sistema informático.

Nuevamente, se muestra en la Figura 74 la gráfica de medias marginales, con objeto de observar los mejores valores de la relación señal/ruido:

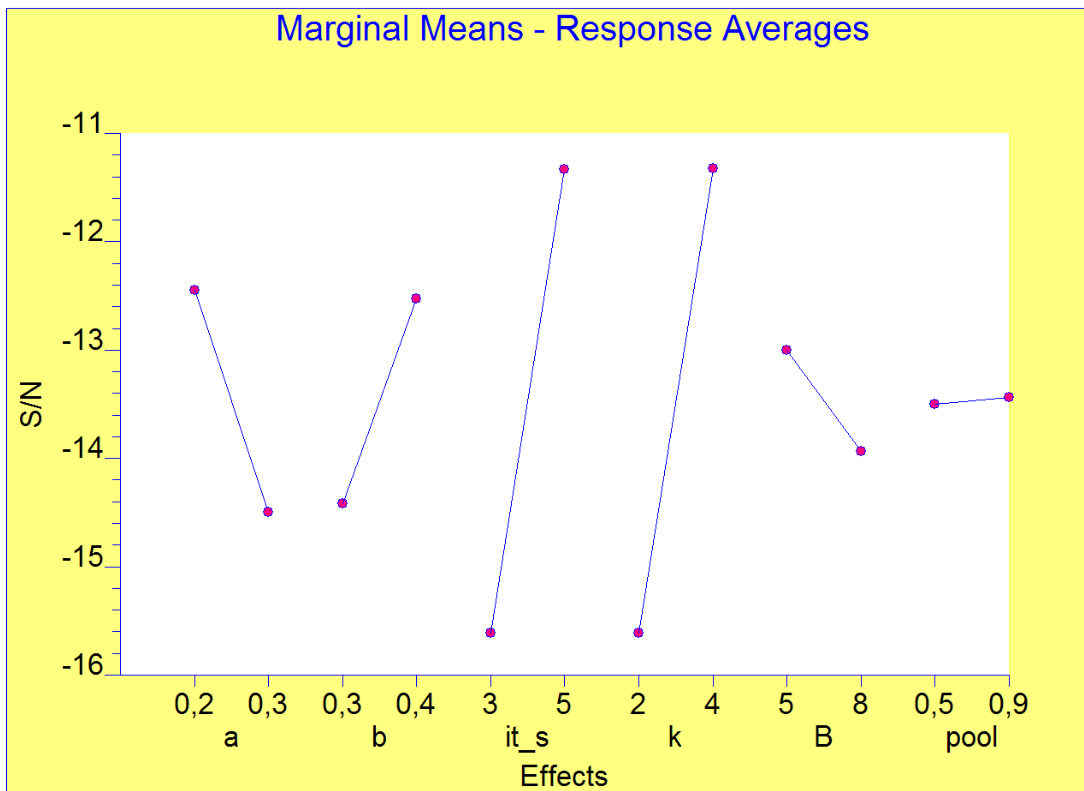


Figura 74: Relación señal/ruido para la eficiencia del algoritmo cuando se varía el número de procesadores para problemas 20x20

Tamaño de problema	α	β	*It_s*	*K*	*B*	*Pool*
20x20	0.2	0.4	5	4	5	0.9
50x5	0.2	0.4	5	4	5	0.9
50x10	0.3	0.4	5	4	8	0.5
50x20	0.2	0.3	5	4	8	0.9
100x5	0.3	0.3	5	2	8	0.9
100x10	0.3	0.3	3	2	8	0.9
100x20	0.2	0.3	5	4	5	0.9
200x10	0.3	0.3	3	2	5	0.9
200x20	0.3	0.4	5	2	8	0.9

Tabla 65: Valores de los parámetros que maximizan la relación señal/ruido para cada tamaño de problema

Como en el primer enfoque, se toman los valores de los parámetros que hacen mayor la relación señal/ruido, ya que estamos buscando la combinación de parámetros que conduce al algoritmo más robusto posible ante diferente número de procesadores. El resultado se muestra en la Tabla 65. Las figuras para el resto de tamaños de problema se muestran en el anexo 3.

Para este segundo enfoque se realiza un experimento confirmatorio con la combinación de parámetros resultante para problemas de dimensión 100x20 en un solo procesador y, una vez obtenida la solución correspondiente, se correrá el algoritmo con la misma combinación de parámetros con diferente número de procesadores, 3, 6, 9 y 12, de los que se han obtenido los resultados anteriores. Se ha tomado ese tamaño de problema por ser de tamaño medio y porque sus tiempos de experimentación no son excesivos.

La combinación de parámetros, para los problemas de 100 trabajos y 20 máquinas, que maximiza la relación señal/ruido para diferente número de procesadores, siendo el factor de ruido el número de procesadores es la siguiente.

Tamaño de problema	α	β	*It_s*	*K*	*B*	*Pool*
100x20	0.2	0.3	5	4	5	0.9

Tabla 66: Combinación de parámetros para maximizar la relación señal/ruido para la serie 100x20

Nº procesadores	Eficiencia
3	0.679
6	0.473
9	0.379
12	0.274
S/N	**-8.281**

Tabla 67: Relación señal/ruido según número de procesadores para la serie 100x20

Resultando, una relación señal/ruido más favorable que con las combinaciones de parámetros anteriores (ver Tabla 64).

5.7 Conclusiones

En este capítulo de la tesis, se aborda el problema del ajuste de parámetros en algoritmos paralelos bajo el criterio de maximización de la eficiencia. Basándonos en los buenos resultados en cuanto a calidad de soluciones obtenidos por el algoritmo CLMpf (el promedio de ARPD para la dimensión de los problemas estudiados antes del ajuste de parámetros para CLMpf es de 0.017 y para CLMpg 0.053) y teniendo en cuenta el empobrecimiento de la eficiencia, nos hemos centrado en el ajuste de los parámetros que afectan a dicho algoritmo.

Para poder ajustar los parámetros de manera adecuada, hemos hecho en primer lugar una revisión sobre el estado del arte en cuanto a diversos métodos susceptibles de ser empleados para resolver este problema. Las ventajas e inconvenientes de dichos métodos han sido puestas de manifiesto, decantándonos finalmente por un método basado en el diseño experimental (DOE).

A continuación se ha formulado una metodología para la optimización de la eficiencia, consistente en dos fases. En la primera de ellas se realiza un estudio de los parámetros sobre los criterios de tiempo y calidad de soluciones (medida mediante ARPD). En esta fase se eliminan los parámetros que no tienen influencia significativa sobre ambas respuestas, ya que se supondrá que no ejercerán una influencia significativa sobre la eficiencia. En la segunda fase se optimizan los parámetros significativos respecto al criterio de maximización de la eficiencia. En esta nueva fase nos basamos en el conocimiento previo sobre el ámbito de los parámetros y se puede afinar en cuanto al número de niveles y a los valores de dichos niveles para cada parámetro.

Los resultados reflejan que los parámetros que afectan a la eficiencia del algoritmo son el número de soluciones K que toma cada proceso para explorar su vecindad, el parámetro α y el tamaño relativo de la memoria central o *pool*. Además se han propuesto los valores más adecuados que se derivan del estudio para el objetivo de máxima eficiencia, incidiendo en su interpretación física.

Posteriormente, se ha estudiado la sensibilidad de la eficiencia del algoritmo ante diferentes tamaños de problema para comprobar la robustez del mismo. Se ha

concluido que, en el caso de nuestro problema donde la dificultad está tanto en el tamaño de las soluciones, el número de trabajos, como en el número de máquinas, la eficiencia del algoritmo paralelo se ve más afectada por el número de máquinas que por el número de trabajos y, por tanto, los niveles seleccionados para los parámetros no ofrecen la suficiente robustez al algoritmo en cuanto a la eficiencia. Por otro lado, los resultados obtenidos muestran que la eficiencia no cambia de forma significativa más que en el salto de 50 a 100 trabajos, por lo que podría plantearse un tipo de algoritmo diferente según el tamaño de problema mayor o menor de 100 trabajos.

Finalmente, se ha realizado un estudio sobre robustez bajo el criterio de eficiencia, siguiendo la metodología de Taguchi. Tal como se ha explicado se han considerado dos posibles escenarios. En el primero de ellos se aborda la robustez del algoritmo en cuanto a eficiencia, pero teniendo en cuenta como factores de ruido los diferentes tipos de instancias del problema que podría resolver el algoritmo presentado. En el segundo caso se aborda la robustez del algoritmo en cuanto a eficiencia, pero teniendo en cuenta como factores de ruido el número diferente de procesadores que pudiera emplear el clúster para resolver instancias de problemas del mismo tamaño.

En ambos casos se han obtenido las configuraciones de valores de los parámetros del algoritmo que permiten maximizar la ganancia, es decir, comportarse de manera más robusta frente a variaciones en el entorno. Para cada enfoque, además, se ha realizado un experimento confirmatorio, con los que se han validado los resultados obtenidos.

En definitiva, en este capítulo se ha buscado estudiar y mejorar las características que hacen bueno a un algoritmo paralelo, como son la calidad de las soluciones obtenidas, la mejora en el tiempo de ejecución y la robustez. La calidad y tiempo se han mejorado de forma conjunta mediante el análisis de la eficiencia. La robustez ante diferentes instancias del problema y ante distinto número de procesadores se ha mejorado mediante la metodología de Taguchi.

Una vez investigadas las prestaciones de los algoritmos paralelos con métodos aproximados, parece de interés investigar su combinación con algoritmos exactos. Para ello, en el capítulo siguiente se estudiará un tipo de algoritmo diferente a los

anteriores, en el que se combinan dos métodos de resolución, con objeto de comparar su comportamiento con el de CLMpg y CLMpf en cuanto a eficiencia y calidad de soluciones.

6 Algoritmo paralelo híbrido pBDS

6.1 Introducción

En este capítulo se pretende explorar el posible beneficio de incorporar estrategias de cooperación entre el algoritmo introducido en el capítulo 4 de esta tesis y un algoritmo exacto. Este tipo de estrategias cooperativas se conoce en la literatura como algoritmos híbridos (Bachelet *et al*., 1996). En el capítulo 3 de esta tesis se ha realizado una revisión bibliográfica sobre el problema de secuenciación en flujo uniforme, en la que se han clasificado los métodos de resolución en métodos exactos y aproximados por separado. En este capítulo nos centraremos en métodos híbridos. Para ello se han separando por un lado aquellos que combinan sólo métodos aproximados o heurísticas de los que combinan métodos exactos con métodos aproximados. En ambas clasificaciones se han revisado, en primer lugar, los algoritmos secuenciales y en segundo lugar, los algoritmos paralelos. En el presente estudio no se tienen en consideración algoritmos que combinan diferentes instancias del mismo procedimiento variando sus parámetros, por ejemplo, combinaciones de recocido simulado que emplean dos temperaturas iniciales diferentes. El objetivo que se persigue con el análisis de algoritmos paralelos híbridos es comprobar si supone una mejora sustancial en las prestaciones del algoritmo paralelo frente a las versiones desarrolladas en el capítulo 3 de esta tesis. Para ello se tendrán en cuenta tanto la calidad de las soluciones obtenidas en la versión anterior con un solo procesador, como la eficiencia del algoritmo cuando se emplean más procesadores.

La conveniencia de la cooperación entre metaheurísticas ha quedado patente en diversos estudios – ver por ejemplo Talbi (2002) o Basseur *et al*. (2004) – donde se muestra que las soluciones obtenidas son de mejor calidad y que el algoritmo es

más robusto, es decir, tiene menor sensibilidad a diferentes instancias del mismo problema. Sin embargo es habitual que dichas implementaciones se realicen en un solo procesador, por lo que pueden llevar a tiempos de computación inaceptables cuando se buscan soluciones de calidad. En este contexto, la computación paralela puede emplearse para reducir el tiempo necesario para encontrar soluciones de una calidad dada o bien para mejorar la calidad de las soluciones que se encuentran en el tiempo disponible para realizar la computación. En algún caso, como por ejemplo en Toulouse *et al.* (1999) y posteriormente en El-Abd y Kamel (2005) se propone la categoría "búsqueda cooperativa" intermedia entre algoritmos paralelos que contienen el mismo tipo de algoritmo en todos los procesos y algoritmos híbridos en general.

La estructura del presente capítulo es la siguiente: En la sección 2, se realiza una revisión bibliográfica sobre algoritmos paralelos híbridos aplicados al problema $F_m/prmu/C_{max}$ y similares. Como se justificará más adelante, estamos interesados en combinar un algoritmo exacto con un algoritmo aproximado, para comprobar si ofrece mejores resultados que los algoritmos paralelos desarrollados en los capítulos anteriores de esta tesis. Por ello, en la sección 3 se introduce una implementación secuencial (monoprocesador) de un algoritmo que combina el algoritmo de búsqueda local empleado (CLM) con el método exacto *branch & bound* (B&B). En la sección 4, este algoritmo secuencial, junto con las aportaciones de interés detectadas en la revisión bibliográfica, se empleará como punto de partida para el diseño de un algoritmo paralelo para n procesadores. En la sección 5 se presenta la experimentación realizada con este algoritmo paralelo híbrido y en la sección 6 se resumen las principales conclusiones del capítulo.

6.2 Estado del arte

A modo de resumen, la revisión que hemos realizado en este capítulo puede ser esquematizada con la clasificación que se muestra en la Figura 75. En ella, se tienen en cuenta en primer lugar dos variantes: algoritmos que combinan únicamente métodos aproximados y algoritmos que combinan métodos exactos y aproximados. Dentro de cada una de ellas, se han separado las referencias relativas a algoritmos secuenciales y algoritmos paralelos. Dentro de los algoritmos paralelos, se han se-

parado los procedimientos basados en sistemas de agentes software y dentro de las combinaciones de métodos exactos y aproximados, se han separado aquellos trabajos que proponen arquitecturas que pueden servir de base para el diseño de estos algoritmos paralelos.

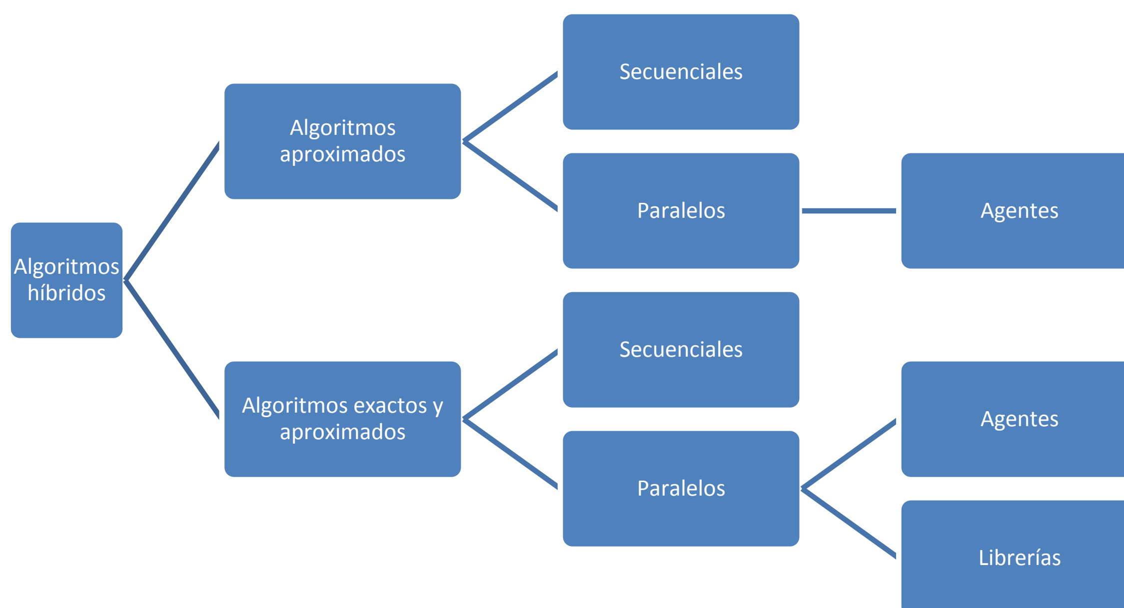


Figura 75: Clasificación propuesta de algoritmos híbridos

Aunque están apareciendo cada vez más trabajos sobre combinaciones de diferentes metaheurísticas (ver Jourdan *et al.*, 2008), pocos autores han combinado métodos exactos y aproximados en paralelo (Cotta *et al.*, 1995, Denzinger y Offerman, 1999, Nwana *et al.*, 2005 o Pessan *et al.*, 2008), siendo la mayoría de las implementaciones híbridas en paralelo combinaciones de métodos aproximados.

6.2.1 Combinaciones de métodos aproximados

6.2.1.1 Algoritmos secuenciales

En este apartado nos hemos centrado en aquellos trabajos que emplean algoritmos secuenciales que combinan metaheurísticas, prestando atención a las que se han aplicado al problema de secuenciación en flujo uniforme o a problemas similares en cuanto a la codificación de soluciones y función objetivo.

Existen revisiones sobre heurísticas híbridas que combinan métodos aproximados y además incluyen algunas referencias a métodos paralelos. Estas son: Talbi (2002), Cotta *et al.* (2005) y Raidl (2006).

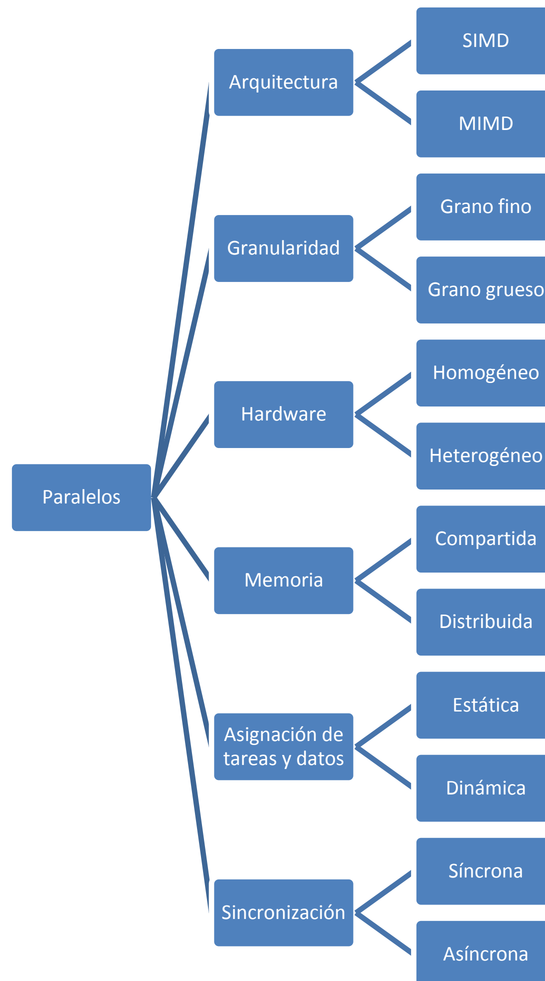


Figura 76: Clasificación de heurísticas paralelas de Raidl (2006)

En Raidl (2006) se realiza una revisión de metaheurísticas híbridas que incluye referencias a métodos paralelos. Estas referencias se clasifican en cuanto a algoritmos paralelos ampliando las clasificaciones de El-Abd y Kamel (2005), Cotta *et al.* (2005) y Blum *et al.* (2005). En cuanto a métodos híbridos, amplía la clasificación de Puchinger y Raidl (2005). Así, clasifica las metaheurísticas híbridas paralelas según el esquema de la Figura 76. En cuanto a la distinción sobre qué metaheurísticas se combinan, se muestran en la Figura 77.

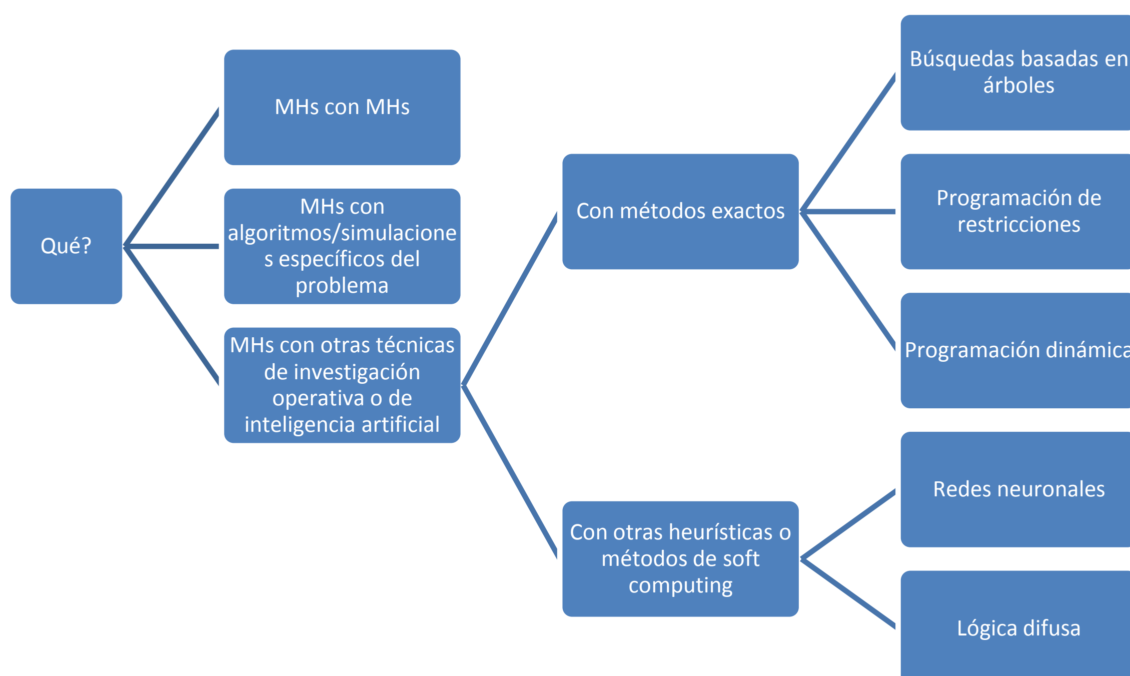


Figura 77: Combinaciones de metaheurísticas de Raidl (2006)

Respecto a combinaciones de metaheurísticas con otros métodos, Raidl (2006) cita la combinación de aquellas con programación de restricciones (*constraint programming* – CP) y la programación lineal entera (*integer linear programming* – ILP).

Algunos de los algoritmos híbridos que se han aplicado con éxito al problema objeto de estudio son los de Yamada y Reeves (1997), donde un algoritmo genético se combina con búsqueda local, el de Stützle (1998c) en el que se mejora un algoritmo basado en colonia de hormigas mediante búsqueda local, llamado MMAS o *Max-Min Ant System* y por último, el de Esquivel *et al.* (2002) que introduce la heurística NEH en algunas fases de la búsqueda mediante colonias de hormigas.

En Nearchou (2004), el autor añade a un algoritmo de recocido simulado características de algoritmos genéticos y de búsqueda local para aplicarlo a la resolución del problema $F_m/prmu/C_{max}$. Los resultados que presenta en cuanto a calidad de solu-

ciones y tiempo de ejecución son del orden o mejores que algunos de los trabajos más conocidos sobre la batería de Taillard (1993).

Técnicas	Referencia
GA+LS	Yamada y Reeves (1997)
ACO+LS	Stutzle (1998)
ACO+NEH	Esquivel *et al.* (2002)
SA+GA+LS	Nearchou (2004)
PSO+LS	Kuo *et al.* (2008)
Varios	Zobolas *et al.* (2008)

Tabla 68: Algoritmos secuenciales que combinan métodos aproximados aplicados a problemas de permutación (GA: *Genetic Algorithms*, LS: *Local Search*, SA *Simulated Annealing*, ACO: *Ant Colony Optimization, PSO: Particle Swarm Optimization*)

En Kuo *et al.* (2008) se emplea optimización mediante cúmulos de partículas (Kennedy y Eberhart, 1995) mejorando el método propuesto por Lian *et al.* (2008) para el problema $F_m/prmu/C_{max}$ con un esquema de diversificación que los autores llaman RK (*Random Key*) y un esquema de intensificación que los autores llaman IE (*Individual Enhancement*). RK consiste en un método para generar soluciones de forma aleatoria, mientras que IE consiste en una búsqueda local que genera la vecindad mediante la permutación de dos trabajos dentro de la secuencia. A pesar de la sencillez de la propuesta, el procedimiento híbrido mejora los resultados del procedimiento original de Lian *et al.* (2008).

El trabajo de Zobolas *et al.* (2008) es muy interesante por aplicarse a nuestro problema y presentar gran cantidad de resultados experimentales tanto de su trabajo como el de otros autores en este mismo problema. El método que proponen, NEGA$_{VNS}$, combina cuatro heurísticas constructivas, NEH (Nawaz *et al.*, 1983), CDS (Campbell *et al.*, 1970), Palmer (Palmer, 1965) y Gupta (Gupta, 1971) con dos

metaheurísticas, VNS (Hansen y Mladenovic, 2001) y GA (Holland, 1992). En su implementación secuencial, $NEGA_{VNS}$ llega a muy buenos resultados con la batería de Taillard (1993) incluso comparados con los de algoritmos paralelos como el de Wodecki y Bozejko (2002) que implementa recocido simulado (SA – *Simulated Annealing*) en paralelo, como se comentó en el capítulo 3 de esta tesis.

En todos los casos se presentan hibridaciones de alguna variante de algoritmo evolutivo para su aplicación al problema de estudio. Se han observado buenos resultados tanto en el caso de combinaciones con heurísticas constructivas como con algoritmos de búsqueda local en alguna de sus variantes. Sobre el tipo de vecindad empleada en la parte de búsqueda local, es llamativo que distintos autores lleguen a conclusiones diferentes, apoyándose en resultados experimentales. Es el caso de Nearchou (2004) para el que la vecindad que ofrece mejores resultados se genera intercambiando dos trabajos de forma aleatoria, mientras que en Zobolas *et al.* (2008) da mejor resultado la inserción de un trabajo en una nueva posición. Sin embargo, los resultados presentados en ambos trabajos no son comparables, ya que aunque los sistemas informáticos empleados son similares, los tiempos de experimentación son muchos mayores en este último, llegando a soluciones de mayor calidad al aplicar su método a problemas de la batería de Taillard (1993).

6.2.1.2 Algoritmos en paralelo

Métodos	Referencia
TS+GA	Matsumura *et al.* (2000)
TS+ACO	Talbi *et al.* (2001)
TS+SA	Mack *et al.* (2004)
LS	Huntley y Brown (1996)
GA+SA	Huntley y Brown (1991)
	Talbi *et al.* (1998a)

Tabla 69: Algoritmos paralelos que combinan métodos aproximados aplicados a problemas de permutación (TS: *Tabu Search*, GA/EA: *Genetic Algorithms/Evolutionary Algorithms*)

En cuanto a métodos paralelos que combinen sólo algoritmos aproximados aplicados al problema $F_m/prmu/C_{max}$ o similares, se han encontrado algunas referencias que combinan las técnicas que se resumen en la Tabla 69.

Comenzando por los algoritmos que aplican alguna variación de búsqueda tabú, Matsumura *et al.* (2000) describen dos versiones de metaheurísticas en paralelo:

- Búsqueda tabú cooperativa en paralelo, cPTS (*Cooperative Parallel Tabu Search*): En este método se toman conceptos de algoritmos genéticos para recombinar las soluciones que se van a intercambiar entre procesos mediante el operador de cruce EX (*edge recombination crossover*). Las soluciones se intercambian seleccionando los procesos que van a participar en el intercambio mediante dos métodos: en el primero se elige el proceso que ha encontrado la mejor solución hasta el momento. En el segundo, de entre dos procesos elegidos al azar, se selecciona aquél con la mejor solución. Este segundo método de selección es el que obtiene mejores resultados en su aplicación al TSP.

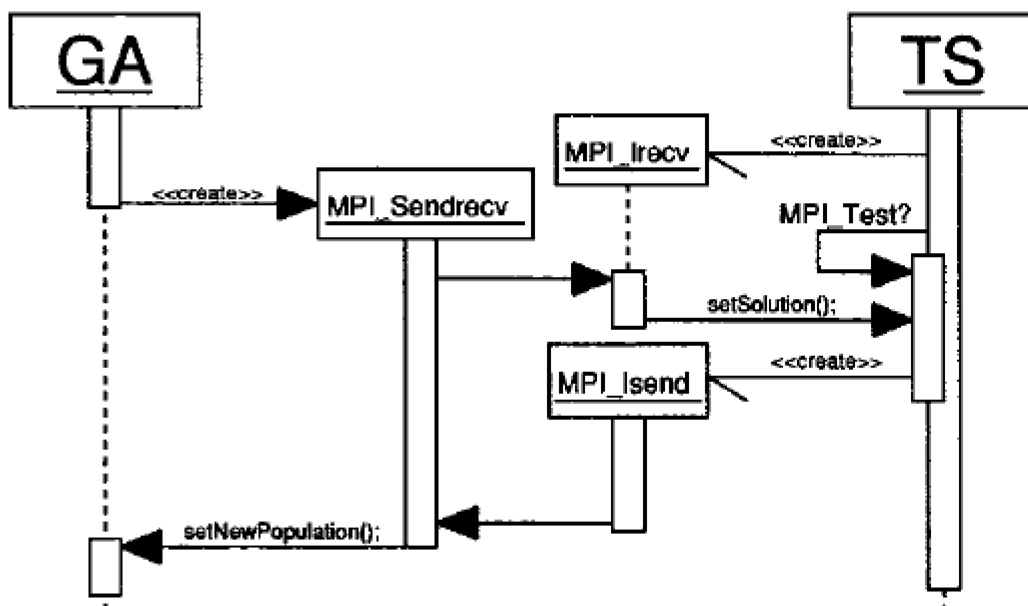


Figura 78: Diagrama de secuencia del intercambio de mensajes en HPMH de Matsumura *et al.* (2000)

- Metaheurística paralela híbrida, HPMH (*Hybrid Parallel MetaHeuristic*) (Figura 78): es otro algoritmo híbrido que combina TS y GA. Emplean únicamente un

proceso para cada metaheurística, siendo el GA quien inicia la comunicación, enviando al TS la mejor solución encontrada cuando se estanca la búsqueda del GA. TS responde enviando la mejor solución encontrada hasta el momento como punto de partida para la nueva población y continúa su búsqueda sustituyendo su mejor solución por la recibida del GA si mejora la función objetivo. En su experimentación con el problema de la mochila, el algoritmo fue capaz de encontrar el óptimo en menos de la mitad del tiempo que el TS secuencial salvo en instancias complejas del problema.

Los autores presentan aceleraciones tanto superiores como inferiores a la lineal. En su caso, emplearon dos PCs conectados mediante *fast ethernet* y con MPI para el intercambio de mensajes. La aceleración inferior a la lineal se obtuvo con los problemas de mayor dificultad. En la Tabla 70 hemos extraído los tiempos promedio y hemos calculado la aceleración y eficiencia correspondientes a su trabajo.

Problema	Búsqueda tabú secuencial	Híbrido en paralelo	Aceleración	Eficiencia
sento2	59.47	19.26	3.09	1.54
weish22	22.13	6.86	3.22	1.61
weish25	27.41	5.65	4.85	2.42
weish26	119.44	20.66	5.78	2.89
weing7	5274.28	3062.60	1.72	0.86
weing8	89.57	49.26	1.81	0.90

Tabla 70: Tiempos promedio, aceleración y eficiencia conseguidas por el algoritmo paralelo propuesto por Matsumura *et al.* (2000)

Talbi *et al.* (2001) aplican una combinación en paralelo de TS y ACO al problema de asignación cuadrática. De forma similar a otros algoritmos paralelos, emplean una arquitectura maestro/trabajador en la que el proceso maestro mantiene una memoria central que contiene la matriz de feromona y la mejor solución encontrada hasta el momento y controla la comunicación, mientras que los procesos trabajadores realizan el proceso de búsqueda de soluciones. Cada proceso trabajador recibe la matriz de feromona, construye una solución completa, le aplica búsqueda tabú y

envía al proceso maestro la solución encontrada y la matriz local de frecuencia. El proceso maestro se encarga entonces de actualizar las matrices de feromona y de frecuencia y la mejor solución para continuar iterando. Los autores remarcan la ganancia en las prestaciones del algoritmo con respecto a algoritmos similares que emplean colonias de hormigas gracias tanto al empleo de TS dentro de cada hormiga como a la computación paralela.

En Mack *et al.* (2004) se combinan en paralelo SA con TS en su aplicación al problema tridimensional de carga de contenedores. Las soluciones de este problema se representan como secuencias de empaquetado, por lo que resulta similar a nuestro problema de estudio. Para el problema de empaquetado, los autores comienzan implementando SA de forma secuencial y observan una mejora en la calidad de las soluciones con respecto al empleo de TS para el mismo problema. Posteriormente construyen un algoritmo híbrido combinando SA con TS, obteniendo así soluciones de mejor calidad. Por último, realizan una implementación en paralelo de este algoritmo híbrido para mejorar aún más la calidad de las soluciones. El procedimiento consiste en aplicar varias iteraciones de TS a las soluciones obtenidas mediante SA, con lo que se evitaría no explorar soluciones prometedoras antes de pasar a otra región del espacio de soluciones en la siguiente iteración de SA. La implementación en paralelo consistió en lanzar la ejecución de varias instancias del algoritmo híbrido de forma independiente, de las que se selecciona la mejor solución tras un tiempo de búsqueda. Se confía en que la aleatoriedad del SA hace que sea baja la probabilidad de exploración de las mismas soluciones. Los autores no obtuvieron mejoras al emplear comunicación entre los distintos procesos.

En Talbi *et al.* (1998a) se combina un algoritmo genético con recocido simulado mediante un esquema que llaman paralelismo adaptativo, que permite equilibrar la carga de trabajo entre los distintos procesos de forma dinámica conforme avanza el funcionamiento del algoritmo. El procedimiento es similar al descrito en el capítulo 3 de esta tesis. En aquel caso, Talbi *et al.* (1998b) emplearon este esquema para implementar un procedimiento de búsqueda tabú en paralelo. Ambos procedimientos se aplicaron al problema de asignación cuadrática.

Por último, los trabajos de Huntley y Brown (1991) y Huntley y Brown (1996) implementan algoritmos híbridos en paralelo para encontrar soluciones de buena calidad a instancias grandes del QAP. El algoritmo SAGA (Huntley y Brown, 1991) combina SA con un GA en paralelo. El algoritmo genético se encarga de generar k soluciones de partida que son mejoradas o maduradas en paralelo por k procesos SA. Estos devuelven las soluciones mejoradas al algoritmo genético, que construye la siguiente generación a partir de estas soluciones “maduradas” y así sucesivamente. Sus resultados consiguieron obtener soluciones óptimas de hasta 18 elementos. El método GALO (Huntley y Brown, 1996) sustituye, en un algoritmo similar al SAGA, el recocido simulado por la búsqueda local para mejorar la calidad de las soluciones obtenidas en cada generación del algoritmo genético. Con este algoritmo paralelo los autores llegaron a las mejores soluciones conocidas en la literatura para el TSP.

Como resumen de este tipo de algoritmos paralelos, volvemos a observar que en la mayoría de ellos se emplean algoritmos evolutivos cuyas soluciones se mejoran o maduran mediante otra heurística, búsqueda tabú o recocido simulado, antes de pasar a una nueva iteración. En el caso de Mack *et al.* (2004) se trata de un algoritmo basado en recocido simulado que genera las soluciones que son mejoradas mediante búsqueda tabú antes de cambiar de temperatura. También se emplea en la mayor parte de ellos un esquema maestro/trabajador con el que se mantiene una memoria común de las mejores soluciones encontradas y se reparte así la carga de trabajos entre procesos. En el caso de Talbi (1998a) este reparto de la carga se realiza dinámicamente conforme se va disponiendo de más procesadores para el cálculo. En su caso se llegan a emplear cien ordenadores personales junto con ordenadores masivamente paralelos. En el otro extremo de paralelización, se encuentra el trabajo de Matsumura *et al.* (2000), que únicamente emplea dos procesos, uno para cada algoritmo.

Por último, se han encontrado bastantes referencias sobre un concepto similar al anterior, como es el de trabajo en paralelo mediante agentes software, también conocidos como *Asynchronous Teams* o *A-Teams* (ver Talukdar y Ramesh, 1992, Gorti *et al.,* 1996, Chang *et al.,* 1997, Sanjai *et al.*, 1998, Talukdar, 1998, Talukdar *et al.,* 1998, Corrêa *et al.,* 2003 y Camponogara y Talukdar, 2004). Mediante el mismo,

se pueden implementar sistemas híbridos sin más que asignar a cada agente un algoritmo distinto. Sin embargo, la mayoría de ellos sólo propone combinaciones de métodos aproximados. Aunque los trabajos citados presentan buenos resultados en su aplicación a problemas de optimización, ninguno de ellos describe aplicaciones al problema objeto de estudio o similares por lo que no se entrará en más detalles sobre su funcionamiento.

6.2.2 Combinaciones de métodos exactos y aproximados

6.2.2.1 Algoritmos secuenciales

Existen algunas referencias de interés que contemplan una revisión sobre la combinación de métodos exactos y aproximados, concretamente los trabajos de (Dumitrescu y Stützle, 2003, Puchinger y Raidl, 2005 y Jourdan *et al.,* 2008). Este último trabajo extiende la clasificación de Talbi (2002) para tener en cuenta métodos exactos y aproximados. Además, todas ellas incluyen alguna referencia sobre métodos paralelos. En la Tabla 71 se resumen las referencias consultadas sobre algoritmos híbridos secuenciales que combinan métodos exactos y aproximados aplicados al problema $F_m/prmu/C_{max}$ o similares.

		Método exacto: B&B
Métodos aprox.	**GA/EA**	Nagar *et al.* (1996)
		Cotta y Troya (2003)
	LS	Haouari y Ladhari (2003)
		Framiñán y Pastor (2008)

Tabla 71: Algoritmos híbridos secuenciales que combinan métodos exactos y aproximados

Comenzando por los métodos que combinan algoritmos genéticos con B&B, el trabajo de Nagar *et al.* (1996) se ha aplicado al problema de secuenciación en flujo regular únicamente con dos máquinas con el doble objetivo de minimizar tanto el *makespan* como el *flowtime*. La forma de combinar los dos algoritmos consiste en que

el algoritmo genético evitará las zonas del espacio que ya han sido podadas por B&B.

El trabajo de Haouari y Ladhari (2003) introduce un algoritmo B&B dentro de un algoritmo LS para encontrar soluciones al problema de secuenciación con el objetivo de minimizar el *makespan* al que llaman BBLS (*Branch and Bound based Local Search*).

En Cotta y Troya (2003) se implementa una combinación de B&B con algoritmos evolutivos, introduciendo B&B como un operador dentro del algoritmo evolutivo que llaman *Forma Analysis*. En este caso, la propuesta consiste en emplear B&B para guiar el operador de recombinación del algoritmo genético y seleccionar así la mejor de las posibles soluciones descendientes de una dada. A este operador le llaman *Dynastically Optimal Recombination Operator*.

En Framiñán y Pastor (2008) se propone el método BDS (*Bound Driven Search*) que también combina un algoritmo B&B con otro de búsqueda local. En este caso, en lugar de introducir uno dentro del otro, se ejecutan varias iteraciones de uno antes de ejecutar el otro. Para ello, se generan soluciones mediante B&B que son mejoradas de forma iterativa por el algoritmo de búsqueda local. La información sobre las zonas del espacio de soluciones podadas por el algoritmo B&B también es tenida en cuenta por el algoritmo de búsqueda.

En conclusión, sobre este tipo de métodos híbridos existen dos tendencias en su diseño: introducir una heurística como un operador dentro de B&B o bien ejecutar un algoritmo a continuación del otro, aprovechando la información negativa sobre zonas del espacio de baja calidad proporcionada por las soluciones podadas por B&B y la información positiva sobre las mejores cotas superiores encontradas por las heurísticas.

Comparando los mejores resultados de los algoritmos híbridos secuenciales revisados para 7 problemas de la batería de Taillard con los resultados de referencia de Taillard (1993), de Nowicki y Smutnicki (1996) y los algoritmos secuenciales B&B LH de Ladhari y Haouari (2005), HPSO de Kuo *et al.* (2008) y NEGA$_{VNS}$ de Zobolas *et al.* (2008) se observa que: para la misma calidad de soluciones, los algoritmos LH y BBLS son los más rápidos en los problemas de 5 máquinas. HPSO es el más rápido en problemas pequeños, de 20x10 y 20x20. LH es el mejor en el problema de 50

trabajos y 10 máquinas y NEGA$_{VNS}$ en el problema de 50 trabajos y 20 máquinas. Lamentablemente, no se pueden comparar de forma directa los tiempos de ejecución de BDS, ya que se emplea como criterio de parada un tiempo máximo de ejecución: 225, 450 y 900 segundos y no se dispone del tiempo empleado en su caso para obtener la mejor solución.

Hay que tener en cuenta que las comparaciones realizadas han de hacerse con cierta cautela, ya que los ordenadores empleados no son similares. Tal como apuntan Zobolas *et al.* (2008) los valores óptimos de la literatura se han conseguido en ordenadores muy potentes, encontrando soluciones en muy corto espacio de tiempo.

Los resultados de los trabajos revisados se han obtenido con ordenadores personales con procesadores P IV de similares características: Ordenados de mayor a menor potencia, NEGA$_{VNS}$ con 2.4 GHz, LH y BDS 1.8 GHz, HPSO 1.73 GHz y BBLS con 1.5 GHz. Por tanto, se puede concluir que ninguno de los algoritmos propuestos supera al de Ladhari y Haouari (2005), aunque sin olvidar que en el problema de mayor dificultad de los analizados, se hace necesaria la cooperación de varias heurísticas que emplean Zobolas *et al.* (2008) para obtener los mejores resultados.

6.2.2.2 Algoritmos en paralelo

En cuanto a métodos paralelos que combinen algoritmos exactos y aproximados, se encuentran escasas referencias aplicadas al problema $F_m/prmu/C_{max}$ o similares.

El artículo de Loiola *et al.* (2007) es una amplísima revisión de métodos de resolución del problema de asignación cuadrática con 365 referencias centradas principalmente en el QAP. En dicho trabajo se incluyen referencias sobre métodos híbridos que combinan métodos exactos y aproximados en paralelo. Sobre métodos exactos distintos de B&B, en la revisión de Loiola *et al.* (2007) se ha empleado programación lineal dinámica sólo para resolver determinadas instancias del problema de asignación cuadrática. También muestran que los métodos de planos cortantes no ofrecen buenos resultados aunque se han empleado para formular algunas heurísticas que emplean programación lineal entera mixta y la descomposición de Benders. Debido a la lentitud de la convergencia de este método, sólo se ha empleado con instancias pequeñas del QAP.

En la revisión de Jourdan *et al.* (2008), se recogen 61 referencias, de las que 3 se refieren a implementaciones en paralelo. En este caso, sólo se hace referencia a B&B como método exacto.

Las referencias sobre algoritmos híbridos en paralelo que combinan métodos exactos y aproximados aplicados a problemas similares al problema objeto de estudio son las recogidas en la Tabla 72.

		Método exacto: B&B
Métodos aproximados	GA/EA	Cotta *et al.* (1995)
		Denzinger y Offerman (1999)
		Pessan *et al.* (2008)
	SA	Nwana *et al.* (2005)
	Varios	Brüngger *et al.* (1999)
		Alba *et al.* (2006)

Tabla 72: Algoritmos paralelos que combinan métodos exactos y aproximados aplicados a problemas de permutación

Nos referiremos en primer lugar, a combinaciones de algoritmos genéticos y evolutivos con B&B:

En Cotta *et al.* (1995), se emplea una combinación de B&B con GA en paralelo para encontrar buenas soluciones al TSP. En este caso, los autores proponen tres variantes para combinar la ejecución de ambos algoritmos en paralelo: en la primera, el algoritmo genético envía las mejores soluciones que encuentra al proceso B&B, de forma que se poden ramas del árbol mientras que el algoritmo B&B envía al GA las mejores soluciones parciales encontradas. Las dificultades de esta versión son la disparidad tanto en tiempo de ejecución como en calidad de soluciones que presentan los dos algoritmos, sobre todo en el caso del TSP, lo que lleva a que se pueda estancar el GA con “superindividuos” o que éste no ayude al B&B. La segunda versión que presentan consiste en incorporar el algoritmo B&B como una parte del GA.

El algoritmo B&B se emplearía a la hora de seleccionar los mejores individuos de entre aquellos generados en cada iteración del GA. Estos autores emplean también como operador el EX (*Edge Recombination*), ya citado en la sección anterior en Matsumura *et al.* (2000). La tercera versión consiste en emplear en paralelo varios GA y un solo B&B, con lo que se mejoraría tanto el tiempo de ejecución como la calidad de las soluciones del GA y compensaría el menor tiempo en el que llega B&B a esa calidad de soluciones. Los mejores resultados obtenidos por los autores indican que la mejor alternativa es la tercera, empleando también comunicación entre los GA.

El trabajo de Pessan *et al.* (2008) combina de nuevo B&B con un algoritmo genético en paralelo para resolver problemas de secuenciación con reinicio. Ambos métodos colaboran entre sí de forma similar a como lo hacen en otros de los trabajos revisados: los procesos B&B reducen el espacio de exploración, mientras que cuando el GA mejora la mejor solución encontrada hasta el momento, ayuda a podar más ramas del árbol B&B.

En segundo lugar, siguiendo con las referencias recogidas en la Tabla 72, detallamos los algoritmos híbridos paralelos que combinan B&B con SA.

PArallel-COoperative Framework - PACO (Nwana *et al.,* 2005) (No confundir con *Parallel Ant Colony Optimization*) combina un proceso ejecutando B&B con otro que ejecuta una versión extendida de recocido simulado que contiene en varios puntos B&B con búsqueda en profundidad: *Mixed Integer Programming Simmulated Annealing* (MIPSA) aplicado a problemas de programación lineal entera mixta. Para combinar ambos métodos, se basan en Darby-Dowman y Little (1998). Sólo emplearon un proceso para cada algoritmo. La implementación emplea PVM como librería de paso de mensajes y la información intercambiada por los dos procesos consiste en las cotas encontradas con MIPSA y en la variable seleccionada en B&B. Los autores no dan indicación, hasta el día de hoy, sobre posibilidades de extender esta arquitectura a más de dos procesos.

De los trabajos revisados, hasta el momento no se han encontrado aplicaciones al problema $F_m/prmu/C_{max}$ mediante algoritmos paralelos híbridos que combinen métodos exactos y aproximados. Sin embargo, aparecen referencias a problemas

similares, tanto problemas de secuenciación con distinta función objetivo como problemas cuyas soluciones se representan como permutaciones.

De manera similar al caso de combinaciones de algoritmos aproximados, también existen sistemas basados en agentes para el caso de la combinación de métodos exactos y aproximados.

TECHS (*TEams for Cooperative Heterogeneous Search*) de Denzinger y Offerman (1999) es un algoritmo paralelo en el que se propone un sistema de agentes que ejecutan por un lado un algoritmo genético y por otro lado B&B para resolver un problema de secuenciación en entornos tipo taller. El esquema propuesto empleando tres agentes (dos emplean B&B y uno GA para compensar el mayor tiempo que emplea B&B en cada transición) mejora los tiempos de ejecución y la calidad de las soluciones con respecto al mejor resultado de cada uno de los agentes por separado. Para seleccionar la información que se va a enviar de un proceso a otro y la que va a recibir cada proceso emplean funciones que llaman *send-referees* y *receive-referees* con lo que consiguen reducir el volumen de información intercambiada y mejoran la eficiencia del algoritmo paralelo. Aunque no presentan cifras, consiguieron en algunos casos valores de aceleración superlineal.

Aunque hemos prestado menos atención a aplicaciones de la computación distribuida, es interesante el trabajo de Bendjoudi *et al.* (2008). Estos autores utilizan B&B y GA en paralelo en un *grid* para resolver el problema $F_m/prmu/C_{max}$. Los dos conceptos que emplean son el *Grid Computing* y el *P2P Computing*. Emplean una arquitectura maestro/trabajador para que, realizando los procesos trabajadores parte del trabajo de comunicación, se evite el cuello de botella que se produciría en el proceso maestro al tratar de controlar él mismo todo el trabajo de los procesos trabajadores.

Los tiempos de ejecución que obtuvieron Bendjoudi *et al.* (2008) para encontrar el óptimo de algunos de los ejemplos de la batería de Taillard experimentando con gran número de procesadores separados geográficamente, se muestran en la Tabla 73. Se muestran entre paréntesis los resultados obtenidos con una versión anterior del algoritmo paralelo que únicamente incluía B&B (ver Bendjoudi *et al.,* 2007). La primera columna muestra el número de procesos de cada tipo y la segunda el tiem-

po de despliegue del programa entre los distintos procesadores que se encargan de su resolución.

np (GA/BB)	Despl	Ta_20_5_2	Ta_20_5_3	Ta_20_10_1	Ta_20_10_2	Ta_100_5_1
6 (1/5)	15	(3) 4	(492) 401	(1815) 1287	(277) 234	-
20 (5/15)	46	(10) 8	(409) 391	(170) 129	(111) 98	-
50 (15/35)	112	(16) 11	(277) 263	(100) 97	(59) 51	-
100 (20/80)	234	17	193	81	50	-
200 (40/160)	504	-	151	77	-	-
300 (60/240)	713	-	152	-	-	7h [5572]
600 (100/400)	1949	-	-	-	-	6h57m[5571]
1500 (300/1200)	4186		-	-	-	6h 57min [5571]

Tabla 73: Resultados en tiempo de despliegue y de ejecución de Bendjoudi *et al.* (2008) y Bendjoudi *et al.* (2007), entre paréntesis.

En versiones con menor número de procesadores, de 6 a 300, emplean una parte de ellos en el B&B paralelo y otros en el GA paralelo. Una vez que terminan su procesamiento los procesos que ejecutan el GA, se unen a los que ejecutan B&B, con lo que terminan todos ejecutando el B&B paralelo. La implementación se basa en emplear Proactive (Proactive Parallel Suite s.f.), un software que permite usar los recursos ociosos de ordenadores de una red mediante una librería que funciona en entorno java.

Otra de las tendencias es la de desarrollar arquitecturas o *frameworks* para implementar algoritmos paralelos híbridos combinando varios tipos de algoritmo: ZRAM y MALLBA son las que han tenido mayor difusión.

ZRAM de Brüngger *et al.* (1999) es una librería desarrollada por Ambros Marzetta y mantenido actualmente por David Bremmer (Bremmer, 2008), que permite el desarrollo de aplicaciones en paralelo que impliquen varias heurísticas diferentes como B&B, LS, BT o TSE. El objetivo de esta arquitectura es lograr eficiencia, portabilidad y facilidad. Para conseguir portabilidad, se basa en tres interfaces que separan cuatro capas de software. La primera capa es la del sistema físico o interfaz de paso de mensajes. La segunda capa está formada por los módulos de servicio o interfaz de virtualización. La tercera capa la forman los motores de búsqueda

o interfaz de aplicaciones. Por último, se encuentra la capa de aplicación. La interfaz de paso de mensajes propuesta, MPI-, es una simplificación del estándar MPI sin comunicadores ni, por lo tanto, operaciones colectivas. En cambio se ha comprobado su funcionamiento sobre distintos equipos como Paragon de Intel, redes de estaciones de trabajo, Cenju-3 de NEC o un solo ordenador MAC. La interfaz de virtualización proporciona, aparte del funcionamiento como un solo procesador virtual, algunas de las prestaciones propias de algoritmos paralelos como son el equilibrado de la carga de trabajo, la detección del final de la ejecución o tolerancia a fallos. La interfaz de aplicaciones contiene librerías para *backtrack*, B&B, búsqueda inversa. De esta forma, el desarrollador únicamente programará una aplicación secuencial en la capa de aplicación que hará uso de algunas llamadas a funciones de búsqueda. En las implementaciones en paralelo de B&B se ha empleado con éxito para resolver instancias del QAP, del *Minimum Vertex* Cover y del TSP.

Alba *et al.* (2006) combinan métodos exactos, heurísticas e híbridos en el proyecto MALLBA. MALLBA es una librería similar a ZRAM que proporciona esqueletos o estructuras de algoritmos de forma que el programador puede concentrarse en la resolución del problema concreto, mientras que las cuestiones relativas al hardware o a la programación paralela quedan en manos de la librería. MALLBA contiene esqueletos de algoritmos exactos como B&B y programación dinámica y esqueletos de heurísticas como búsqueda local, escalada, metrópolis, SA, TS y EA. Cada una de ellas o combinaciones de los mismos se pueden implementar en paralelo con varios tipos de esquema maestro-esclavo.

6.2.3 Conclusiones sobre algoritmos híbridos que incluyen métodos exactos y aproximados

En las revisiones que han hecho otros autores sobre métodos híbridos, se encuentran escasas referencias a este tipo de combinación en paralelo. En concreto, aplicados al problema de estudio únicamente se refiere el método B&B como método exacto.

La forma en la que se ha implementado la combinación entre B&B y métodos aproximados, varía de unos autores a otros, aunque todos hacen que el método aproximado aproveche el conocimiento que el B&B aporta sobre zonas del espacio

que no contienen buenas soluciones y a su vez, éste se aprovecha de la mejora en las cotas superiores encuentran los métodos aproximados.

La arquitectura empleada en los métodos paralelos suele ser tipo maestro/trabajador, empleando un proceso para coordinar la información. La información que se intercambia es, como mínimo, la mejor solución encontrada por el método aproximado y las soluciones parciales podadas por el método exacto. En algún caso, tanto en métodos secuenciales como en paralelos, se ha propuesto incluir el algoritmo B&B como parte del algoritmo aproximado, pero es más frecuente dedicar procesos a B&B y procesos al método aproximado o bien ejecutarlos de forma secuencial.

En los métodos paralelos, la proporción entre procesos que ejecutan el método exacto y el aproximado que da mejores resultados es un solo B&B y el resto métodos aproximados. Nos encontramos con versiones desde dos procesos, uno para B&B y otro para el método aproximado hasta versiones con varios cientos de procesos. En este último caso Bendjoudi *et al.* (2007) dado que los autores van buscando confirmar el óptimo del problema, el número de procesos que ejecuta B&B va aumentando hasta ocupar el total de procesos.

6.3 Descripción del algoritmo secuencial BDS

A continuación, vamos a detallar el desarrollo de un algoritmo paralelo híbrido tratando de implementar las ideas obtenidas de la revisión de la literatura en cuanto a comunicación y a algoritmos híbridos que combinan métodos exactos y aproximados.

El algoritmo original de esta tesis que se va a implementar en paralelo se basa en el método *Bound Driven Search* (BDS) secuencial (Framiñán y Pastor, 2008) que combina un algoritmo aproximado, *Complete Local Search with Memory* o CLM (Ghosh y Sierksma, 2002) que se ha estudiado en detalle en el capítulo 4 de esta tesis y un algoritmo de ramificación y acotación o *Branch and Bound* (B&B)(Land y Doig, 1960) comentado en el capítulo 3.

BDS maneja cinco listas de soluciones. Tres de ellas son las tomadas de CLM: LIVE, DEAD y NEWGEN. La primera, la lista LIVE, contiene las soluciones pendientes de

ser exploradas, DEAD contiene aquellas cuya vecindad ha sido completamente explorada y NEWGEN las que acaban de encontrarse en la vecindad de las soluciones exploradas y mejoran el umbral de aceptación de soluciones. Este umbral se basa en el mejor valor de la función objetivo encontrado hasta el momento. A estas listas se añaden TREE y FORBIDDEN. La lista TREE contiene las soluciones parciales que van a ser evaluadas siguiendo un procedimiento en árbol similar a B&B. La última es la lista FORBIDDEN, que contiene las soluciones parciales que han sido descartadas tras la exploración del árbol por haber quedado por encima de las cotas inferiores planteadas.

6.3.1 Estructura general

El pseudocódigo del algoritmo BDS secuencial de Framiñán y Pastor (2008) es el siguiente:

```
procedure BoundDrivenSearch()
Initialize algorithm, i.e. set Nb, Nc, K, and τ
Sbest ←  ObtainInitialSolution()
do
      for Nb iterations: ExploreTree()
      for Nc iterations: PerformCLM()
      ClearList(LIVE)
while StoppingCondition(parameters) = FALSE
```

La fase de inicialización implica dar valores a los parámetros del algoritmo:

`Nb` es el número de iteraciones realizadas por el algoritmo de exploración del árbol.

`Nc` es el número de iteraciones realizadas por el algoritmo de búsqueda local.

`K` y τ corresponden con los parámetros del algoritmo CLM (ver apartado 4.2).

`Sbest` es la mejor solución encontrada hasta el momento.

La función `ObtainInitialSolution()` genera una solución de partida para el método aproximado mediante una heurística constructiva. En la aplicación al problema de secuenciación en flujo uniforme, la que da mejores resultados es la NEH (Kalczynski y Kamburowski, 2007).

6.3.2 Método exacto

La función `ExploreTree()` selecciona un nodo de la lista TREE de acuerdo a una función de pesos. Si añadir un componente a la solución implica obtener una solución completa, su función objetivo se compara con la mejor hasta el momento. Si es mejor, se ha encontrado una nueva mejor solución. Se añade a la lista LIVE como una solución cuya vecindad debería explorarse con CLM. Si no se obtiene una solución completa, se obtiene una buena solución completa con la función `ObtainCompleteSolutionFromNode()`. El pseudocódigo de la función `ExploreTree()` correspondiente al algoritmo exacto propuesto por Framiñán y Pastor (2008) y a su vez basado en B&B es el mostrado en la Figura 79.

En este pseudocódigo se llaman nodos a las soluciones parciales que se van generando conforme se va ramificando el árbol. `f(S)` es la función objetivo, que para el problema que nos ocupa es el *makespan* de la secuencia `S`.

6.3.3 Método aproximado

La función `PerformCLM()` realiza una iteración del algoritmo CLM: a partir de las K mejores soluciones de la lista LIVE explora todas sus soluciones vecinas que no estén en alguna de las listas, incluyendo en este algoritmo híbrido la lista de soluciones parciales FORBIDDEN generadas por la función `ExploreTree()`. La función `PerformCLM()` añade las que mejoren un cierto umbral de la función objetivo a la lista LIVE para su posterior exploración.

Esta función incluye ahora, antes de la comprobación de la condición de parada, el vaciado de la lista LIVE según se propone en Framiñán y Pastor (2008).

```
procedure ExploreTree()
begin
      if Sbest mejoró en PerformCLM(), then UpdateTree()
      S ←  GetNodeFromTree()
      if S =∅:
detiene el algoritmo y devuelve Sbest como la solución optima (en otro
caso, se garantiza que el problema no tiene solución admisible)
      else:
            borra el nodo seleccionado de la lista TREE
if añadir un componente c a S produce una solución completa:
                  ∀ i que se puede añadir a la solución parcial S:
                        Si ←  añadiendo un componente i a S.
                        if f(Si) < f(Sbest), then Sbest ← Si
                        añadir Si a LIVE
            else
                  ∀ i que se puede añadir a la solución parcial S:
                        Si ←  añade un componente i a S.
                        LB(Si) ←  CalculateLowerBound(Si).
                        if LB(Si) < f(Sbest)
                              añadir Si a TREE.
                        else
                              añadir Si a FORBIDDEN.
                        end
            end
            seleccionar Sr := {Sr : LB(Sr) ≤ LB(Si)}.
            S ← ObtainCompleteSolFromNode(Sr).
            if f(S) < f(Sbest), then Sbest  ← S
            añadir S a LIVE
      end
      if Sbest se ha mejorado, then UpdateTree()
end of procedure ExploreTree()
```

Figura 79: Pseudocódigo del algoritmo exacto de Framiñán y Pastor (2008)

Para ver más detalles sobre este procedimiento y un pseudocódigo del mismo, se puede consultar el capítulo 4 de esta tesis o el artículo original de Ghosh y Sierksma (2002).

6.4 Descripción del algoritmo paralelo híbrido pBDS

Como aportación original de esta tesis, presentamos en este apartado un algoritmo paralelo híbrido que combina un método aproximado y un método exacto. El método exacto que se va a emplear se basa en B&B, ya que se ha visto como el más apropiado para el problema de secuenciación en flujo uniforme en la revisión de la literatura hecha en el apartado 6.3.

El algoritmo aproximado que vamos a emplear en combinación con B&B se basa en el algoritmo CLM (Ghosh y Sierksma, 2002) ya revisado y con el que hemos desarrollado otros algoritmos paralelos en esta tesis.

Como en se ha descrito en capítulos anteriores de esta tesis, el sistema sobre el que se implementará el algoritmo, un clúster de ordenadores (NOW – *Network Of Workstations*) no dispone de memoria compartida, por lo que la información debe compartirse entre procesos mediante un mecanismo de paso de mensajes. Continuamos empleando en este capítulo la versión LAM de MPI como mecanismo de paso de mensajes. En este caso, nos decantamos por una arquitectura de procesos iguales, en la que no existe un proceso maestro o jefe que coordine al resto con objeto de eliminar el posible cuello de botella que supondría la coordinación de todos los procesos por uno solo.

En lugar de combinar un algoritmo dentro de otro, dedicaremos varios procesos a B&B y varios a CLM como por ejemplo en Cotta *et al.* (1995), o bien, en Pessan *et al.* (2008) comunicándose entre sí tanto las soluciones podadas por los procesos B&B como las mejores cotas encontradas por los procesos CLM.

6.4.1 Comienzo de la exploración

En la Figura 80 se presenta un diagrama de secuencia del algoritmo paralelo.

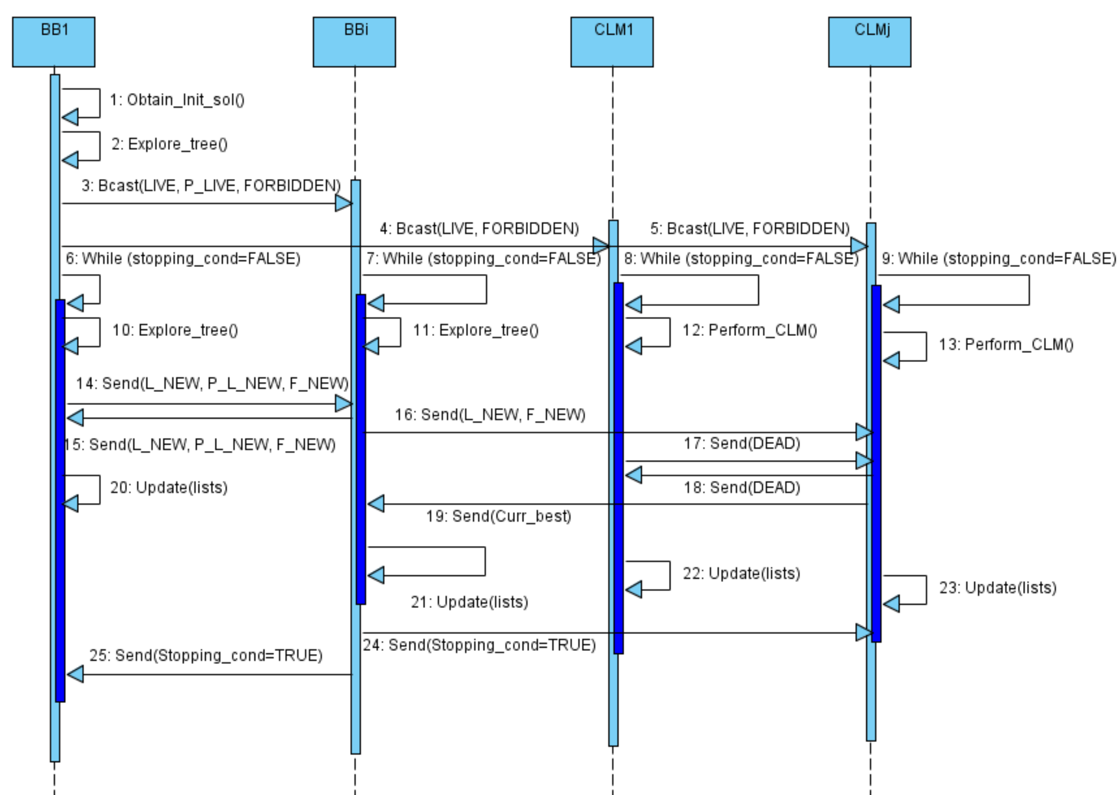


Figura 80: Diagrama de secuencia del algoritmo pBDS propuesto en este capítulo de la tesis

Siguiendo el esquema planteado en el algoritmo secuencial, el algoritmo comienza en uno de los procesos B&B, que genera una solución inicial mediante una heurística constructiva derivada de la NEH (paso 1 en la Figura 80). Esta solución de partida sirve para explorar una primera vez el árbol (paso 2) y generar así tantas soluciones parciales como procesos B&B se hayan iniciado. De esta forma, una vez que se les reparte esta solución inicial (paso 3), los procesos B&B comenzarán la exploración del árbol cada uno desde una rama diferente. Esta elección es más sencilla que la propuesta en el algoritmo $NEGA_{VNS}$ de Zobolas *et al.* (2008) pero garantiza una solución inicial de calidad para el algoritmo, tal como proponen Framiñán y Pastor (2008).

A su vez, los procesos CLM comienzan su exploración a partir de cada una de las soluciones diferentes de la lista LIVE generadas en la primera exploración del primer proceso B&B, que han sido enviadas mediante mensajes de difusión (pasos 4 y 5). El empleo de mensajes de difusión para el envío de las soluciones de partida es la mejor forma de aprovechar la comunicación entre los procesos en el envío de un

dato que debe ser recogido por todos los procesos, ya que el mensaje no se envía desde un solo proceso sino que todos colaboran en la comunicación.

Es interesante resaltar que los procesos B&B envían a los procesos del mismo tipo únicamente la información necesaria para su búsqueda, como son las soluciones parciales generadas (lista P_LIVE), las soluciones parciales pertenecientes a ramas podadas (lista FORBIDDEN) y la mejor solución completa encontrada, que emplearán como cota superior.

Por su parte, los procesos CLM reciben de los procesos B&B tanto las soluciones completas encontradas (formando la lista LIVE) como las soluciones de la lista FORBIDDEN. Los procesos CLM intercambian entre sí soluciones sin explorar (LIVE) y soluciones cuya vecindad ya ha sido explorada (DEAD). No se intercambian las soluciones de la lista NEWGEN, que quedan como un almacenamiento temporal dentro de cada proceso.

A partir de la difusión inicial de soluciones de partida, mientras no se registre una condición de parada (pasos 6 y 7) los procesos B&B realizan una exploración en profundidad del árbol de soluciones (pasos 10 y 11). Al comienzo de cada iteración, cada proceso B&B borra de su lista TREE de soluciones parciales tantas soluciones como el rango del proceso (el número del proceso dentro del total de procesos que están ejecutando el algoritmo paralelo). Con esta estrategia se evita que dos procesos diferentes exploren la misma rama del árbol.

En el caso de los procesos CLM, mientras no se registre la condición de parada (pasos 8 y 9) exploran la vecindad de las soluciones de la lista LIVE (pasos 12 y 13). Al comienzo de cada iteración, cada proceso borra tantas soluciones de LIVE como el número de soluciones total dividido por el número de procesos CLM y multiplicado por el rango del proceso dentro del grupo de procesos CLM. Al estar ordenada la lista LIVE según el valor de *makespan*, se garantiza que cada proceso CLM parte en su exploración de una solución lo suficientemente alejada de la que van a emplear como punto de partida el resto de procesos CLM. El intercambio entre procesos CLM de las soluciones de la lista DEAD evita que un proceso llegue a explorar las mismas soluciones que otros.

6.4.2 Intercambios de información entre procesos

Para diseñar el intercambio de información entre procesos, hemos empleado el concepto de conservación de la energía, tal como se describe en Pacheco (1997) para un algoritmo de búsqueda en árbol. Cuando un proceso termina su exploración, busca más trabajo mirando si tiene mensajes pendientes de otros procesos. Si existen mensajes, los recoge y continúa su trabajo. De esta manera se trata de mejorar la eficiencia del algoritmo, ya que el empleo de procesos tan dispares como los que estamos empleando puede dar lugar a procesadores desocupados mientras otros están sobrecargados de trabajo.

En nuestro caso, cuando un proceso termina su exploración, comprueba si tiene pendientes de recibir mensajes enviados por otros procesos. Si existe algún mensaje, lo recibe y actuará según el contenido del mensaje.

Los procesos B&B reciben mensajes que pueden ser de cinco tipos:

- El mensaje contiene soluciones parciales encontradas por otros procesos B&B (pasos 14 y 15), con lo que actualizará la lista P_LIVE.
- El mensaje contiene soluciones parciales de la lista FORBIDDEN de otros procesos B&B (pasos 14 y 15), con lo que deberá actualizar el contenido de sus listas FORBIDDEN y P_LIVE (pasos 20 y 21).
- El mensaje contiene la mejor solución encontrada por otro proceso CLM (paso 19), con lo que actualizará, si es mejorada, su cota superior y en ese caso también deberá actualizar las listas P_LIVE y FORBIDDEN (pasos 20 y 21).
- El mensaje contiene un aviso de que otro proceso B&B (pasos 14 y 15) ha vaciado su lista P_LIVE, con lo que deberá comprobar si todos los procesos le han enviado ese mensaje, que significará que el algoritmo debe detenerse por haberse llegado al óptimo del problema.
- El mensaje contiene un aviso de que otro proceso ha cumplido la condición de parada (paso 25), el número de iteraciones en el caso de procesos B&B, con lo también deberá detenerse.

Los procesos CLM reciben mensajes de cuatro tipos diferentes:

- El mensaje contiene soluciones parciales que han sido podadas por los procesos B&B (paso 16) con lo que deberá actualizar su tanto lista FORBIDDEN como su lista LIVE, para eliminar soluciones.
- El mensaje contiene soluciones completas encontradas por procesos B&B (paso 16) o por procesos CLM (lista LIVE, pasos 17 y 18) con lo que deberá actualizar su lista LIVE con las nuevas soluciones (pasos 22 y 23).
- El mensaje contiene soluciones cuya vecindad ya ha sido explorada por otro proceso CLM (pasos 17 y 18), con lo que se almacenarán en la lista DEAD y se revisará la lista LIVE, de forma que se eliminen las soluciones ya exploradas por otros procesos (pasos 22 y 23).
- El mensaje contiene un aviso de que otro proceso ha encontrado una condición de parada (paso 24), con lo que deberá detenerse.

Esta estrategia de intercambio de información tanto positiva, sobre la mejor solución encontrada, como negativa, bien sobre soluciones ya exploradas o bien sobre soluciones podadas, ha sido empleada por varios autores como por ejemplo en el trabajo de Denzinger y Offerman (1999). Cuando se mejora en un proceso B&B la mejor solución existente hasta el momento, al enviarla al proceso CLM actualizará la lista LIVE, dirigiendo la búsqueda hacia esa zona del espacio de soluciones.

La finalización del algoritmo viene dada por dos condiciones:

- La más evidente se da cuando se ha encontrado el óptimo. Para verificar esta condición, los procesos B&B mantienen un vector binario con el número de procesos como dimensión, de manera que en cada componente i se registra si el proceso i ha vaciado ya su árbol de soluciones o no. Este vector se va actualizando conforme se reciben mensajes desde los procesos B&B, de forma que si un proceso ha vaciado su lista pero recibe soluciones de otro para continuar la exploración, se registre dicho proceso como no vacío.
- La segunda condición de parada que hemos considerado es la realización de un número máximo de iteraciones por parte de alguno de los procesos B&B, que avisará al resto mediante un mensaje que provocará la detención del resto de procesos.

6.5 Experimentación

Con objeto de comprobar la escalabilidad del algoritmo paralelo que hemos propuesto en base a su eficiencia, se ha realizado una experimentación exhaustiva. Además, esta experimentación se empleará también para obtener unos valores apropiados de los parámetros del algoritmo que, en una etapa posterior podrían ser ajustados mediante la estrategia propuesta en capítulos anteriores.

La experimentación se ha llevado a cabo en el clúster de ordenadores descrito con detalle en el capítulo 4 de esta tesis. Son 12 ordenadores con procesador P IV a 3.2 GHz y conectados entre sí mediante una red *gigabit Ethernet*. El sistema operativo es Linux y la implementación se ha realizado en lenguaje C junto con la librería LAM-MPI de paso de mensajes.

6.5.1 Parámetros de exploración

La elección de los valores de los parámetros se ha basado tanto en los obtenidos en los capítulos anteriores de esta tesis como en los empleados por otros autores en algoritmos híbridos tanto secuenciales como paralelos.

- Parámetros del umbral de aceptación de soluciones: Se ha comprobado que la combinación de valores $\alpha_0 = 0.3$ y $\beta = 0.3$, tal como se derivaba del estudio del capítulo 5 obtiene mejores resultados que los valores de los parámetros iguales a cero propuestos por Framiñán y Pastor (2008). Esto supondría en la práctica aceptar únicamente aquellas soluciones de la vecindad que mejoren la mejor función objetivo encontrada hasta el momento.
- Tamaño máximo de memoria en el algoritmo CLM: No se ha limitado el tamaño de memoria en los procesos CLM, con lo que se explora toda la vecindad de cada solución. Este valor evita que los procesos CLM se estanquen durante demasiado tiempo, descompensando la comunicación con los procesos B&B, que han realizado muchas más iteraciones en el mismo espacio de tiempo. Cuando sucede esto, no se produce un intercambio de mensajes entre procesos B&B y procesos CLM hasta pasadas bastantes iteraciones de los procesos B&B.

- Número de iteraciones en cada proceso B&B: Los procesos B&B realizan 5 iteraciones locales antes de intercambiar datos con el resto de procesos y comenzar una nueva iteración global. En cada iteración local, cada proceso B&B realiza la exploración de un nivel del árbol de soluciones, seleccionando de entre las soluciones de ese nivel la mejor rama según una función de pesos y profundizando en esa rama hasta llegar a una solución completa. La función de pesos se ha tomado según proponen Framiñán y Pastor (2008) a su vez tomado de Ladhari y Haouari (2005) y designada por LB1 por su facilidad de implementación. Esta cota inferior se calcula relajando el problema $F_m/prmu/C_{max}$ para convertirlo en un problema de una máquina $1|r_j, q_j|C_{max}$ asignando en cada máquina M_k (k=1, …,m) instantes de comienzo r_j a cada trabajo basados en la suma de los tiempos de proceso en las máquinas precedentes e instantes de terminación q_j como la suma de los tiempos de proceso en las máquinas posteriores a M_k:

$$r_j = \sum_{i=1}^{k-1} p_{ij} \;\; si\; k = 2, \dots, m$$
$$r_j = 0 \; si \; k = 1$$

$$q_j = \sum_{i=k+1}^{m} p_{ij} \;\; si\; k = 1, \dots, m-1$$
$$q_j = 0 \; si \; k = m$$

La cota inferior denotada como LB1 en Ladhari y Haouari (2005) se ha elegido para cada máquina M_k como:

$$LB1_k = \min_{j \in J} r_{kj} + \sum_{j \in J} p_{kj} + \min_{j \in J} q_{kj}$$

Y la que se selecciona para el problema es el máximo de las anteriores:

$$LB1 = \max_{1 \leq k \leq m} LB1_k$$

- Número de iteraciones en cada proceso CLM: los procesos realizan una iteración local antes de intercambiar datos con el resto de procesos y comenzar una nueva iteración global. En cada una de las iteraciones, seleccionan K soluciones y exploran su vecindad.

6.5.2 Parámetros del algoritmo paralelo

- Número de procesos de cada tipo: Tras una experimentación previa, se ha confirmado la propuesta de autores como Cotta *et al.* (2005) en el que se propone, como mejor proporción, el empleo de un solo proceso B&B y el resto empleando el algoritmo aproximado CLM. Con esta configuración se han conseguido los mejores valores de tiempo de ejecución para un número variable de procesos en la batería estudiada.
- Número de soluciones intercambiadas entre procesos: El número de soluciones intercambiadas se ha tomado diferente del empleado en las versiones de algoritmo paralelo implementadas en capítulos anteriores de esta tesis. En este caso, tras una experimentación previa, en lugar de un pequeño número de soluciones, se ha observado que el valor óptimo es de 50 soluciones. Valores superiores a 100 soluciones provocan que en ocasiones se trunquen los mensajes entre procesos y no se reciban completos. Valores inferiores han dado como resultado peores tiempos para llegar a una calidad dada de las soluciones. Esta observación se debe a que con un número pequeño de soluciones intercambiadas, los procesos no tienen tanto conocimiento sobre las zonas del espacio que ya han sido exploradas por procesos CLM o las zonas que han sido podadas por procesos B&B.

6.5.3 Resultados experimentales

Con objeto de comprobar la eficiencia del algoritmo paralelo, se han realizado 3 repeticiones de cada uno de los problemas de la conocida batería de Taillard (1993) tomando como criterio de parada el valor de la función objetivo alcanzado en Framiñán y Pastor (2008). Este criterio de parada es menos exigente que tratar

de llegar a las mejores cotas conocidas pero a su vez, es más complejo que el criterio tomado en capítulos anteriores de esta tesis. A pesar de haber realizado únicamente 3 réplicas de cada uno de los problemas, no se han observado variaciones significativas en los tiempos obtenidos, lo que da una indicación de la robustez del algoritmo pBDS.

Se han realizado pruebas con 3, 6, 9 y 12 procesos, de forma que se puedan obtener, para cada uno de los problemas, los valores tanto de la eficiencia $e(n)$ como de eficiencia incremental generalizada $gie(n,m)$, cuya expresión es la siguiente:

$$gie(n,m) = \frac{n \cdot t(n)}{m \cdot t(m)} \tag{23}$$

Como ya se comentó en el capítulo 2, este valor mide el aprovechamiento de los procesadores que se añaden cuando se pasa de n procesadores a m procesadores. $t(n)$ es el tiempo de ejecución del algoritmo empleando n procesadores.

Para calcular los valores de eficiencia, se han comparado los tiempos de ejecución necesarios con varios procesadores para llegar a la cota propuesta con el empleado ejecutando el algoritmo en un solo procesador con un proceso B&B y un proceso CLM de forma que la comparación se realice con el algoritmo más parecido posible al algoritmo paralelo. De esta forma, aun no comparando el algoritmo con el mejor de la literatura, se obtiene una idea más clara de la escalabilidad del mismo.

6.5.3.1 Eficiencia de pBDS

Los valores de eficiencia obtenidos con este algoritmo, se resumen en la Tabla 74 para 3, 6, 9 y 12 procesos.

	Eficiencia según número de procesos			
	3	**6**	**9**	**12**
20x5	2.82	1.56	1.14	0.86
20x10	2.07	1.79	1.24	1.37
20x20	0.99	0.60	0.93	0.86
50x5	1.31	0.93	0.54	0.39
50x10	2.93	1.95	1.53	1.07
50x20	0.70	0.46	0.40	0.37
Prom.	**1.80**	**1.22**	**0.96**	**0.82**

Tabla 74: Eficiencia del algoritmo pBDS

Los valores de eficiencia incremental generalizada se muestran en la Tabla 75:

	3 a 6	3 a 9	3 a12	6 a 9	6 a 12	9 a 12
20x5	0.80	0.64	0.51	0.79	0.54	0.80
20x10	2.08	1.34	1.88	0.71	0.64	0.96
20x20	0.65	0.91	0.81	1.61	0.65	0.98
50x5	0.56	0.37	0.27	0.70	0.49	0.74
50x10	1.32	1.60	1.34	2.41	0.50	0.75
50x20	0.82	0.76	0.66	1.17	0.67	1.01
Promedio	**1.04**	**0.94**	**0.91**	**1.23**	**0.58**	**0.87**

Tabla 75: Eficiencia incremental generalizada del algoritmo pBDS

Los resultados por cada uno de los problemas de la batería se recogen en el Anexo 6.

6.5.3.2 Comparación de los resultados

Se van a comparar los resultados de eficiencia de los tres algorimtos diseñados aunque las pruebas que se han realizado en este capítulo no se corresponden con el número de procesadores empleado con el algoritmo CLMpg. , sí se dispone de ambos valores para 6 y 12 procesadores, ver Los valores de eficiencia obtenidos en el algoritmo CLMpg son los tomados de la Tabla 25 y los del CLMpf son los tomados

de la Tabla 52. En la Tabla 76 se puede observar que los valores de eficiencia conseguidos con el algoritmo pBDS son superiores en la mayoría de las ocasiones.

		CLMpg						CLMpf				pBDS			
	np	**2**	**4**	**6**	**8**	**10**	**12**	**3**	**6**	**9**	**12**	**3**	**6**	**9**	**12**
problema	**20x5**	0.26	0.45	0.38	0.50	0.38	0.04					2.82	1.56	1.14	0.86
	20x10	0.31	0.56	0.64	0.53	0.43	0.63					2.07	1.79	1.24	1.37
	20x20	0.46	0.73	0.36	0.47	0.30	0.41	1.05	0.47	0.28	1.98	0.99	0.61	0.93	0.85
	50x5	0.16	0.19	0.24	0.22	0.08	0.12	0.79	0.30	1.51	1.10	1.31	0.93	0.54	0.39
	50x10	0.29	0.26	0.65	0.27	0.33	0.26	0.85	0.59	0.34	0.24	2.93	1.95	1.53	1.07
	50x20	0.77	0.85	0.59	0.81	1.32	0.76	1.04	0.74	0.49	0.37	0.70	0.46	0.40	0.37
promedio		**0.44**						**0.76**				**1.20**			

Tabla 76: Comparación de la eficiencia de los algoritmos CLMpg, CLMpf y pBDS

En la Figura 81 y siguientes se muestran los datos comparados de eficiencia de los algoritmos CLMpg, CLMpf y pBDS.

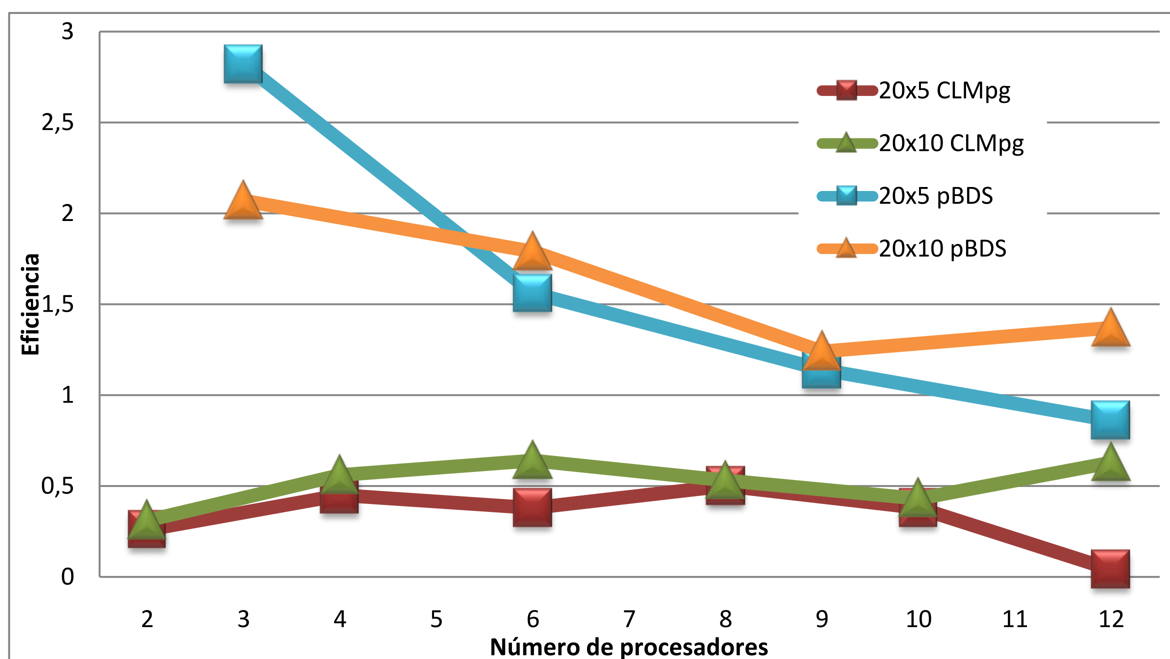


Figura 81: Comparación de la eficiencia de los algoritmos CLMpg y pBDS (20x5 y 20x10)

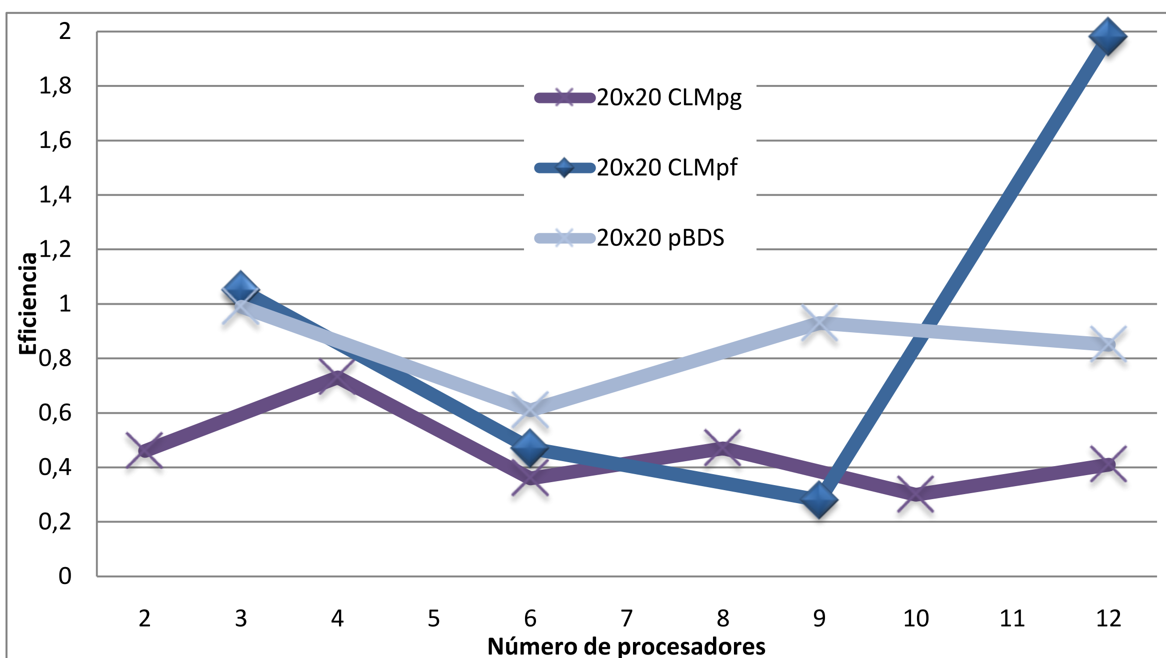


Figura 82: Comparación de la eficiencia de los algoritmos CLMpg, CLMpf y pBDS (20x20)

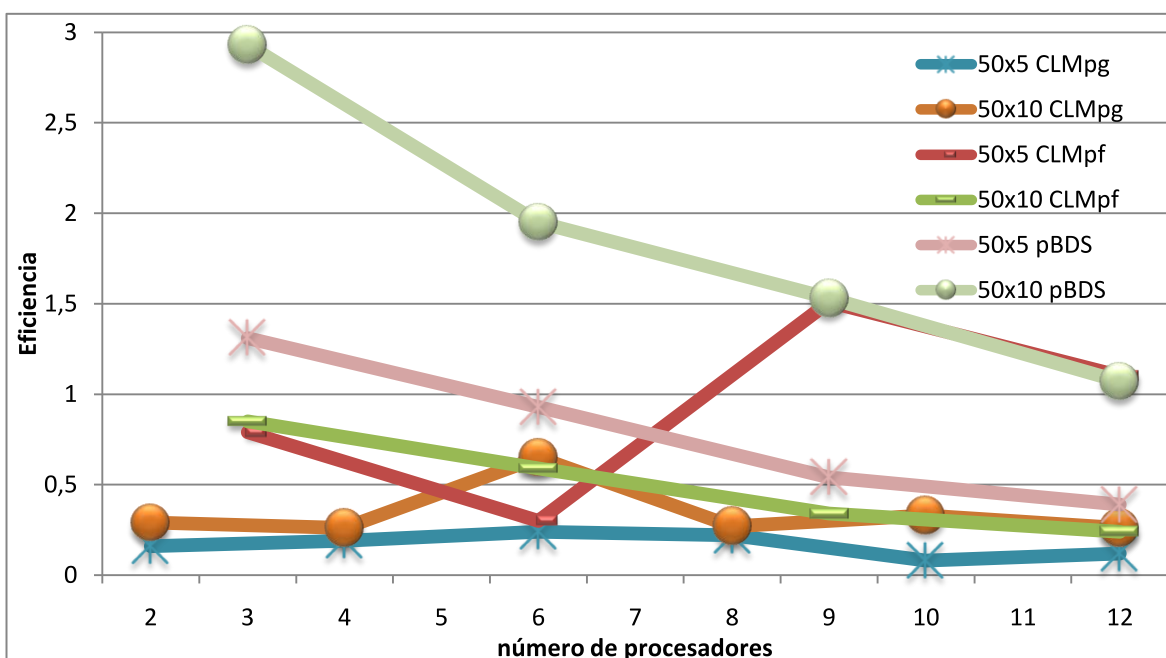


Figura 83: Comparación de la eficiencia de los algoritmos CLMpg, CLMpf y pBDS (50x5 y 50x10)

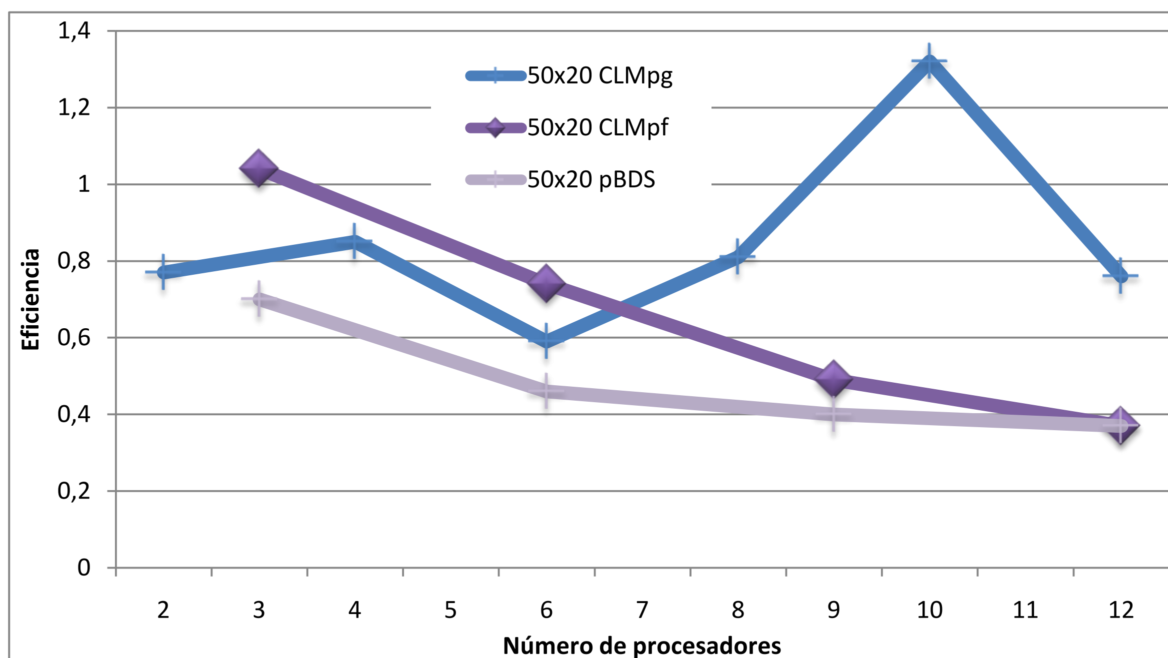


Figura 84: Comparación de la eficiencia de los algoritmos CLMpg, CLMpf y pBDS (50x20)

En estas figuras se puede observar, además de sus mejores valores, que la eficiencia del algoritmo pBDS se degrada mucho más lentamente con el número de procesadores que el algoritmo CLMpf, llegando a resultados de aceleración superlineal en el caso de la serie de problemas 50x10.

6.6 Conclusiones

En este capítulo se ha diseñado e implementado un algoritmo híbrido paralelo, pBDS, que combina un método exacto basado en ramificación y acotación y un algoritmo aproximado basado en búsqueda local.

A diferencia de los anteriores diseños desarrollados en esta tesis, el de este capítulo emplea un tipo de comunicación asíncrona que no detiene la ejecución de los procesos cuando se envían o reciben datos de otros procesos.

Se ha empleado el concepto de conservación de la energía, descrito por ejemplo en Pacheco (1997) para mejorar la eficiencia del algoritmo equilibrando la carga de trabajo entre procesos. Esta es una de las mayores dificultades de la implementación en paralelo de algoritmos de ramificación. Según el concepto de conservación de la energía, cada proceso, al terminar una iteración, comprueba si existen mensa-

jes pendientes de ser recibidos, de forma que pueda continuar su trabajo a partir de una nueva solución o detener definitivamente la exploración si otro proceso ha alcanzado la condición de parada. Esta condición de parada puede consistir en alcanzar un número dado de iteraciones por los procesos que realizan el algoritmo exacto, en que estos procesos alcancen el óptimo o bien en que se alcance una calidad dada de la solución. Si no se ha recibido la señal de parada, una vez recibidos todos los mensajes pendientes, el proceso envía los resultados de su exploración al resto de procesos y continúa su trabajo.

En el diseño de pBDS se han seguido también las ideas propuestas en Nwana *et al.* (2005), Darby-Dowman y Little (1998) o Pessan *et al.* (2008) sobre el intercambio de información tanto positiva como negativa entre los procesos que realizan la exploración. De este modo, los procesos que realizan la búsqueda local obtienen de los procesos que exploran el árbol de soluciones tanto soluciones completas desde las que iniciar la búsqueda como soluciones parciales que han sido podadas y no conducen, por tanto, a buenas soluciones. Los procesos que exploran el árbol de soluciones obtienen información positiva de los procesos que realizan la búsqueda local cuando éstos obtienen una nueva cota superior para el problema, lo que les permite podar nuevas ramas del árbol.

Se ha estudiado la proporción entre procesos que realizan el algoritmo exacto y procesos que realizan la búsqueda local, llegándose a la conclusión de que cuanto mayor sea el número de procesos que realizan la búsqueda local, el tiempo de ejecución es menor para un mismo número total de procesos. Esta es también la mejor combinación de procesos obtenida en trabajos como el de Cotta *et al.* (1995). El tiempo total necesario para llegar a soluciones de una calidad dada ha resultado ser muy sensible al número de procesos que ejecutan el algoritmo exacto.

Frente a los algoritmos CLMpg y CLMpf desarrollados en los capítulos anteriores de esta tesis, el algoritmo pBDS ha resultado ser más eficiente tanto al estudiar la eficiencia incremental generalizada como la eficiencia teniendo en cuenta el tiempo de ejecución del algoritmo en un solo procesador ejecutando un proceso B&B y un proceso CLM.

7 Conclusiones y líneas futuras de investigación

En este trabajo se ha estudiado el problema de secuenciación de trabajos en flujo uniforme con el objetivo de minimizar el tiempo máximo de terminación o *makespan* mediante computación paralela.

En primer lugar se ha revisado el estado del arte sobre computación paralela, tanto en lo que se refiere a las ventajas y los inconvenientes como a las medidas que habitualmente se emplean para establecer la eficiencia de los algoritmos paralelos. También se han revisado las distintas plataformas de computación paralela disponibles en lo que se refiere a la arquitectura hardware y software. Este estudio ha servido de base para seleccionar y adquirir la plataforma de computación paralela que se ha empleado en este trabajo.

A continuación, se ha descrito el estado de la cuestión relativo a los algoritmos de computación paralela, extendiendo la clasificación de Crainic *et al.* (1997). En un paso posterior, se han revisado de forma exhaustiva los algoritmos paralelos disponibles tanto para el problema $F_m/prmu/C_{max}$ como para problemas similares respecto al tipo de vecindad, estructura de las soluciones, u otras variantes que pudieran ser adaptadas al problema objeto de estudio. Dichos problemas se han descrito de acuerdo a la clasificación para los algoritmos paralelos analizados anteriormente.

A modo de resumen, se ha comprobado la existencia en la literatura de numerosos trabajos en el campo de la optimización orientados a la obtención de soluciones de calidad mediante diferentes procedimientos. Son menos, aunque también numerosos, los trabajos que tratan de aprovechar las prestaciones de sistemas paralelos. Por el

contrario, se han encontrado escasas referencias centradas en la resolución o en la aproximación de soluciones al problema $F_m/prmu/C_{max}$ mediante algoritmos paralelos.

La revisión del estado del arte nos ha servido para diseñar e implementar dos algoritmos paralelos, CLMpg y CLMpf, basados en un algoritmo de búsqueda local bastante flexible, CLM. El primero está orientado a sistemas intrínsecamente de grano grueso como clústeres de PCs y se ha empleado para encontrar un ajuste inicial de parámetros. Para lograrlo, se ha seguido una experimentación para estudiar tanto su eficiencia cuando se aplica a la resolución del problema $F_m/prmu/C_{max}$ como para lograr el mejor compromiso posible entre calidad de soluciones y tiempo de ejecución del algoritmo. Como resultado, se han seleccionado los parámetros significativos y sus niveles para conseguir un compromiso entre tiempo de ejecución y calidad de soluciones. Los resultados en cuanto a eficiencia son, por lo general inferiores a los obtenidos con sistemas masivamente paralelos, como las redes de *transputers* y superiores a los obtenidos en sistemas virtuales.

Así, se ha rediseñado el algoritmo CLMpg de forma que se aumenta la comunicación entre procesos, pasando a un algoritmo de grano fino, CLMpf. De esta forma se ha buscado un doble objetivo. Por un lado, el aumento de comunicación entre procesos lleva a una mejor exploración del espacio de soluciones, con lo que se pretenden conseguir mejores soluciones en un tiempo menor. Por otro lado, se han modificado los parámetros de funcionamiento de forma que se ha mejorado su eficiencia.

El aumento de comunicación entre procesos ha traído también, como efecto indeseado, una mayor degradación de la eficiencia del algoritmo cuando se aumenta el número de procesadores. Esta degradación es debida, principalmente a la necesidad de la sincronización entre procesos. En el anexo 6 se muestran los valores de eficiencia resultando más eficiente en promedio el algoritmo de grano grueso.

Siguiendo con la búsqueda de la mayor eficiencia posible del algoritmo paralelo y sin perder de vista el objetivo de lograr soluciones de calidad en un tiempo de computación reducido, se ha propuesto una metodología para ajustar un algoritmo paralelo. El objetivo de esta metodología ha sido obtener un algoritmo paralelo no solo con la máxima eficiencia sino con un comportamiento robusto tanto ante dife-

rentes instancias del problema de estudio como para su funcionamiento en clústeres con diferente número de procesadores.

La necesidad de tal metodología se plantea por la complejidad del ajuste de parámetros de un algoritmo paralelo. No se debe olvidar que, las metaheurísticas requieren de un ajuste de sus parámetros y criterios de funcionamiento para obtener las mejores soluciones para cada problema concreto en un tiempo razonablemente corto. En el caso de un algoritmo paralelo, hay que añadir parámetros y características propias del mismo como número de procesadores, puntos de intercambio de información o contenido de la información intercambiada entre procesos.

La metodología propuesta nos ha llevado a un mayor conocimiento del comportamiento del algoritmo paralelo ante dos factores como son el tamaño del problema y el tamaño del sistema informático sobre el que funciona el algoritmo. Este conocimiento ha permitido seleccionar los parámetros de exploración y del algoritmo paralelo más adecuados al problema de secuenciación, mejorándose los valores de eficiencia y robustez del algoritmo (ver apartado 5.6).

Continuando con nuestra investigación sobre un algoritmo paralelo eficiente aplicado a problemas de secuenciación en flujo uniforme, hemos diseñado un algoritmo paralelo híbrido que combina un método exacto basado en Branch & Bound y un método aproximado basado en CLM. Para mejorar la eficiencia del algoritmo, se ha prescindido de un proceso que coordine al resto y de la sincronización en la comunicación, con lo que se ha logrado un algoritmo que supera en eficiencia a los presentados en capítulos anteriores y ofreciendo soluciones de mayor calidad.

7.1 Contribuciones

7.1.1 Aportaciones de la tesis

- Se han revisado y clasificado los algoritmos paralelos basados en metaheurísticas y aplicados a problemas de secuenciación en flujo uniforme con el objetivo de minimizar el *makespan*.
- Se ha seleccionado y configurado una plataforma informática basada en software libre adecuada para la implementación de dichos algoritmos paralelos.

- Se han diseñado e implementado tres algoritmos paralelos para su aplicación al problema $F_m|prmu|C_{max}$. Los dos primeros, CLMpg y CLMpf están basados en un algoritmo aproximado, mientras que el tercero, pBDS, es un algoritmo paralelo híbrido que combina un método exacto y un método aproximado.
- Se ha diseñado una metodología para el ajuste de parámetros de los algoritmos paralelos desarrollados de forma que se consiga la máxima eficiencia y la máxima robustez ante variaciones en el tamaño del problema y ante variaciones en el número de procesadores empleados.
- Se ha realizado una experimentación exhaustiva con cada uno de los algoritmos diseñados, de forma que se han obtenido resultados sobre la eficiencia de los mismos y su robustez.
- Se han revisado y clasificado los algoritmos paralelos híbridos que combinan tanto diferentes metaheurísticas como metaheurísticas y métodos exactos.

7.1.2 Trabajos derivados de esta tesis

- León Blanco, J.M.; González Rodriguez, P.L.; Pérez González, P.; Molina Pariente, J.M.; A survey of parallel hybrid applications to the permutation flow shop scheduling problem and similar problems. 3rd International Conference on Industrial Engineering and Industrial Management. Barcelona-Terrassa, September 2nd-4th 2009.
- León Blanco, J.M.; Framiñán Torres, J.M.; González Rodríguez, P.L.; Ruiz Usano, R. A prototype of parallel hybrid algorithm. 3rd International Conference on Industrial Engineering and Industrial Management. Barcelona-Terrassa, September 2nd-4th 2009.
- León Blanco, J.M.; Framiñán Torres, J.M.; González Rodríguez, P.L.; Pérez González, P.; Ruiz Usano, R. (2007). Comparación de Dos Medidas de la Eficiencia de Algoritmos Paralelos. International Conference on Industrial Engineering and Industrial Management, Cio 2007. Madrid, pp. 1343-1348
- León Blanco, J.M.; Framiñán Torres, J.M.; González Rodríguez, P.L.; Pérez González, P.; Ruiz Usano, R. (2006). Influencia de la Granularidad en las

Prestaciones de Algoritmos Paralelos. X Congreso de Ingeniería de Organización. Valencia, pp. 217-218

- León Blanco, J.M.; González Rodríguez, P.L.; Ruiz Usano, R. (2005). Algoritmo Paralelo Basado en Agentes Inteligentes Aplicado al Problema de Secuenciación de Trabajos en Flujo Uniforme. IX Congreso de Ingeniería de Organización. Gijón (Asturias), pp. 173-174.
- León Blanco, J.M.; Framiñán Torres, J.M.; González Rodríguez, P.L.; Ruiz Usano, R. (2004). Implementación de Meta-Heurísticas en Paralelo Mediante LAM-MPI: Situación Actual y Perspectivas de Futuro. VIII Congreso de Ingeniería de Organización. Congreso de Ingeniería de Organización (8). Núm. 8. Leganés, Madrid. Universidad Carlos III, pp. 531-540.

7.2 Futuras líneas de trabajo

El punto inicial en el que debería continuar nuestra investigación es el ajuste fino de los parámetros del algoritmo paralelo hibrido. La metodología diseñada en los capítulos anteriores de esta tesis y basada en diseño experimental llevó a buenos resultados con el algoritmo CLMpf, lo que hace suponer que también llevaría a mejorar la eficiencia y robustez del algoritmo pBDS.

A partir de aquí se pueden ampliar los resultados obtenidos a otras metaheurísticas híbridas, el ajuste automático de los parámetros del algoritmo desarrollado o, ya más adelante, nuevas topologías de comunicación entre procesos.

La investigación de las tecnologías relacionadas con la inteligencia artificial, por ejemplo sistemas de agentes inteligentes, se encuentra en nuestro caso en una fase muy inicial. Se han obtenido algunos resultados preliminares que apuntan hacia las ventajas en cuanto a ajuste de los parámetros de búsqueda que tendría una arquitectura de nuestro algoritmo paralelo basado en agentes inteligentes, cuya inteligencia estuviera basada en una red neuronal. De este modo, se aprovecharía mejor la capacidad de cómputo disponible y se reduciría el tiempo de experimentación necesario para un ajuste óptimo de los parámetros del algoritmo paralelo. Hay que hacer notar que, en el caso de un algoritmo paralelo, no sólo es necesario el ajuste de los parámetros del algoritmo de búsqueda local, sino también los que están

relacionados con el algoritmo paralelo, como el número de procesos empleados, la cantidad de información que deben intercambiar estos procesos y con qué frecuencia entre otros.

La topología virtual de la comunicación puede representar mejoras en la exploración del espacio de soluciones cuando procesos que se encuentran en regiones cercanas intercambian soluciones entre sí. En este caso, la investigación se encuentra todavía en una fase muy inicial.

Puede ser de interés, el desarrollo de una metodología que facilite la extensión de los resultados obtenidos con este trabajo a otras metaheurísticas como las dotadas de inteligencia tipo colonia de hormigas, algoritmos genéticos o redes neuronales.

Deben perfeccionarse numerosos aspectos del algoritmo presentado. Entre ellos cabe citar los siguientes: estancamiento en mínimos locales, mejoras en la comunicación o cambios en la arquitectura del algoritmo paralelo.

El riesgo de estancamiento en mínimos locales se ha manifestado a lo largo de toda la implementación de los tres algoritmos paralelos, con lo que deben buscarse alternativas que permitan generar soluciones para salir de dichos mínimos y evolucionar favorablemente.

Acerca de la comunicación entre procesos, debe mejorarse la sincronización entre procesos, ya que cuando los tiempos de exploración en distintos procesos son muy diferentes se producen esperas y llegan a perderse parte de los mensajes enviados, al ser sobrescritos por mensajes de otros procesos.

Otras mejoras del algoritmo actual implican un cambio en su estructura. Algunas de ellas son:

- Insistir en la división del espacio de búsqueda, de forma que cada proceso pudiera explorar sólo una zona del espacio, tal como se propone en Solar *et al.* (2002). De todas formas no debe perderse de vista que esta propuesta es compleja para los problema de secuenciación.
- Comparar o emplear directamente otras heurísticas para su implementación. Algunas como los algoritmos genéticos parecen más prometedoras por la mayor facilidad de dividir el espacio de soluciones y por la aceleración que

puede conseguirse en paralelo. La búsqueda tabú ha sido estudiada por muchos autores para su implementación en paralelo y, en algún caso, puede mejorar el aprovechamiento de la capacidad de cálculo disponible.

- Por último, cambiar la topología actual de la red para emplear topologías más complejas de comunicación, como mallas o toros quedan fuera del objeto de este trabajo, aunque un buen punto de partida para estas mejoras puede ser el trabajo de Vigo y Maniezzo (1997).

8 Referencias

Abramson, D.A.; Abela, J. (1992). A Parallel Genetic Algorithm for Solving the School Timetabling Problem. 15 Australian Computer Science Conference, Hobart.

Abramson, D.A. (1992). A Very High Speed Architecture to Support Simulated Annealing. IEEE Computer. Vol. 25, no. 5, pp. 27–34.

Acacio, M.; Cánovas, O.; García, J.M.; López-de-Teruel, P.E.; (2002). MPI–Delphi: an MPI implementation for visual programming environments and heterogeneous computing. Future Generation Computer Systems. Vol. 18, pp. 317–333.

Adenso-Díaz, B.; García-Carvajal, S.; Lozano, S.; (2006). An empirical investigation on parallelization strategies for Scatter Search. European Journal of Operational Research. Vol. 169, pp. 490–507.

Adenso-Díaz, B.; Laguna, M. (2006) Fine Tuning of Algorithms Using Fractional Experimental Designs and Local search. Operations Research. Vol. 54, no. 1, pp. 99–114.

Aggarwal, A.; Chandra, A.K.; Snir, M. (1989). On communication latency in PRAM computations. Proceedings of the first annual ACM symposium on Parallel algorithms and architectures, pp. 11–21.

Aiex, R. M.; Resende, M. G. C.; Ribeiro, C. C. (2002). Probability distribution of solution time in GRASP: An experimental investigation. Journal of Heuristics, Vol. 8, pp. 343–373.

Aiex, R. M.; Resende, M. G. C.; Pardalos, P.; Toraldo, G. (2005). GRASP with Path Relinking for Three-Index Assignment. INFORMS Journal on Computing. Vol. 17, no. 2, pp. 224–247.

AIMS (2007). Automated Instrumentation and Monitoring System site. NASA. http://www.nas.nasa.gov/Software/AIMS/

Alba, E.; Almeida,F.; Blesa,M.; Cotta, C.; Díaz,M.; Dorta,I.; Gabarró,J.; León,C.; Luque,G.; Petit,J.; Rodríguez,C.; Rojas,A.; Xhafa,F. (2006). Efficient parallel LAN/WAN algorithms for optimization. The mallba project. Parallel Computing Vol. 32, no. 5–6, pp. 415–440.

Alba, E. (editor) (2005) Parallel Metaheuristics: A New Class of Algorithms. Wiley-Interscience.

Alba, E.; Troya, J.M. (1990). A Useful Review on Coarse Grain Parallel Genetic Algorithms. Artificial Intelligence. Vol. 2, pp. 189–214.

Alba, E.; Dorronsoro, B. (2004). Solving the vehicle routing problem by using cellular genetic algorithms. European Conference on Evolutionary Computation in Combinatorial Optimisation. LNCS Vol. 3004/2004, pp. 11-20.

Amd (2007). http://www.amd.com.

Amdahl, G.M. (1967). Validity of the single-processor approach to achieving large scale computing capabilities. AFIPS Conference Proceedings. Vol. 30, pp. 483–485. (Atlantic City, N.J., Apr. 18–20). AFIPS Press, Reston.

Anstreicher, K. M. (2003). Recent advances in the solution of quadratic assignment problems. Mathematical Programming Vol. 97, no. 1, pp. 27–42.

Ashour, S. (1970). A branch-and-bound algorithm for the flow shop problem scheduling problem. AIIE Transactions Vol. 2, pp. 172–176.

Attanasio, A; Cordeau, J.F.; Ghiani, G.; Laporte, G. (2004). Parallel Tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. Parallel Computing. Vol. 30, pp. 377–387.

Bachelet, V; Talbi, E. (2000). A parallel co-evolutionary metaheuristic. Parallel and Distributed Processing. LNCS. pp. 628–635.

Bachelet,V.; Preux,P.; Talbi, E.-G. (1996). Parallel hybrid meta-heuristics: Application to the quadratic assignment problem, pp. 233-242.

Bala, V.; Bruck, J.; Bryant, R.; Cypher, R.; de Jong, P.; Elustondon, P.; Frye, D.; Ho, A.; Ho, C.T.; Irwin, G.; Kipnis, S.; Lawrence, R.; Snir, M. (1994). The IBM external user interface for scalable parallel systems. Parallel Computing. Vol. 20, no. 4, pp. 445-462.

Barcelona Supercomputing Center (2007). http://www.bsc.es.

Barr, R.S.; Hickman, B.L. (1993). Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions. ORSA Journal on Computing. Vol. 5, no. 1, pp. 2–18.

Barr, R.S.; Golden, B.L.; Kelly, J.P.; Resende, M.J.C.; Stewart, W.R. (1995) Designing and Reporting on Computational Experiments with Heuristic Methods. Journal of Heuristics. Vol. 1, pp. 9–32.

Barron, I.M.; Cavill, P.; May, D.; Wilson, P. (1983). Transputer does 5 or more MIPS even when not used in parallel. Electronics. Vol. 56, no. 23, pp. 109–115.

Barton, E.; Cownie, J.; McLaren, M. (1994). Message passing on the Meiko CS-2. Parallel Computing. Vol. 20, no. 4, pp. 497-507.

Basseur, M.; Lemesre, J.; Dhaenens, C.; Talbi, E.-G. (2004). Cooperation between branch and bound and evolutionary approaches to solve a bi-objective flow shop problem. Editado por Springer Berlin / Heidelberg. Experimental and Efficient Algorithms. Third International Workshop. Angra dos Reis, Brazil: Springer Berlin / Heidelberg, pp.72-86.

Battiti, R.; Tecchiolli, G. (1992). Parallel Biased Search for Combinatorial Optimization: Genetic Algorithms and TABU. Microprocessors and Microsystems.Vol. 16, no. 7, pp. 351–367.

Belkadi, K.; Gourgand, M.; Benyettou, M. (2006a). Resolution of Scheduling Problem of the Production Systems by Sequential and Parallel Tabu Search. Journal of Applied Sciences. Vol. 6, no. 7, pp. 1534–1539.

Belkadi, K.; Gourgand, M.; Benyettou, M. (2006b). Parallel Genetic Algorithms With Migration for the Hybrid Flow Shop Scheduling Problem. Journal of Applied Mathematics and Decision Sciences. Vol. 2006. Article ID 65746. pp. 1–17.

Belkadi, K.; Gourgand, M.; Benyettou, M.; Aribi, A. (2006c). Sequential and Parallel Genetic Algorithms with Migration for the Hybrid Flow Shop Scheduling Problem. Journal of Applied Sciences. Vol. 6, no. 4, pp. 775–778.

Bendjoudi, A.; Guerdah, S.; Mansoura, M.; Melab, N.; Talbi, E.G. (2008). P2P B&B and GA for the Flow-Shop Scheduling Problem. Metaheuristics for Scheduling in Distributed Computing Environments. pp. 301–321. Springer.

Bendjoudi, A.; Melab, N.; Talbi, E.G. (2007). A parallel P2P Branch-and-Bound Algorithm for Computational Grids. Seventh IEEE International Symposium on Cluster Computing and the Grid. Rio de Janeiro, Brazil: IEEE. pp. 749–754.

Benett, F.H.; Koza, J.R.; Shipman, J.; Stiffelman, O. (1999) Building a Parallel Computer System for $18,000 that Performs a Half Peta-Flop a Day. GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, July 13–17, 1999, Orlando, Florida USA. San Francisco, CA: Morgan Kaufmann, pp. 1484–1490.

The Beowulf cluster site. (2007). http://www.beowulf.org

Bianchini, R.; Brown, C.M. (1993). Parallel genetic algorithms on distributed-memory architectures, Transputer Research and Applications. Vol. 6, pp. 67–82.

Blanco, V.; Cabaleiro, J.M.; Pena, T.F.; Doallo, R.; Sanjurjo, J.R.; Argüello, F.; Plata, O.G.; Rivera, E.F.; Zapata, E.L.; (1993). Evaluación gráfica de programas paralelos en *Transputers* mediante ParaGraph. Technical Report No. UMA-DAC-93/08. IV Jornadas de Paralelismo. Santander, España, 27 al 29 de septiembre de 1993.

Blesa, M.J.; Hernández, Ll.; Xhafa, F. (2001). Parallel skeletons for tabu search method. Eighth International Conference on Parallel and Distributed Systems (ICPADS'01). pp.23–28.

Blum, C.; Roli, A.; Alba, E. (2005). An introduction to metaheuristic techniques. En Parallel Metaheuristics, a New Class of Algorithms, de E. Alba, editado por E. Alba, 3–42. John Wiley.

Le Bouthillier, A.; Crainic, T.G. (2005). A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. Computers & Operations Research. Vol. 32, pp. 1685–1708.

Bolosky, W.J.; Scott, M.L.; Fitzgerald, R.P.; Fowler, R.J.; Cox, A.J. (1991). NUMA policies and their relation to memory architecture. Proceedings of the fourth international conference on Architectural support for programming languages and operating systems, pp. 212–221.

Bożejko, W.; Wodecki, M. (2008a). Parallel scatter search algorithm for the flow shop sequencing problem. PPAM 2007. 7th International Conference on Parallel Processing and Applied Mathematics. Wyrzykowski, R.; Szymanski et B. Ed., LNCS, Springer. To appear.

Bożejko, W.; Wodecki, M. (2008b). Parallel path-relinking method for the flow shop scheduling problem. International Conference on Computational Science (ICCS 2008). LNCS. Springer. To appear.

Bożejko, W.; Wodecki, M. (2004a). The New Concepts in Parallel Simulated Annealing Method. L. Rutkowski *et al.,* Eds.): ICAISC 2004, LNAI 3070, pp. 853–859.

Bożejko, W.; Wodecki, M. (2004b). Parallel tabu search method approach for very difficult permutation scheduling problems. Proceedings of International Conference on Parallel Computing in Electrical Engineering: PARELEC '04, pp. 156–161.

Bożejko, W.; Wodecki, M. (2002). Solving the flow shop problem by parallel tabu search. Proceedings of International Conference on Parallel Computing in Electrical Engineering, 2002. PARELEC '02, pp. 189–194.

Brawer, S. (1986). Introduction to Parallel Programming. Academic Press, Inc. ISBN 0-12-128470-0.

Bremmer, D. ZRAM. 2008. http://www.cs.unb.ca/~bremner/software/zram/ (2008).

Brown, A.P.G.; Lomnicki, Z.A. (1966). Some applications of the branch-and-bound algorithm to the machine sequencing problem. Operational Research Quarterly. Vol. 17, no. 2, pp. 173–186.

Brüngger, A.; Marzetta, A.; Fukuda, K.; Nievergelt, J. (1999). The parallel search bench ZRAM and its applications. Annals of Operations Research. Vol. 90, no. 0, pp. 45–63.

Brüngger, A.; Marzetta, A.; Clausen, J.; Perregaard, M. (1998). Solving Large-Scale QAP Problems in Parallel with the Search Library ZRAM. Journal of Parallel and Distributed Computing, Vol. 50, no. 1–2, pp. 157–169.

Brüngger, A.; Marzetta, A.; Clausen, J.; Perregaard, M. (1997). Joining forces in solving large-scale quadratic assignment problems. 11th International Parallel Processing Symposium, 1997. Geneva, Switzerland: IEEE Computer Society Press, pp. 418–427.

Butler R.M.; Lusk, E.L. (1994). Monitors, messages, and clusters: The p4 parallel programming system. Parallel Computing. Vol. 20, no. 4, pp. 547–564.

Buyya, R. (1999). High Performance Cluster Computing. Prentice-Hall.

Cahon, S.; Melab, N.; Talbi E. G.; (2004) Building with ParadisEO reusable parallel and distributed evolutionary algorithms. Parallel Computing, 30, pp. 677–697.

Calegari, P.; Guidec, F.; Kuonen, P. Kobler, D. (1997). Parallel Island-Based Genetic Algorithm for Radio Network Design. Journal of Parallel and Distributed Computing, vol. 47, pp. 86–90.

Calkin, R.; Hempel R.; Hoppe H.C.; Wypior, P. (1994). Portable programming with the PARMACS message-passing library. Parallel Computing. Vol. 20, no. 4, pp. 615–632.

Calzarossa, M.; Massari, L.; Tessera, D. (2004). A methodology towards automatic performance analysis of parallel applications. Parallel Computing. Vol. 30. 211–223.

Campbell, H.G.; Dudek, R.A.; Smith M.L.. (1970). A Heuristic Algorithm for the n Job, m Machine Sequencing Problem. Management Science. Vol. 16, no. 10, pp. 630–637.

Camponogara, E.; Talukdar, S.N. (2004). Designing communication networks for distributed control agents. European Journal of Operational Research. Vol. 153, no. 3, pp. 544–563.

Cantú-Paz, E. (1997). A Survey of Parallel Genetic Algorithms. IlliGAL Report No. 97003. Illinois Genetic Algorithms Laboratory. University of Illinois at Urbana Champaign.

Cantú-Paz, E. (2001). Migration Policies, Selection Pressure, and Parallel Evolutionary Algorithms. Journal of Heuristics, 7, pp. 311–334.

Cantú-Paz, E. (2007). Parameter Setting in Parallel Genetic Algorithms, Studies in Computational Intelligence (SCI). Vol. 54, pp. 259–276.

Caricato, P.; Ghiani, G; Grieco, A.; Guerriero, E. (2003). Parallel tabu search for a pickup and delivery problem under track contention. Parallel Computing. Vol. 29, pp. 631–639.

Carlier, J.; Rebaï, M.I. (1996). Two branch-and-bound algorithms for the permutation Flow shop problem. European Journal of Operational Research. Vol. 90, no. 2, pp. 238–251.

Carriero, N.J.; Gelernter, D.; Mattson, T.G.; Sherman, A.H. (1994). The Linda alternative to message-passing systems. Parallel Computing. Vol. 20, no. 4, pp. 633–655.

Carriero, N.; Gelernter, D. (1989). Linda in Context. Communications of the ACM. Vol. 32, no. 4, pp. 444–458.

Chakrapani, J.; Skorin-Kapov, J. (1992). A connectionist approach to the quadratic assignment problem. Computers and Operations Research. Vol. 19, no. 3–4: 287–295.

Chang, P.; Dolan, j:; Hemmerle, J.; Terk, M.; Talukdar, S.N. (1997). Asynchronous teams (A-Teams) and the A-Teams toolkit: an agent-based problem-solving architecture and software framework. Proceedings of the 1997 Artificial Neural Networks in Engineering Conference, ANNIE'97. St.Louis, MO, USA: ASME, pp. 73–78.

Chapin, J.; Herrod, A.; Rosenblum, M.; Gupta, A. (1995). Memory system performance of UNIX on CC-NUMA multiprocessors. Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems. Ottawa, Ontario, Canada. pp. 1 – 13.

Cheng, J.; Kise, H.; Matsumoto, H. (1997). A branch-and-bound algorithm with fuzzy inference for a permutation Flow shop scheduling problem. European Journal of Operational Research. Vol. 96, no. 3, pp. 578–590.

Clausen, J; Perregaard, M.. (1997). Solving Large Quadratic Assignment Problems in Parallel. Computational Optimization and Applications (Kluwer Academic Publishers) Vol. 8, no. 2 (septiembre 1997), pp. 111–127.

Clausen, J.; Karisch, S.E.; Perregaard, M.; Rendl, F. (1998). On the Applicability of Lower Bounds for Solving Rectilinear Quadratic Assignment Problems in Parallel. Computational Optimization and Applications (Springer Netherlands). Vol. 10, no. 2 (5 1998), pp. 127–147.

Clausen, J.; Larsson Träff, J. (1991). Implementation of parallel branch-and-bound algorithms – experiences with the graph partitioning problem. Annals of Operations Research (Springer-Netherlands). Vol. 33, no. 5 (mayo 1991), pp. 329–349.

Comer, D.E.; Stevens, D.L. (1993). Internetworking with TCP/IP. Vol. 3. Prentice Hall.

Companys R. (1999) Note on an improved branch-and-bound algorithm to solve n/m/P/Fmax problems. TOP. Vol. 7, no. 11, pp. 25–31.

Companys, R.; Mateo, M. (2007). Different behaviour of a double branch-and-bound algorithm on $F_m/prmu/C_{max}$ and $F_m/block/C_{max}$ problems. Computers & Operations Research. Vol. 34, pp. 938–953.

Conway, R.W.; Maxwell, W.L.; Miller, L.W. (1967). Theory of Scheduling. Addison Wesley, Reading, MA.

Cordeau, J.F.; Laporte, G. (2004). Tabu Search Heuristics for the Vehicle Routing Problem. Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search, C. Rego and B. Alidaee, eds., Kluwer, Boston, 145–163, 2004.

Corrêa, R.C.; Gomes, F.; Oliveira, C.A.S.; Pardalos, P.M. (2003). A parallel implementation of an asynchronous team to the point-to-point connection problem. Parallel Computing (Elsevier). Vol. 29, no. 4, pp. 447–466.

Cotta, C.; Talbi, E.-G.; Alba, E. (2005). Parallel Hybrid Metaheuristics. Cap. 15 de Parallel Metaheuristics, de E. Alba, editado por E. Alba, 347–370. John Wiley & Sons Inc..

Cotta, C.; Aldana, J.F.; Nebro, A.J.; Troya, J.M. (1995). Hybridizing genetic algorithms with branch and bound techniques for the resolution of the TSP. Editado por D.W. Pearson, N.C. Steele y R.F. Albrecht. International Conference on Artificial Neural Networks and Genetic Algorithms. Alès, France: Springer-Verlag, pp. 277–280.

Cotta, C.; Troya, J.M. (2003). Embedding Branch and Bound within Evolutionary Algorithms. Applied Intelligence. Vol. 1, pp. 137–153.

Crainic, T.G.; Le Cun, B.; Roucairol, C. (2006). Parallel Branch and Bound Algorithms. Parallel Combinatorial Optimization. El-Ghazali Talbi, Ed. John Wiley & Sons.

Crainic, T.G.; Toulouse, M.; Gendreau, M. (1997). Towards a taxonomy of parallel Tabu search algorithms. INFORMS Journal on Computing. Vol. 9, pp. 61–72.

Crainic, T.G.; Gendreau, M.; Farvolden, J.M. (2002). A Simplex-Based Tabu Search Method for Capacitated Network Design. INFORMS Journal on Computing. Vol. 12, no. 3, pp. 223–236.

Crainic, T.G.; Gendreau, M. (2002). Cooperative Parallel Tabu Search for Capacitated Network Design. Journal of Heuristics. Vol. 8, pp. 601–627. Kluwer.

Crainic, T.G.; Toulouse, M. (2002). Parallel Strategies for Meta-heuristics. State-of-the-Art Handbook in Metaheuristics, F. Glover, G. Kochenberger (Eds.), Kluwer Academic Publishers.

Crainic, T.G.; Toulouse, M. (1998). Parallel Metaheuristics. Fleet Management and Logistics, T.G. Crainic and G. Laporte (Eds.), pp. 205–251. Kluwer.

Crichlow, J.M. (1988). An introduction to distributed and parallel computing. Prentice Hall.

Culler, D.; Karp, R.; Patterson, D.; Sahay, A.; Schauser, K.E.; Santos, E.; Subramonian, R.; von Eicken, T. (1993). LogP: towards a realistic model of parallel computation.

Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 1–12.

Cung, V.D.; Martins, S.; Ribeiro, C.; Roucairol, C. (2001) Strategies for the parallel implementation of Metaheuristics. Essays and Surveys in Metaheuristics. C.C. Ribeiro and P. Hansen, editors), Kluwer Academic Publishers.

Darby-Dowman, K.; Little, J. (1998). Properties of Some Combinatorial Optimization Problems and Their Effect on the Performance of Integer Programming and Constraint Logic Programming. INFORMS Journal on Computing. Vol. 10, no. 3, pp. 276–286

De Rose, L.; Zhang, Y.; Reed, D.A. (1998). SvPablo: A Multi-Language Performance Analysis System. Lecture Notes in Computer Science. Vol. 1469/1998. P. 352.

Den Besten, M.; Stützle, T. (2001). Neighborhoods Revisited: An Experimental Investigation into the Effectiveness of Variable Neighborhood Descent for Scheduling. MIC'2001- 4th Metaheuristics International Conference, pp. 545–549.

Denzinger, J.; Offerman, T. (1999). On cooperation between evolutionary algorithms and other search paradigms. Proceedings of Congress on Evolutionary Computation CEC1999. Washington, DC, USA: IEEE, 1999, pp. 2317–2324.

Digalakis, J.; Margaritis, K. (2004) Performance comparison of memetic algorithms. Applied Mathematics and Computation 158 pp. 237–252.

Dongarra, J.; Foster, I.; Fox, G.; Gropp, W.; Kennedy, K.; Torczon, L.; White, A. (2003). Sourcebook of Parallel Computing. Morgan Kaufmann Publishers.

Dorigo, M; Di Caro, G. (1999). Ant Algorithms for Discrete Optimization. Artificial Life. Vol. 5, no. 2, pp. 137–172. MIT Press.

Dorigo, M.; Maniezzo, V.; Colorni, A. (1996). The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B, Vol. 26, no. 1. pp.1–13.

Dumitrescu, I.; Stutzle, T. (2003). Combinations of local search and exact algorithms. Applications of Evolutionary Computing (SPRINGER-VERLAG). Vol. 2611, pp. 211–223.

Eicken, T.; Culler, D.; Goldstein, S.; Schauser, K. (1992). Active messages: A mechanism for integrated communication and computation. Proceedings of the Nineteenth International Symposium on Computer Architecture. ACM Press.

El-Abd, M.; Kamel, M. (2005). A Taxonomy of Cooperative Search Algorithms. Editado por M.J. Blesa *et al*. Hybrid Metaheuristics (Springer-Verlag Berlin Heidelberg), pp. 32–41.

Esquivel, S.; Leguizamón, G.; Zuppa, F.; Gallard, R. (2002). A Performance Comparison of Alternative Heuristics for the Flow Shop Scheduling Problem. Vol. LNCS 2279, de EvoWorkshops 2002, editado por S. Cagnoni *et al*., 51–60. Berlin Heidelberg: Springer-Verlag.

Feo, T.A.; Resende, M.G.C.; Smith, S.H. (1994). A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. Operations Research. Vol. 42, no. 5, pp. 860–878.

Feschet, F.; Miguet, S.; Perroton, L. (1998). ParList: A Parallel Data Structure for Dynamic Load Balancing. Journal of parallel and distributed computing. Vol. 51, pp. 114–135.

Fiechter, C.N. (1994). A parallel tabu search algorithm for large traveling salesman problems. Discrete Applied Mathematics. Vol. 51, pp. 243–267.

Flower, J.; Kolawa, A. (1994). Express is not just a message passing system current and future directions in Express. Parallel Computing. Vol. 20, no. 4, pp. 597–614.

Flynn, M.J. (1966). Very High-speed Computing Systems. Proceedings of the IEEE. Vol. 54, no. 12, pp. 1901–1909.

Fortune, S.; Willie, J. (1978). Parallelism in random access machines. Proceedings of the tenth annual ACM symposium on Theory of computing, pp. 114 – 118.

Foster, C.C.; Riseman, E.M. (1972) Percolation of Code to Enhance Parallel Dispatching and Execution. IEEE Transactions on Computers. Vol. 21, no. 12, pp. 1411–1415.

Fraigniaud, P.; Vial, S. (1999). Comparison of heuristics for one-to-one and all-to-all communications in partial meshes. Parallel Processing Letters. Vol. 9. n 1, pp. 9–20.

Framiñán, J.M.; Pastor, R. (2008). A proposal of a hybrid meta-strategy for Combinatorial Optimization Problems. Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics. Nottingham, UK. EU/ME, the European Chapter on Metaheuristics (sponsored by EURO, the European Association of OR Societies).

Framiñán, J.M.; Schuster, C. (2006) An enhanced timetabling procedure for the no-wait job-shop problem: A complete local search approach. Computers & operations research. Vol. 331, pp. 1200–1213.

Framiñán, J.M.; Gupta, J.N.D.; Leisten, R. (2004) A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. Journal of the Operational Research Society. Vol. 55, pp. 1243–1255.

Framiñán, J.M.; Leisten, R.; 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. Vol. 41, no. 1. pp. 121–148.

Framiñán, J.M.; Leisten, R.; Ruiz-Usano, R. (2002) Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. European Journal of Operational Research. Vol. 141, pp. 559–569.

França, P. M.; Tin Jr, G.; Buriol, L. S. (2006). Genetic algorithms for the no-wait flowshop sequencing problem with time restrictions. International Journal of Production Research. Vol. 44, no. 5, pp. 939 – 957.

Galil (1986). Optimal parallel algorithms for string matching. Information and Control. Vol. 67, no. 1–3, pp. 144–157.

García López, F.; Melián-Batista, B.; Moreno-Pérez, J.A.; Moreno Vega, J.M. (2002). The Parallel Variable Neighborhood Search for the p-Median Problem. Journal of Heuristics. Vol. 8, no. 3, pp. 375–388.

Garey, M.; Johnson, D.; Sethi, R. (1976). The Complexity of flowshop and Jobshop Scheduling. Mathematics of Operations Research. Vol. 1, pp. 117–129.

Geist, A.; Beguelin, A.; Dongarra, J.; Jiang, W.; Manchek, R.; Sunderam, V. (1994). PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press.

Gendreau, M.; Laporte, G.; Semet, F.; (2001). A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. Parallel Computing Vol. 17, no.12, pp. 1641–1653.

Ghosh, D. (2003). Neighborhood search heuristics for the uncapacitated facility location problem. European Journal of Operational Research Vol. 150, pp. 150–162.

Ghosh, D.; Sierksma, G.; (2002). Complete Local Search with Memory. Journal of Heuristics. Vol. 8, no. 6, pp. 571–584. Kluwer.

Gibbons, A.; Rytter, W. (1988). Efficient Parallel Algorithms. Cambridge University Press.

Gillet, B.E. (1976). Introduction to Operational Research. A Computer-Oriented Algorithmic Approach. McGraw-Hill.

Glover, F. (1998). A template for scatter search and path relinking. In: Hao, J.-K., *et al*., Eds.), Artificial Evolution, LNCS 1363. Springer, London, pp. 13–54.

Glover, F. (1989). Tabu Search – Part 1. ORSA Journal on Computing. Vol. 1, no. 3, pp. 190–206.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. Decision Sciences. Vol. 8, no. 1, pp. 156–166.

Gondzio, J.; Sarkissian, R.;Vial, J.-Ph. (2001). Parallel Implementation of a Central Decomposition Method for Solving Large-Scale Planning Problems. Computational Optimization and Applications. Vol. 19, 1, p. 5.

Gorti, S.R.; Humair, S.; Sriram, R.D.; Talukdar, S.N.; Murthy, S. (1996). Solving constraint satisfaction problems using ATeams. Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM. Vol. 10, no. 1, pp. 1–19.

Grabowski J. (1980). On two-machine scheduling with release dates to minimize maximum lateness. Opsearch. Vol. 17, no. 2, pp. 133–154.

Graham, R.; Lawler, E.; Lenstra, J.; Kan, A.R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics. Vol. 5, pp. 287–326.

Gropp, W.D.; Lusk, E. (1992). A test implementation of the MPI draft message-passing standard. Technical Report ANL-92/47, Argonne National Laboratory.

Gropp W.D.; Smith, B. (1993). Chameleon Parallel Programming Tools users manual. Technical Report ANL-93/30, Argonne National Laboratory, Argonne, IL.

Gropp, W.D.; Lusk, E.; Skjellum, A. (1999) Using MPI, Portable Parallel Programming with the Message-Passing Interface. 2nd ed. The MIT Press.

Grosch, H. A. (1953). High speed arithmetic: the digital computer as a research tool. , Journal of the Optical Society of America, no. 43, Vol. 4. pp.306–311.

Gunst, R.F.; Mason, R.L. (1991). How to construct fractional factorial experiments. (The ASQ Basic References in Quality Control: Statistical Techniques. Vol. 14) Shapiro, S.S. and Mykytka, Coeditors. American Society for Quality. Statistics Division.

Gupta, J.N.D.; Sexton, R.S.; Tunc, E.A. (2000). Selecting Scheduling Heuristics Using Neural Networks. INFORMS Journal on Computing. Vol. 12, no. 2, pp. 150–162.

Gupta, J.N.D. (1971). A Functional Heuristic Algorithm for the Flowshop Scheduling Problem. Operational Research Quarterly (Palgrave Macmillan Journals). Vol. 22, no. 1, pp. 39–47.

Hafidi, Z.; Talbi, E.G.; Geib, J.-M. (1996). MARS: Adaptive scheduling of parallel applications in a multi-user heterogeneous environment. European School of Computer Science ESPPE'96: Parallel Programming Environments for High Performance Computing, Alpe d'Huez, France, pp. 119–122.

Hagersten, E.; Landin, A.; Haridi, S. (1992). DDM-a cache-only memory architecture. IEEE Computer. Vol. 25, no. 9, pp. 44–54.

Hansen, P.; Mladenovic, N.; Moreno Pérez, J.A. (2003). Variable Neighborhood Search. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. Vol. 19, pp. 77–92.

Hansen, P.; Mladenovic, N. (2001). Variable neighborhood search: Principles and applications. European Journal of Operational Research. Vol. 130, no. 3, pp. 449–467.

Haouari, M.; Ladhari, T. (2003). A branch-and-bound-based local search method for the flow shop problem. Journal of the Operational Research Society. Vol. 54, no. 10, pp. 1076–1084.

Haq, A.; Saravanan, M.; Vivekraj, A.; Prasad, T. (2007). A scatter search approach for general flowshop scheduling problem. The International Journal of Advanced Manufacturing Technology. Vol. 31, no. 7, pp. 731–736.

Hascoët, L. (2001) A method for automatic placement of communications in SPMD parallelization. Parallel Computing. Vol. 27, pp. 1655–1664.

Heath, M.T.; Finger, J.E. (2003). Paragraph: A performance visualization tool for MPI. http://www.csar.uiuc.edu/software/paragraph.

Hempel, R. (1991) The ANL/GMD macros (PARMACS) in FORTRAN for portable parallel programming using the message passing programming model, user's guide and reference manual. Technical Memorandum, Gesellschaft fiir Mathematik und Datenverabeitung mbH, West Germany.

Hempel, R.; Hey, A.J.G.; McBryan, O.; Walker, D.W. (1994). Message passing interfaces. Parallel Computing. Vol. 20, no. 4, pp. 415–416.

High Performance Fortran Forum (2007). Rice University. http://hpff.rice.edu/.

Holland, J.H. (1992). Genetic Algorithms. Scientific American. Vol. 267, no. 1, pp. 66–73.

Holland, J. (1975) Adaptation in Natural and Artificial Systems: An Introductory Analysis. University of Michigan Press, Ann Arbor, MI.

Hughes, C; Hughes, T. (2003). Parallel and Distributed Programming Using C++. Addison-Wesley. ISBN 0-13-101376-9.

Huntley, C.L.; Brown, D.E. (1996). Parallel Genetic Algorithms with Local Search. Computers & Operations Research (Elsevier). Vol. 23, no. 6, pp. 559–571.

Huntley, C.L.; Brown, D.E. (1991). A parallel heuristic for quadratic assignment problems. Computers & Operations Research, (Elsevier). Vol. 18, no. 3, pp. 275–289.

IBM (2007). http://www.ibm.com.

Ignall, E.; Schrage, L.E. (1965). Application of the branch-and-bound technique to some flow shop problems. Operations Research. Vol. 13, no. 3, pp. 400–412.

Ikegami, A.; Aoyagi, K.; Iizuka, K. (1993). A parallel branch-and-bound algorithm for integer linear programming and its implementation on a distributed multiprocessor. Systems and Computers in Japan (Wiley Periodicals, Inc.). Vol. 24, no. 3, pp. 35–47.

INA: Institut d'Informatique Appliquée. (2007) http://ina2.eivd.ch

Intel (2007). http://www.intel.com.

James, T.; Rego, C.; Glover, F. (2005). Sequential and Parallel Path-Relinking Algorithms for the Quadratic Assignment Problem, IEEE Intelligent Systems, 20(4)58–65, 2005.

Jelasity, M.; Martínez Ortigosa, P.; García, I. (2001). UEGO, an Abstract Clustering Technique for Multimodal Global Optimization. Journal of Heuristics. Vol. 7, pp. 215–233.

Johnson, S.M. (1954). Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly. Vol. 1, pp. 61–68.

Jourdan, L.; Basseur, M.; Talbi, E.-G.. (2008). Hybridizing exact methods and metaheuristics: A taxonomy. European Journal of Operational Research (In Press, Corrected Proof)

Juurlink, B.H.H.; Wijshoff, H.A.G. (1995). The E-BSP Model: Incorporating Unbalanced Communication and General Locality into the BSP Model. Technical Report 95–44, Leiden University.

Juurlink, B.H.H.; Wijshoff, H.A.G. (1996). A quantitative comparison of parallel computation models. ACM Symposium on Parallel Algorithms and Architectures, pp. 13–25.

Kalczynski, P.J.; Kamburowski, J. (2007). On the NEH heuristic for minimizing the makespan in permutation flow shops. Omega. Vol. 35, no. 1, pp. 53–60.

Karp, A.H. (1987). Programming for Parallelism. IEEE Computer. Vol. 20, no. 5, pp. 43–57.

Kebbal, D.; Talbi, E.-G.; Geib, J.-M. (2001). Scheduling parallel adaptive applications in networks of workstations and clusters of processors. Proceedings of 2001 IEEE International Conference on Cluster Computing, pp. 116–123.

Kennedy, J.; Eberhart, R. (1995). Particle swarm optimization. Proceedings of IEEE international conference on neural network. Vol.4, pp. 1942–1948.

Kennedy, K; Koelbel, C.; Cima, H. (2007). The rise and fall of High Performance Fortran: an historical object lesson. History of Programming Languages. Proceedings of the third ACM SIGPLAN conference on History of programming languages, pp. 7–1 – 7–22.

Kindervater, G.A.P.; Lenstra, J.K. (1988). Parallel computing in combinatorial optimizacion. Annals of Operations Research. Vol. 14, no. 1, pp. 245–289.

Kirley, M. (2002) A Cellular Genetic Algorithm with Disturbances: Optimisation Using Dynamic Spatial Interactions. Journal of Heuristics, vol. 8, pp. 321–342.

Kirpatrick, S.; Gelatt, C.D.; Vecchi, M.P. (1983). Optimization by Simulated Annealing. Science, New Series, Vol. 20, no. 4598, pp. 671–680.

Kliewer, G. (2000a). A General Software Library for Parallel Simulated Annealing. Proceedings of EURO Winter Institute on Metaheuristics in Combinatorial Optimisation, Lac Noir, Switzerland. Elsevier.

Kliewer, G. (2000b). A General Software Library for Parallel Simulated Annealing and its Application in Airline Industry. Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000), pp. 55–61.

Kohlmorgen, U.; Schmeck, H.; Haase, K. (1999). Experiences with fine-grained parallel genetic algorithms. Annals of Operations Research. Vol. 90, pp. 203–219.

Kuhn, R.H.; Padua, D.A. (1981) Tutorial on Parallel Processing. IEEE Computer Society Press.

Kuo, I-H.; Horng, S.J.; Kao, T.W.; Lin, T.L.; Lee, C.L.; Terano, T.; Pan, Y. (2009). An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model. Expert Systems with Applications. Vol. 36, no. 3–2, pp. 7027–7032.

Ladhari T.; Haouari, M. (2005). A computational study of the permutation flow shop problem based on a tight lower bound. Computers & Operations Research. Vol. 32, pp. 1831–1847.

Lageweg, B.J.; Lenstra, J.K.; Rinnooy Kan, A.H.G. (1978). A general bounding scheme for the permutation Flow-shop problem. Operations Research. Vol. 26, no. 1, pp. 53–67.

Laguna, M.; Martí, R. (2003). Scatter Search. Methodology and Implementations in C. Kluwer, Boston.

Land, A.H.; Doig, A.G. (1969). An automatic method for solving discrete programming problems. Econometrica. Vol. 28, no. 3, pp. 427–520.

Laursen, P.S. (1993). Simple approaches to parallel Branch and Bound. Parallel Computing (Elsevier). Vol. 19, no. 2, pp. 143–152.

Laursen, P. (1994). Problem independent parallel simulated annealing using selection and migration. Parallel Problem Solving from Nature III, LNCS. pp. 408–417.

Leibniz Rechenzentrum (2007). http://www.lrz-muenchen.de

León Blanco, J.M.; Framiñán Torres, J.M.; González Rodríguez, P.L.; Pérez González, P.; Ruiz Usano, R. (2006). Influencia de la Granularidad en las Prestaciones de Algoritmos Paralelos. X Congreso de Ingeniería de Organización. Valencia, pp. 217-218.

Lian, Z.; Gu, X.; Jiao, B. (2008). A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. Chaos, Solitons & Fractals (Elsevier). Vol. 35, no. 5, pp. 851–861.

Lilja, D.J. (1994). A multiprocessor architecture combining fine-grained and coarse-grained parallelism strategies. Parallel Computing. Vol. 20, pp. 729–751.

Lin, S.H.; Goodman, E.D.; Punch, W.F. (1997). Investigating Parallel Genetic Algorithms on Job Shop Scheduling Problems. 6th International Conference on Evolutionary Programming, pp. 383–393.

Liu, C.S.; Tseng, C.H. (2000). Parallel Synchronous and Asynchronous Space-Decomposition Algorithms for Large-Scale Minimization Problems. Computational Optimization and Application. Vol. 17, 1, p. 85.

Liu, J.; Reeves, C.R. (2001). Constructive and composite heuristic solutions to the P||∑Ci scheduling problem. Eurpoean Journal of Operational Research. Vol. 132, pp. 439–452.

Liu, J.; Wu, J.; Panda, D. K. (2004). High Performance RDMA-Based MPI Implementation over InfiniBand. International Journal of Parallel Programming, 2004. In Press.

Liu, Z. (1998). Worst case analysis of scheduling heuristics of parallel systems. Parallel Computing. Vol. 24, pp. 863–891.

Local Area Multicomputer (2007). http://www.lam-mpi.org.

Loiola, E.M., Maia de Abreu, N.M.; Boaventura-Netto, P.O.; Hahn, P.; Querido, T. (2007). A survey for the quadratic assignment problem. European Journal of Operational Research. Vol. 176, no. 2, pp. 657–690.

Lomnicki Z. (1965). A branch-and-bound algorithm for the exact solution of the three-machine scheduling problem. Operational Research Quarterly. Vol. 16, no. 1, pp. 89–100.

Maassen, J.; Nieuwpoort, R.V.; Veldema, R.; Bal, H.; Kielmann, T.; Jacobs, C.; Hof-MAN, R. (2001). Efficient Java RMI for parallel programming. ACM Transactions on Programming Languages and Systems (TOPLAS). Vol. 23. No 6, pp. 747–775.

Mack, D.; Bortfeldt, A.; Gehring, H. (2004). A parallel hybrid local search algorithm for the container loading problem. International Transactions in Operational Research. Vol. 11, no. 5, pp. 511–533.

Malek, M.; Guruswamy, M.; Pandya, M.; Owens, H. (1989). Serial and Parallel Simulated Annealing and Tabu Search Algorithms for the Travelling Salesman Problem. Annals of Operations Research. Vol. 21, pp. 59–84.

Mans, B.; Mautor, T.; Roucairol, C. (1995). A parallel depth first search branch and bound algorithm for the quadratic assignment problem. European Journal of Operational Research (Elsevier). Vol. 81, no. 3, pp. 617–628.

Markus, S.; Kim, S.B.; Pantazopoulos, K.; Ocken, A.L.; Maharry, D.; (1996). Performance Evaluation of MPI Implementations and MPI Based Parallel ELLPACK Solvers. Proceedings of the Second MPI Developers Conference p. 162. Notre Dame - South Bend, IN.

Martin, O.; Otto, S.W.; Felten, E.W. (1991). Large-Step Markov Chains for the Travelling Salesman Problem. Complex Systems. Vol. 5, no. 3, pp. 299–326.

Martins, S.L.; Ribeiro, C.C.; Rodríguez N.; (2002). Parallel Computing Environments. Handbook of applied Optimization. Editado por Panos M. Pardalos and Mauricio G. C. Resende. Oxford University Press, Inc, pp. 1029–1043.

Matsumura, T.; Nakamura, M.; Tamaki, S. (2000). A Parallel Tabu Search and Its Hybridization with Genetic Algorithms. International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '00), p. 18,

Mautor, T. Roucairol, C. (1994). A new exact algorithm for the solution of quadratic assignment problems. Discrete Applied Mathematics (Elsevier). Vol. 55, no. 3, pp. 281–293.

McBryan, O. (1994). An overview of message passing environments. Parallel Computing. Vol. 20, no. 4, pp. 417–444.

McMahon, G.B.; Burton, P.G. (1967). Flowshop scheduling with the branch-and-bound method. Operations Research. Vol. 15, no. 3, pp. 473–481.

Michalewicz, Z. (1993). A Hierarchy of Evolution Programs: An Experimental Study, Evolutionary Computation. Vol.1, no.1, pp. 51–76.

Microsoft (2007). http://www.microsoft.com

Miller, B.P.; Callaghan, M.D.; Cargille, J.M.; Hollingsworth, J.K.; Irvin, R.B.; Karavanic, K.L.; Kunchithapadam, K.; Newhall, T. (1995). The Paradyn parallel performance measurement tool. Computer. Vol. 28, no. 11, pp. 37–46.

Minsky, M.; Papert, S. (1971). On some associative, parallel, and analog computations. Associative Information Techniques. Jacks, E.J. Ed. American Elsevier. New York.

Moe, R.; Sørevik, T. (2005). Parallel Branch and Bound Algorithms on Internet Connected Workstations. High Performance Computing and Communcations, First Inter-

national Conference, September 21–23, 2005. Proceedings. Sorrento, Italy: Springer Berlin / Heidelberg, pp. 768–775.

Moe, R. (2003). GRIBB – Branch-and-Bound Methods on the Internet. Editado por Springer Berlin/Heidelberg. Parallel Processing and Applied Mathematics, 5th International Conference, September 7–10, 2003. Revised Papers. Czestochowa, Poland: Springer Berlin/Heidelberg, pp. 1020–1027.

Montgomery, D.C. (2005). Design and Analysis of Experiments. 6th edition. John Wiley & Sons.

Moore, G.E. (1965). Cramming More Components Onto Integrated Circuits. Electronics. Vol. 38, no. 8.

Mosix (2006). http://www.mosix.org. Última consulta: 8/2006.

MPICH 2 (2006). http://www-unix.mcs.anl.gov/mpi/mpich. Última consulta: 8/2006.

MPI Forum (2009). http://www.mpi-forum.org/. Última consulta, 7/2009.

Mühlenbein, H. (1989). Parallel genetic algorithms, population genetics and combinatorial optimization. Proceedings of the Third International Conference on Genetic Algorithms, pp. 416–421. Morgan-Kauffmann.

Mühlenbein, H. (1991). Asynchronous parallel search by the parallel genetic algorithm. Ramachandran, V., editor. Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, pp. 526–533. IEEE Computer Society.

Mühlenbein, H.; Gorges-Schleuter, M.; Kramer, O. (1987). New solutions to the mapping problem of parallel systems: The evolution approach. Parallel Computing. Vol. 4, pp. 269–279.

Mühlenbein, H., Gorges-Schleuter, M., y Kramer, O. (1988). Evolution algorithms in combinatorial optimization. Parallel Computing. Vol. 7, pp. 65–85.

Nagar, A.; Heragu, S.; Haddock, J. (1996). A combined branch-and-bound and genetic algorithm based approach for a flowshop scheduling problem. Annals of Operations Research (Springer Nederlands). Vol. 63, no. 3, pp. 397–414.

Nawaz, M.; Enscore Jr., E.E.; Ham, I. (1983). A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem. Omega. Vol. 11, no. 1, pp. 91–95.

Nearchou, A.C. (2004). A novel metaheuristic approach for the flow shop scheduling problem. Engineering Applications of Artificial Intelligence. Vol. 17, no. 3, pp. 289–300.

Netlib Repository at UTK and ORNL. (2007).

Niar, S.; Freville, A. (1996) A Parallel Tabu Search Algorithm For The 0-1 Multidimensional Knapsack Problem. Baltzer Journals. ISSN 1063-7133/97.

Nowicki, E.; Smutnicki, C.; (1996). A fast tabu search algorithm for the permutation flow-shop problem. European Journal of Operational Research. Vol. 91, pp. 160–175.

Nowostawski, M.; Poli, R. (1999). Parallel Genetic Algorithms Taxonomy. : 3rd International Conference on Knowledge-Based Intelligent Information Engineering, pp. 88–92.

Nwana, V.; Darby-Dowman, K.; Mitra, G. (2005). A co-operative parallel heuristic for mixed zero–one linear programming: Combining simulated annealing with branch and bound. European Journal of Operational Research. Vol. 164, no. 1, pp. 12–23.

Ochi, L.S.; Silva, M.B.; Drummond, L. (2001). Metaheuristics based on GRASP and VNS for solving traveling purchaser problem. MIC'2001- 4th Metaheuristics International Conference, pp. 489–494.

Oguz, C.; Fung, Y.F.; Ercan, M.F.; Qi., X.T. (2003). Parallel Genetic Algorithm for a Flow-Shop Problem with Multiprocessor Tasks. Lecture Notes in Computer Science, Vol. 2667, pp. 987 – 997.

Okamoto, S.; Watanabe, I.; Iizuka, H. (1995). A new parallel algorithm for the n-Job, m-machine flow-shop scheduling problem. Systems and Computers in Japan (Wiley Periodicals, Inc.). Vol. 26, no. 2, pp. 10–21.

Okamoto, S.; Watanabe, I.; Iizuka, H. (1994). A new optimal algorithm for the permutation flow-shop problem and its parallel implementation. Computers & Industrial Engineering. Vol. 27, no. 1–4, pp. 39–42.

Oliveira, C.A.S.; Pardalos, P.N. (2004). Randomized parallel algorithms for the multidimensional assignment problem. Applied Numerical Mathematics (Elsevier). Vol. 49, no. 1, pp. 117–133.

The openMosix Project (2006). http://openmosix.sourceforge.net.

Open MPI: Open Source High Performance Computing (2007). http://www.open-mpi.org

Opttek Systems (2007). http://www.opttek.com.

OR-Library (2008). http://people.brunel.ac.uk/~mastjjb/jeb/info.html

Osman, I.; Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. OMEGA, The International Journal of Management Science. Vol. 17, no. 6, pp. 551–557.

Pacheco, P.S. (1997). Parallel Programming with MPI. Morgan Kaufmann Publishers Inc./Elsevier.

Palisade (2007). http://www.palisade–europe.com.

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 (Operational Research Society). Vol. 16, no. 1, pp. 101–107.

Parallel Performance Tools (2007). University of Wisconsin Madison. http://www.cs.wisc.edu/paradyn.

Parallel Virtual Machine (2007). http://www.csm.ornl.gov/pvm/.

Pardalos, P.M.; Pitsoulis, L.; Mavridou, T.; Resende, M.G.C. (1996). A Parallel GRASP for MAX-SAT Problems. Invited paper. PARA96 – Workshop on Applied Parallel Computing in Industrial Problems and Optimization, Lyngby, Denmark.

Pardalos, P.M.; Pitsoulis, L.; Mavridou, T.; Resende, M.G.C. (1995). Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP. Invited paper, Workshop on Parallel Algorithms for Irregularly Structured Problems. Lyon, France.

Pardalos, P.M.; Pitsoulis, L.S.; Resende, M.C.G. (1994). A parallel GRASP implementation for the quadratic assignment problem. Parallel Algorithms for Irregularly Structured Problems, Irregular'94.

Pardalos, P.M.; Pitsoulis, L.; Mavridou, T.; Resende, M.G.C. (1995). A Parallel GRASP Implementation for the Quadratic Assignment Problem, Proceedings of Parallel Algorithms for Irregularly Structured Problems (Irregular '94), Kluwer Academic Publishers.

Pardalos, P.; Rodgers, G.P. (1990). Parallel branch and bound algorithms for quadratic zero–one programs on the hypercube architecture. Annals of Operations Research (Springer Netherlands). Vol. 22, no. 1, pp. 271–292.

Pardalos, P.M., Crouse, J.V. (1989). A parallel algorithm for the quadratic assignment problem. Proceedings of the 1989 ACM/IEEE conference on Supercomputing. Reno, Nevada, United States: ACM, pp. 351–360.

Pardo Merino, A.; Ruiz Díaz, M.A. (2005). Análisis de datos con SPSS 13 Base. McGraw-Hill/Interamericana de España S.A.U.

Pessan, C.; Bouquard, J.L.; Néron, E. (2008). Genetic Branch-and-Bound or Exact Genetic Algorithm? Editado por Springer Berlin / Heidelberg. Artificial Evolution. Tours, France: Springer-Verlag. Vol. 4926/2008, pp. 136–147.

Pfister, G.F.; Brantley, W.C.; George, D.A.; Harvey, S.L.; Kleinfelder, W.J.; McAuliffe, K.P.; Melton, E.A.; Norton, V.A.; Weiss, J. (1985). The Research Parallel Processor Prototype (RP3): Introduction and Architecture. Proceedings of the 1985 International Conference on Parallel Processing, pp. 764–771.

Pierce, P. (1994). The NX message passing interface. Parallel Computing. Vol. 20, no. 4, pp. 463–480.

Pinedo, M. (2002). Scheduling. Theory, Algorithms and Systems. Prentice-Hall.

Pintér, J.D. (1996). Continuous Global Optimization Software: A Brief Review. Optima. Mathematical Programming Society Newsletter, no. 52, pp. 1–16.

Plastino, A.; Ribeiro, C.C.; Rodríguez, N. (2003). Developing SPMD applications with load balancing. Parallel Computing. Vol. 29, pp. 743–766.

Potts, C.N. (1980). An adaptive branching rule for the permutation Flow-shop problem. European Journal of Operational Research. Vol. 5, no.1. pp.19–25.

Proactive Parallel Suite. http://proactive.inria.fr (2008).

Puchinger, J.; Raidl, G. (2005). Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification. Editado por J. Mira and J.R. Alvarez. Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach. Las Palmas, Canary Islands, Spain, June 15–18, 2005, Proceedings, Part II: Springer-Verlag, pp. 41–43.

Quinn, M.J. (1987). Designing Efficient Algorithms for Parallel Computers. McGraw-Hill Inc. ISBN 0-07-051071-7.

Rad, S.F.; Ruiz, R.; Boroojerdian, N. (2007). New High Performing Heuristics for Minimizing Makespan in Permutation Flowshops. OMEGA, The International Journal of Management Science (In press).

Raghavan, N.R.S.; Waghmare, T. (2002). DPAC: An Object-Oriented Distributed and Parallel Computing Framework for Manufacturing Applications. IEEE Transactions on Robotics and Automation. Vol. 18, 4, pp. 431–443

Raidl, G. (2006). A unified view on hybrid metaheuristics. Editado por Francisco Almeida *et al*. Proceedings of the Hybrid Metaheuristics Workshop. Springer, pp. 1–12.

Rajendran, C.; Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research. Vol. 155, no. 2. pp. 426–438.

Randall, M.; Lewis, A. (2002). A Parallel Implementation of Ant Colony Optimization. Journal of Parallel and Distributed Computing. Vol. 62, pp. 1421–1432.

Reed, D.A.; Aydt, R.A.; DeRose, L.; Mendes, C.L.; Ribler, R.L.; Shaffer, E.; Simitci, H.;. Vetter, J.S.; Wells, D.R.; Whitmore, S.; Zhang, Y. (1998) Performance Analysis of Parallel Systems. Approaches and Open Problems. Proceedings of the Joint Symposium on Parallel Processing (JSPP) 1998, pp. 239–256.

Reeves, C.R. (1995). A genetic algorithm for flowshop sequencing. Computers and Operations Research. Vol. 22, no. 1, pp. 5–13.

Reeves, C.R. (1999). Landscapes, operators and heuristic search. Annals of Operations Research. Vol. 86, pp. 473–490. J.C. Baltzer AG, Science Publishers.

Reeves, C.R.; Yamada, T. (1998). Genetic algorithms, path relinking and the flowshop sequencing problem. Evolutionary Computation journal. Vol. 6, no. 1, pp. 230–234.

Resende, M.G.C.; Pardalos, P.M.; Eksioglu, S.D. (1999). Parallel Metaheuristics for Combinatorial Optimization. Presentado en la International School on Advanced Algorithmic Techniques for Parallel Computations with Applications, Natal (RN) Brasil.

Rettberg, R.; Thomas, R. (1986). Contention is No Obstacle to Shared-Memory multiprocessing. Communications of ACM. Vol. 29, no.12, pp. 1202–1212.

Rocha, P.L.; Ravetti, M.G.; Mateus, G.R. (2004). The meta-heuristic grasp as an upper bound for a branch and bound algorithm in a scheduling problem with non-related parallel machines and sequence-dependent setup times. EU/ME Workshop. Nottingham, UK.

Rochat, Y.; Taillard, E. (1995). Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics. Vol. 1, no. 1, pp. 147–167.

Roli, A.; Blum, C.; (2001) Critical Parallelization of Local Search for MAX-SAT. F. Esposito (Ed.): AI*IA 2001, LNAI 2175, pp. 147–158.

Roucairol, C. (1996). Parallel processing for difficult combinatorial optimization problems. European Journal of Operational Research. Vol. 92, pp. 573–590.

Roucairol, C. (1989). Parallel Computing in Combinatorial Optimization. Computer Physics Reports (Elsevier). Vol. 11, no. 1–6, pp. 195–220.

Roucairol, C. (1987). A parallel branch and bound algorithm for the quadratic assignment problem. Discrete Applied Mathematics (Elsevier). Vol. 18, no. 2, pp. 211–225.

Roussel-Ragot, P; Dreyfus, G. (1990). A Problem Independent Parallel Implementation of Simulated Annealing : Models and Experiments. IEEE Transactions on Computer-Aided Design. Vol. 5, no. 8, pp. 827–835.

Ruiz, R.; Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research. Vol. 177, pp. 2033–2049.

Ruiz, R.; Maroto, C.; Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. OMEGA, the International Journal of Management Science. Vol. 34, pp. 461–476.

Ruiz,R.; Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research. Vol. 165, pp. 479–494.

Ruiz, R.; Maroto, C.; Alcaraz, J. (2005). Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. European Journal of Operational Research (Elsevier). Vol. 165, no. 1, pp. 34–54.

Salhi, A.; Glaser, H.; De Roure D.; (1998) Parallel implementation of a genetic-programming based tool for symbolic regression. Information Processing Letters 66, pp. 299–307.

Sanjai, S.; Paredis, C.J.J.; Gupta, S.K.; N. Talukdar, S.N. (1998). 3D Spatial Layouts Using A-Teams. Proceedings of DETC'98. Atlanta, Georgia, USA: ASME.

Sankaran, S.; Squyres, J.M.; Barrett, B.; Lumsdaine, A.; Duell, J.; Hargrove, P.; Roman, E.; (2003). The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing. LACSI Symposium. (tb. en The Lam/Mpi Checkpoint/Restart Framework: System-Initiated Checkpointing. Sankaran *et al*. (2005). International Journal of High Performance Computing Applications. Vol. 19, pp. 479–493)

Sanvicente, H; Frausto, J. (2002). MPSA: A Methodology to Parallelize Simulated Annealing and Its Application to the Travelling Salesman Problem. C.A. Coello Coello *et al.,* Eds.): MICAI 2002, LNAI 2313, pp. 89–97.

Schmidt-Voigt, M. (1994). Efficient parallel communication with the nCUBE 2S processor. Parallel Computing. Vol. 20, no. 4, pp. 509–530

Schulze, J.; Fahle, T. (1999). A parallel algorithm for the vehicle routing problem with time window constraints. Annals of Operations Research. Vol. 86, pp. 585–607.

Schwehm, M. (1993). A Massively Parallel Genetic Algorithm on the MasPar MP-1. Artificial Neural Nets and Genetic Algorithms, Albrecht R.F, Reeves C.R & Steele N.C (Eds) Springer-Verlag: New York. pp. 503–507.

SETI (2007). SETI@home project. http://setiathome.berkeley.edu.

Sevkli, M.; Aydin, M.E. (2006). A variable neighbourhood search algorithm for job shop scheduling problems. Evolutionary Computation in Combinatorial Optimization, Proceedings 3906, pp. 261–271.

Shi, L.; Ólafsson, S.; Sun, N. (1999). New parallel randomized algorithms for the traveling salesman problem. Computers & Operations Research. Vol. 26, pp. 371–394.

Sittig, D.F.; Foulser, D.; Carriero, N.; McCorkle, G; Miller, P.L. (1991) A Parallel Computing Approach to Genetic Sequence Comparison: The Master-Worker Paradigm with Interworker Communication. Computers and Biomedical Research, no. 24, pp. 152–169.

Skjellum, A.; Smith, S.G.; Doss, N.E.; Leung, A.P.; Morari, M. (1994). The design and evolution of Zipcode. Parallel Computing. Vol. 20, no. 4, pp. 565–596.

Solar, M.; Parada, V.; Urrutia, R.; (2002) A parallel genetic algorithm to solve the set-covering problem. Computers & Operations Research, Vol. 29, pp. 1221–1235.

Spencer, T.H. (1997). Time-Work Tradeoffs for Parallel Algorithms. Journal of the ACM. Vol. 44, no. 5, pp. 742–778.

Stanislav, G.A. (2007). FreeBSD Developers' Handbook. Cap. 7. http://www.freebsd.og.

Stankovic, N.; Zhang, K.; (1999). Visual programming for message passing systems. International Journal of Software Engineering and Knowledge Engineering. Vol. 9, pp. 397–423.

Steinhöfel, K.; Albrecht, A.; Wong, C.K. (2002). Fast parallel heuristics for the job shop scheduling problem. Computers & Operations Research. Vol. 29, pp. 151–169.

Sterling, T.; Becker, D.J.; Dorband, J.E.; Savarese, D.; Ranawake, U.A.; Packer, C.V. (1995). Beowulf: A Parallel Workstation For Scientific Computation. ICPP '95. Proceedings of the 24th International Conference on Parallel Processing.

Stewart, R.M.; Sulley, C.S. (1966). Compact diagnostic co-processing for avionic use. Cambridge University Press.

Stützle, T. (1998a). Applying iterated local search to the permutation flow shop problem. Technical Report, AIDA-98-04, FG Intellektik, TU Darmstadt.

Stützle, T. (1998b). Parallelization Strategies for Ant Colony Optimization. A.E. Eiben *et al.,* Eds.): PPSN V, LNCS 1498, pp. 722–731.

Stützle, T. (1998c). An ant approach to the flow shop problem. Proceedings of the Sixth European Congress on Intelligent Techniques and Soft Computing. Aachen, Germany: Verlag-Mainz, pp. 1560–1564.

Suh, J.; Van Gucht, D. (1987). Distributed genetic Algorithms, Tech. Report 225, Computer Science Department, Indiana University, Bloomington, IN.

Sunderam, V.S. (1990) PVM: A framework for parallel distributed computing. Concurrency: Practice and Experience. Vol. 2, no. 4, pp. 315–339.

Sunderam, V.S.; Geist, G.A.; Dongarra, J.; Manchek, R. (1994). The PVM concurrent computing system: Evolution, experiences, and trends. Parallel Computing. Vol. 20, no. 4, pp. 531–545.

Surry, P.D.; Radcliffe, N.; (1994) RPL2: A Language and Parallel Framework for Evolutionary Computing. Y. Davidor, H.-P. Schwefel, and R. Manner, editors, Parallel Problem Solving From Nature III, pp. 628–637. Springer-Verlag.

Taguchi, G.; Elsayed A.; Hsiahg, T.C. (1989). Quality engineering in production systems. McGraw-Hill.

Taillard, E. (2005). Problem instances, flowshop sequencing, summary of best known lower and upper bounds of Taillard's instances. http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/flowshop.dir/best_lb_up.txt

Taillard, E. (1994). Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. ORSA Journal on Computing. Vol. 6, no. 2, pp. 108–117.

Taillard, E. (1993). Benchmarks for basic scheduling problems. European Journal of Operational Research, Vol. 64, pp. 278–285.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. Parallel Computing, 17, pp. 443 455.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research, Vol. 47, pp. 65 74.

Talbi, E.G. (2002). A Taxonomy of Hybrid Metaheuristics. Journal of Heuristics. Vol. 8, pp. 541–564.

Talbi, E.G.;Roux, O.; Fonlupt C.; Robillard, D. (2001). Parallel Ant Colonies for the quadratic assignment problem. Future Generation Computer Systems. Vol. 17, pp. 441–449.

Talbi, E.G.; Hafidi, Z.; Geib, J.M. (1998a). A parallel adaptive tabu search approach. Parallel Computing. Vol. 24, pp. 2003–2019.

Talbi, E.G.; Hafidi, Z.; Kebbal, D.; Geib, J.M. (1998b). A fault-tolerant parallel heuristic for assignment problems. Future Generation Computer Systems. Vol. 14, no. 5–6, pp. 425–438.

Talukdar, S.N. (1998). Autonomous cyber agents: Rules for collaboration and concurrency. Proceedings of the Hawaii International Conference on System Sciences. Big Island, HI, USA: Institute of Electrical and Electronics Engineers Computer Society, pp. 57–61.

Talukdar, S.N.; Baerentzen, L.; Gove, A.; De Souza, P. (1998). Asynchronous Teams: Cooperation Schemes for Autonomous Agents. Journal of Heuristics. Vol. 4, no. 4, pp. 295–321.

Talukdar, S.N.; Ramesh, V.C. (1992). A-teams for real-time operations. International Journal of Electrical Power and Energy Systems, Vol. 14, no. 2–3, pp. 138–143.

Tamaki, H.; Nishikawa, Y. (1992). A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling. Proceedings of 2nd Conference on Parallel Problem Solving from Nature, pp. 573–582. Elsevier Science.

TOP500 supercomputer sites (2007). http://www.top500.org. Última revisión: 8/2007.

Tosic, P.T. (2004). A perspective on the future of massively parallel computing: fine-grain vs. coarse-grain parallel models. Comparison & Contrast. Proceedings of the 1st Conference On Computing Frontiers. Association for Computing Machinery.

Toulouse, M.; Thulasiraman, K.; Glover, F. (1999). Multi-level Cooperative Search: A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning. P. Amestoy *et al.,* Eds.): Euro-Par'99, LNCS 1685, pp. 533–542.

Tseng, F.T.; Stafford, E.F. (2008). New MILP models for the permutation flowshop problem. Journal of the Operational Research Society. Vol. 59, no. 10, pp. 1373–1386.

Tucker, L.W.; Mainwaring, A. (1994). CMMD: Active messages on the CM-5. Parallel Computing. Vol. 20, no. 4, pp. 481–496.

Turner, S.; Booth, D. (1987). Comparison of heuristics for flowshop sequencing. OMEGA. Vol. 15, no. 1, pp. 75–85.

Valiant, L.G. (1990). A bridging model for parallel computation. Communications of the ACM. Vol. 33, no. 8, pp. 103–111.

Vallada, E.; Ruiz, R. (2007). Cooperative metaheuristics for the permutation flowshop scheduling problem. 22nd European Conference on Operational Research (EURO'07).

Vallada, E. (2006). Secuenciación en Talleres de Flujo con Fechas de entrega. Nuevos Métodos Metaheurísticos y Computación Paralela. Tesis doctoral. Universidad Politécnica de Valencia.

Varadharajan, T.K.; Rajendran, C. (2005). A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. European Journal of Operational Research. Vol. 167. 772–795.

Vaughan, P.L.; Skjellum, A.; Reese, D.S.; Cheng, F.C. (1995). Migrating from PVM to MPI, part I: The Unify System. Fifth Symposium on the Frontiers of Massively Parallel Computation, pp. 488–495. IEEE Computer Society Press.

Vazquez, G.E.; Brignole, N.B.; Diaz, S.; Bandoni, J.A. (2002). Optimization of industrial problems using parallel processing under distributed environments. Chemical Engineering Communications. Vol. 189, pp. 642–656.

Verhoeven, M.G.A.; Aarts, E.H.L., (1995). Parallel Local Search. Journal of heuristics, 1: 43–65. Kluwer Academic Publishers.

Vigo, D.; Maniezzo, V.; (1997). A Genetic/Tabu Thresholding Hybrid Algorithm for the Process Allocation Problem. Journal of Heuristics. Vol. 3, pp. 91–110.

Van Voorst, B.; Seidel, S. (2000). Comparison of MPI Implementations on a Shared Memory Machine. IPDPS Workshops 2000, pp. 847–854.

Walker, D.W. (1994). The design of a standard message passing interface for distributed memory concurrent computers. Parallel Computing. Vol. 20, no. 4, pp. 657–673

Widmer, M.; Hertz, A. (1989). A new heuristic method for the flow shop sequencing problem. European Journal of Operational Research. Vol. 41, no. 2, pp. 186–193.

Wodecki, M.; Bozejko, W. (2002). Solving the Flow Shop Problem by Parallel Simulated Annealing. R. Wyrzykowski *et al.,* Eds.): PPAM 2001, LNCS 2328, pp. 236–244.

Yamada, T.; Reeves, C.R. (1997). Permutation flowshop scheduling by genetic local search. Proceedings of the 2nd IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems. London, UK: IEEE, pp. 232–238.

Yan, J.C.; (1994). Performance tuning with AIMS-an Automated Instrumentation and Monitoring System for multicomputers. Software Technology, Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences. Vol. 2, no. 4–7, pp. 625–633.

Ying, K.-C., y C.-J. Liao. An ant colony system for permutation flow-shop sequencing. Computers & Operations Research 31 (2004): 791–801.

Zobolas, G.I.; Tarantilis, C.D.; Ioannou, G. (2008). Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. Computers and Operations Research. doi: 10.1016/j.cor.2008.01.007.

9 Anexos

9.1 Anexo 1: Valores de makespan

Valores mínimos de *makespan* considerados en este trabajo.

20x5	ta001	ta002	Ta003	ta004	ta005	ta006	ta007	ta008	ta009	ta010
	1278	1359	1081	1293	1235	1195	1234	1206	1230	1108
20x10	ta011	ta012	Ta013	ta014	ta015	ta016	ta017	ta018	ta019	ta020
	1582	1659	1496	1377	1419	1397	1484	1538	1593	1591
20x20	ta021	ta022	Ta023	ta024	ta025	ta026	ta027	ta028	ta029	ta030
	2297	2099	2326	2223	2291	2226	2273	2200	2237	2178
50x5	ta031	ta032	Ta033	ta034	ta035	ta036	ta037	ta038	ta039	ta040
	2724	2834	2621	2751	2863	2829	2725	2683	2552	2782
50x10	ta041	ta042	Ta043	ta044	ta045	ta046	ta047	ta048	ta049	ta050
	2991	2867	2839	3063	2976	3006	3093	3037	2897	3065
50x20	ta051	ta052	ta053	ta054	ta055	ta056	ta057	ta058	ta059	ta060
	3856	3707	3643	3731	3619	3687	3706	3700	3755	3767
100x5	ta061	ta062	ta063	ta064	ta065	ta066	ta067	ta068	ta069	ta070
	5493	5268	5175	5014	5250	5135	5246	5094	5448	5322
100x10	ta071	ta072	ta073	ta074	ta075	ta076	ta077	ta078	ta079	ta080
	5770	5349	5676	5781	5467	5303	5595	5617	5871	5845
100x20	ta081	ta082	ta083	ta084	ta085	ta086	ta087	ta088	ta089	ta090
	6228	6210	6271	6269	6319	6403	6292	6423	6275	6434
200x10	ta091	ta092	ta093	ta094	ta095	ta096	ta097	ta098	ta099	ta100
	10862	10480	10922	10889	10524	10329	10854	10730	10438	10675
200x20	ta101	ta102	ta103	ta104	ta105	ta106	ta107	ta108	ta109	ta110
	11195	11223	11337	11299	11260	11189	11386	11334	11192	11313
500x20	ta111	ta112	ta113	ta114	ta115	ta116	ta117	ta118	ta119	ta120
	26059	26520	26371	26456	26334	26477	26389	26560	26005	26457

Tabla 77: Batería de Taillard (1993). Valores mínimos de makespan empleados en este documento

9.2 Anexo 2: Patologías en las observaciones

Autores como Crainic y Gendreau (2002), obtuvieron mejores resultados en cuanto a número de observaciones en las que el algoritmo secuencial superaba al paralelo. En su caso fueron 6 de las 180 observaciones que realizaron, las que no llegaron al valor propuesto de la función objetivo.

El algoritmo empleado no es determinista, sino que parte en cada experimento de soluciones generadas de forma aleatoria, que son mejoradas en un proceso de búsqueda local. Para la búsqueda del tiempo de ejecución con varios procesadores del algoritmo paralelo, la condición de parada es que se alcance la misma calidad de soluciones que se obtuvo empleando un solo procesador.

Por ejemplo, en el problema ta024, con 20 trabajos y 20 máquinas, en la primera observación con el segundo tratamiento, con tres procesadores se llegó a un valor de *makespan* de 2253, cuando con dos procesos en un solo procesador, se había alcanzado tras cinco ejecuciones un promedio de 2247.

Dado que se está partiendo, en ambos casos, de soluciones generadas de forma aleatoria, es posible, y de hecho así sucede, que se genere al azar una solución de partida con el algoritmo secuencial cuya calidad no se llegue a alcanzar con el algoritmo paralelo. Esta es una situación no deseable, que trataremos de evitar en una versión posterior del algoritmo, pero vale la pena mencionar que, en el caso del algoritmo síncrono de grano grueso ha sucedido en 719 de las 8640 observaciones, un porcentaje elevado, del 8% de las observaciones.

9.3 Anexo 3: Tablas para la eficiencia y relación señal/ruido en el algoritmo CLMpf

9.3.1 Estudio según el número de procesadores

3 procs	1	2	3	4	5	6	7	8
20x20	0.69	0.43	1.08	0.75	0.30	0.50	1.10	0.55
50x5	0.13	0.32	0.27	0.45	0.71	0.39	0.08	0.22
50x10	0.38	0.61	0.63	0.30	0.50	0.80	0.36	0.59
50x20	0.42	0.36	0.58	0.60	0.39	0.81	0.62	0.49
100x5	0.12	0.18	0.19	0.12	0.16	0.63	0.19	0.24
100x10	0.20	0.43	0.16	0.23	0.35	0.27	0.21	0.66
100x20	0.55	0.43	0.42	0.55	0.30	0.37	0.60	0.46
200x10	0.40	0.09	0.08	0.28	0.15	0.25	0.24	0.11
200x20	0.42	0.46	0.35	0.41	0.52	0.26	0.36	0.55
s/n	**-12.93**	**-13.59**	**-14.41**	**-11.70**	**-11.59**	**-8.75**	**-14.24**	**-12.14**

6 procs	1	2	3	4	5	6	7	8
20x20	0.11	0.39	0.56	0.40	0.10	0.22	0.38	0.16
50x5	0.05	0.15	0.09	0.25	0.07	0.18	0.10	0.11
50x10	0.28	0.56	0.55	0.19	0.35	0.51	0.49	0.36
50x20	0.42	0.50	0.22	0.30	0.41	0.37	0.59	0.28
100x5	0.06	0.09	0.09	0.06	0.08	0.31	0.09	0.12
100x10	0.10	0.22	0.08	0.12	0.17	0.14	0.11	0.33
100x20	0.28	0.21	0.21	0.28	0.15	0.18	0.30	0.23
200x10	0.07	0.13	0.09	0.14	0.17	0.19	0.14	0.13
200x20	0.31	0.29	0.34	0.42	0.36	0.27	0.31	0.29
s/n	**-21.00**	**-15.17**	**-17.94**	**-17.36**	**-18.15**	**-13.44**	**-16.22**	**-15.36**

9 procs	1	2	3	4	5	6	7	8
20x20	0.19	0.19	0.37	0.26	0.15	0.21	0.23	0.19
50x5	0.06	0.07	0.09	0.13	0.04	0.11	0.06	0.08
50x10	0.23	0.24	0.35	0.27	0.32	0.29	0.32	0.34
50x20	0.24	0.35	0.49	0.32	0.24	0.23	0.32	0.33
100x5	0.08	0.07	0.07	0.08	0.07	0.03	0.06	0.17
100x10	0.09	0.22	0.19	0.08	0.24	0.13	0.11	0.15
100x20	0.23	0.34	0.34	0.26	0.25	0.30	0.33	0.23
200x10	0.18	0.08	0.02	0.13	0.07	0.12	0.19	0.07
200x20	0.22	0.24	0.21	0.24	0.42	0.23	0.24	0.25
s/n	**-18.76**	**-18.82**	**-24.10**	**-17.17**	**-21.16**	**-22.19**	**-19.43**	**-17.65**

12 procs	1	2	3	4	5	6	7	8
20x20	0.11	0.13	0.31	0.18	0.10	0.16	0.25	0.10
50x5	0.07	0.07	0.06	0.12	0.08	0.12	0.05	0.03
50x10	0.14	0.19	0.24	0.20	0.20	0.18	0.23	0.24
50x20	0.22	0.31	0.34	0.20	0.21	0.12	0.32	0.32
100x5	0.03	0.09	0.06	0.08	0.12	0.06	0.07	0.11
100x10	0.14	0.17	0.17	0.15	0.18	0.08	0.23	0.11
100x20	0.19	0.24	0.27	0.25	0.21	0.25	0.23	0.25
200x10	0.12	0.08	0.08	0.11	0.11	0.05	0.05	0.17
200x20	0.18	0.19	0.23	0.23	0.26	0.18	0.23	0.25
s/n	**-22.45**	**-18.90**	**-19.44**	**-17.26**	**-17.68**	**-20.84**	**-20.72**	**-21.21**

9.3.2 Estudio según el tamaño de problema

20x20	1	2	3	4	5	6	7	8
e3	2.11	5.50	0.86	1.78	11.12	3.97	0.83	3.35
e6	86.97	6.74	3.22	6.33	94.63	20.58	6.77	39.42
e9	27.83	29.06	7.32	15.24	44.24	22.21	19.67	28.60
e12	79.48	62.93	10.47	31.11	98.50	41.62	15.77	98.55
S/N	-16.91	-14.16	-7.38	-11.34	-17.93	-13.44	-10.32	-16.28

50x5	1	2	3	4	5	6	7	8
e3	57.09	9.78	13.39	4.92	1.99	6.65	148.73	21.25
e6	461.43	41.68	121.93	15.61	194.52	30.49	91.43	84.59
e9	281.48	213.50	113.86	61.79	688.04	82.38	305.95	148.91
e12	233.62	206.34	252.85	64.71	150.39	71.22	379.60	832.17
S/N	-24.12	-20.71	-20.99	-15.65	-24.13	-16.78	-23.64	-24.34

50x10	1	2	3	4	5	6	7	8
e3	6.94	2.66	2.55	11.35	3.98	1.57	7.91	2.86
e6	12.46	3.19	3.30	26.33	7.96	3.87	4.10	7.82
e9	18.50	17.79	8.14	14.07	10.05	12.12	9.73	8.80
e12	47.61	29.00	17.33	24.43	25.39	31.95	19.20	17.47
S/N	-13.30	-11.19	-8.94	-12.80	-10.74	-10.93	-10.10	-9.65

50x20	1	2	3	4	5	6	7	8
e3	5.58	7.82	2.97	2.81	6.67	1.52	2.64	4.16
e6	5.54	4.07	21.56	11.44	5.84	7.25	2.88	12.45
e9	17.43	8.00	4.15	10.04	16.92	18.85	9.51	9.04
e12	20.41	10.61	8.67	24.89	21.73	70.37	9.67	9.74
S/N	-10.88	-8.82	-9.70	-10.90	-11.07	-13.89	-7.91	-9.47

100x5	1	2	3	4	5	6	7	8
e3	65.18	31.18	28.71	71.53	40.63	2.55	28.94	16.95
e6	260.71	124.71	114.83	286.11	162.51	10.21	115.78	67.81
e9	143.87	214.50	203.57	139.39	196.85	1193.76	307.99	36.37
e12	1014.32	138.28	280.86	170.55	74.00	282.22	183.78	77.89
S/N	-25.69	-21.04	-21.96	-22.22	-20.74	-25.71	-22.02	-16.97

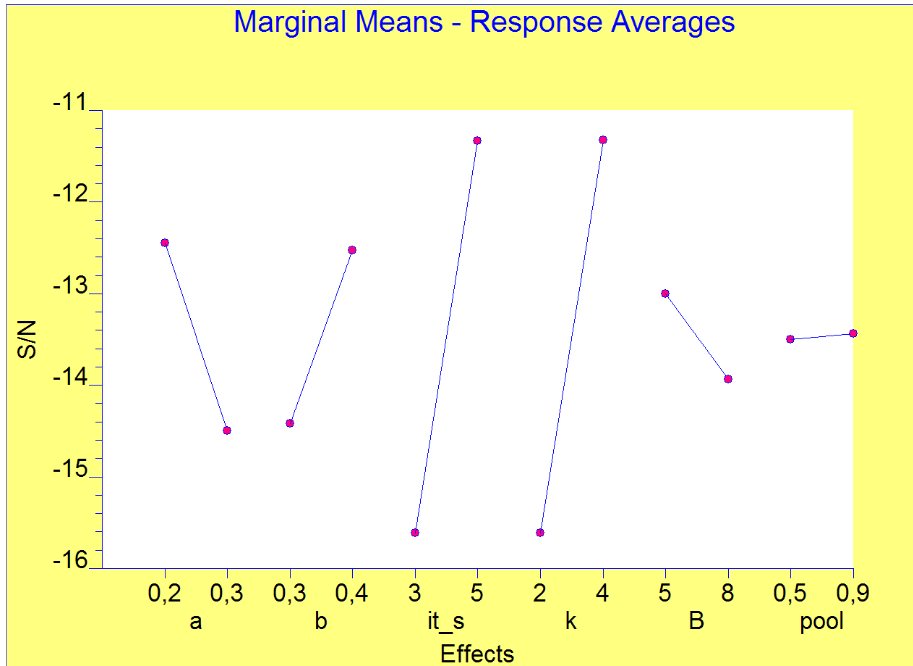
100x10	1	2	3	4	5	6	7	8
e3	24.74	5.39	37.49	18.30	8.38	13.56	21.67	2.33
e6	98.96	21.54	149.95	73.19	33.52	54.25	86.66	9.32
e9	117.94	21.12	28.49	140.90	17.87	56.13	81.84	44.52
e12	52.27	33.92	35.41	45.85	30.60	176.42	19.08	86.66
S/N	-18.66	-13.12	-17.98	-18.42	-13.54	-18.76	-17.19	-15.53

100x20	1	2	3	4	5	6	7	8
e3	3.28	5.45	5.74	3.29	11.36	7.44	2.76	4.82
e6	13.13	21.81	22.95	13.14	45.45	29.76	11.04	19.27
e9	19.46	8.86	8.72	14.49	15.45	11.24	9.37	18.37
e12	28.07	16.95	13.65	16.56	22.79	15.51	18.20	15.99
S/N	-12.04	-11.23	-11.06	-10.74	-13.76	-12.04	-10.15	-11.65

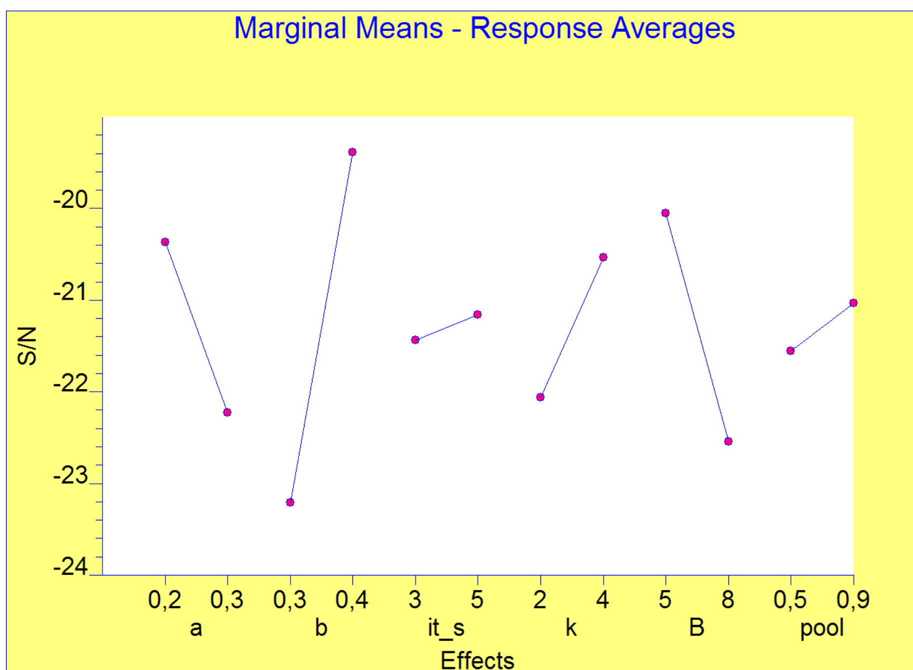
200x10	1	2	3	4	5	6	7	8
e3	6.17	133.30	148.47	13.19	41.91	15.44	18.10	88.38
e6	183.33	60.13	113.25	52.54	36.00	28.60	48.27	56.17
e9	29.36	156.21	1914.37	55.30	181.36	73.98	28.09	213.24
e12	74.91	173.58	154.05	81.35	89.53	372.13	398.58	36.28
S/N	-18.66	-21.17	-27.65	-17.04	-19.41	-20.88	-20.91	-19.94

9.4 *Anexo 4: Gráficas de medias marginales para la relación señal/ruido para distintos tamaños de problema para CLMpf*

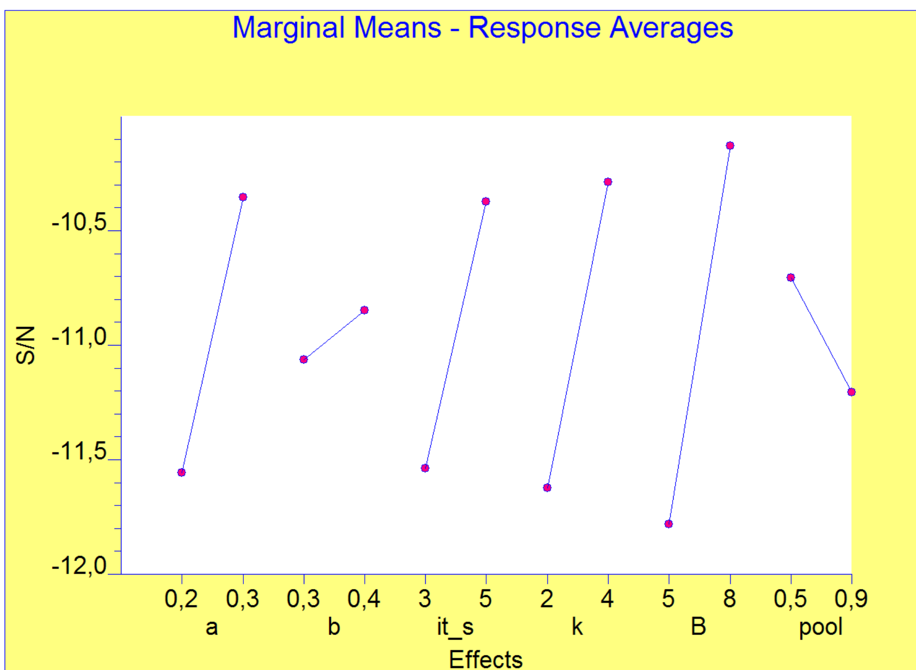
9.4.1 20x20



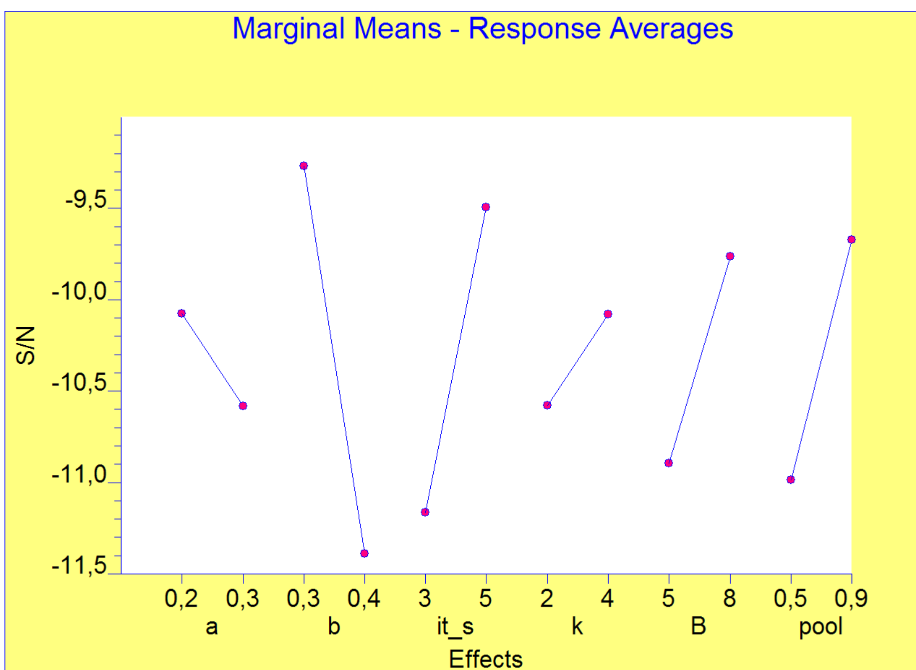
9.4.2 50x5



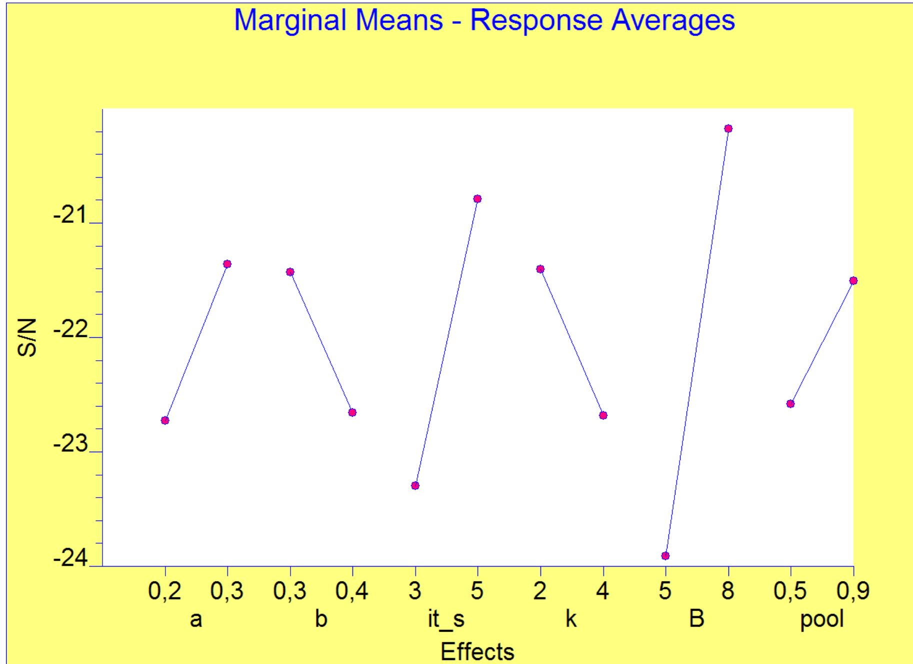
9.4.3 50x10



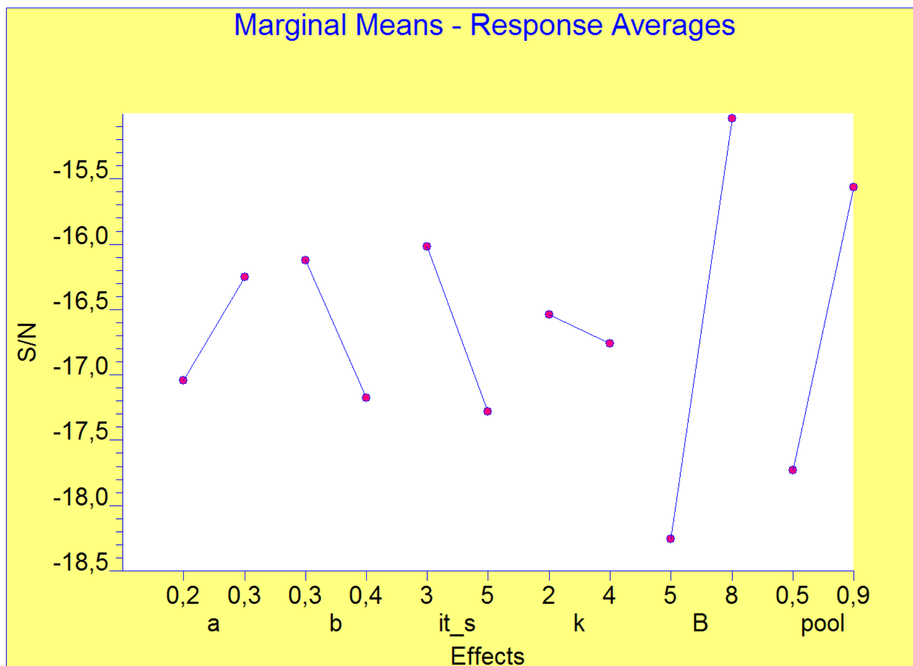
9.4.4 50x20



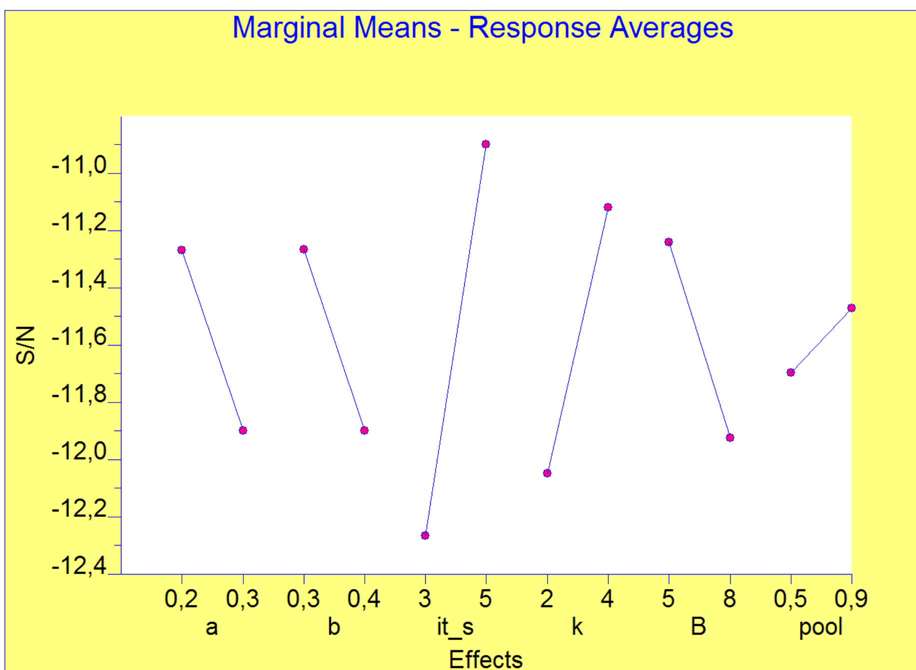
9.4.5 100x5



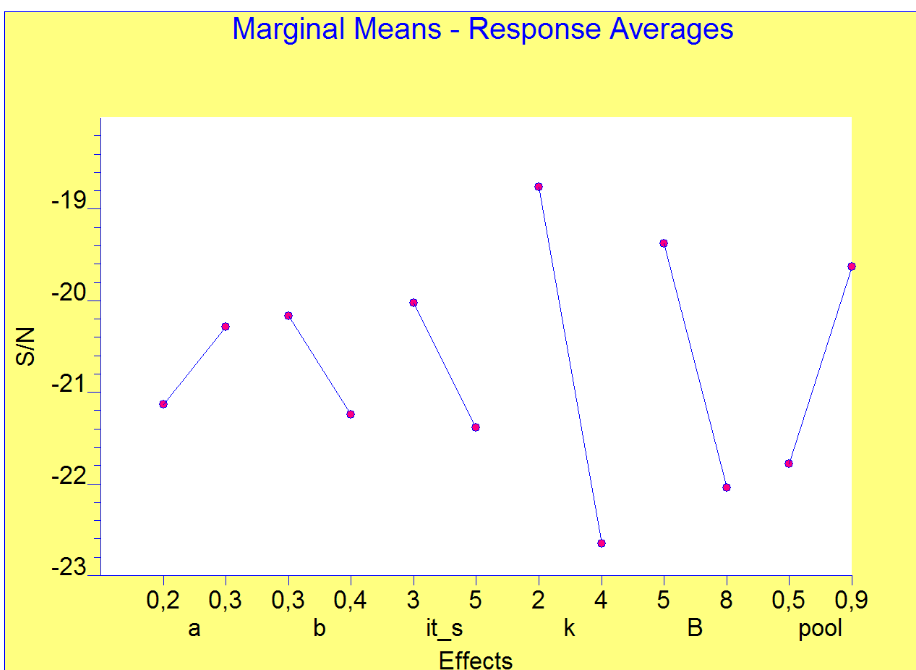
9.4.6 100x10



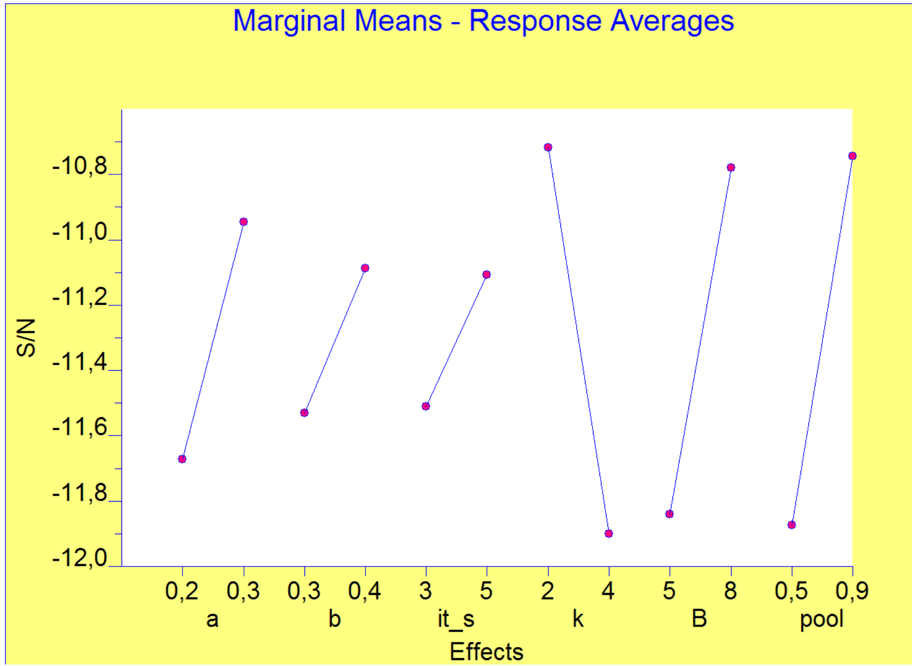
9.4.7 100x20



9.4.8 200x10



9.4.9 200x20



9.5 Anexo 5: Contrastes y tratamientos empleados en la experimentación con CLMpf

Contraste	Origen
A	α, -β x it_s, -K x B, β x B x *pool*, it_s x K x *pool*
B	β, - α x it_s, -K x *pool*, α x B x *pool*, it_s x K x B
C	It_s, - α x β, -B x *pool*, α x k x *pool*, β x K x B
D	K, - α x β, - β x *pool*, α x it_s x *pool*, β x it_s x B
E	B, - α x K, -it_s x *pool*, α x β x *pool*, β x it_s x K
F	*pool*, - β x K, -it_s x B, α x β x B, α x it_s x K
G	- α x *pool*, - β x B, -it_s x K, α x β x K, K x B x *pool*, β x it_s x *pool*, α x it_s x B

Tabla 78: Composición de los contrastes

		Tratamiento							
Cont.	**Factor**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A	α	-1	-1	-1	-1	1	1	1	1
B	β	-1	-1	1	1	1	1	-1	-1
C	It_s	-1	-1	1	1	-1	-1	1	1
D	K	-1	1	1	-1	-1	1	1	-1
E	B	-1	1	1	-1	1	-1	-1	1
F	*pool*	-1	1	-1	1	1	-1	1	-1
G		-1	1	-1	1	-1	1	-1	1

Tabla 79: Conjunto de tratamientos

9.6 Anexo 6: Comparación de los resultados de eficiencia de los algoritmos CLMpg y CLMpf

	CLMpg		**CLMpf**	
np	**6**	**12**	**6**	**12**
20x20	0.36	0.41	0.47	0.20
50x5	0.24	0.12	0.30	0.11
50x10	0.65	0.26	0.59	0.25
50x20	0.59	0.76	0.74	0.37
100x5	0.30	0.29	0.17	0.10
100x10	0.71	0.45	0.42	0.21
100x20	1.02	0.59	0.56	0.30
200x10	0.76	0.46	0.31	0.16
200x20	1.04	0.63	0.49	0.28
Promedio	0.63	0.44	0.45	0.22

9.7 Anexo 7: Resultados de la experimentación con el algoritmo pBDS

9.7.1 Serie 20x5

	Tiempo según número de procesos					Eficiencia según número de procesos			
	1(1+1)	3	6	9	12	3	6	9	12
ta001	0.29	0.18	0.18	0.18	0.19	0.52	0.26	0.17	0.13
ta002	0.10	0.07	0.07	0.07	0.07	0.50	0.24	0.16	0.12
ta003	9.28	3.87	1.25	1.03	0.97	0.80	1.24	1.00	0.80
ta004	34.07	25.88	5.82	4.95	6.00	0.44	0.98	0.76	0.47
ta005	25.97	32.36	38.45	26.41	14.44	0.27	0.11	0.11	0.15
ta006	36.05	10.39	10.10	9.69	13.57	1.16	0.59	0.41	0.22
ta007	5.30	2.18	1.31	0.70	0.53	0.81	0.67	0.84	0.83
ta008	3.42	0.23	0.24	0.24	0.24	4.94	2.41	1.60	1.19
ta009	13.16	0.24	0.25	0.24	0.24	18.25	8.81	6.16	4.58
ta010	0.30	0.18	0.17	0.17	0.17	0.57	0.29	0.19	0.14
Prom.	12.79	7.56	5.78	4.37	3.64	2.83	1.56	1.14	0.86

Tabla 80: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 20x5

20x5	3 a 6	3 a 9	3 a12	6 a 9	6 a 12	9 a 12
ta001	0.50	0.33	0.25	0.66	0.50	0.75
ta002	0.49	0.32	0.24	0.66	0.50	0.74
ta003	1.55	1.25	1.00	0.81	0.53	0.80
ta004	2.22	1.74	1.08	0.78	0.41	0.62
ta005	0.42	0.41	0.56	0.97	0.91	1.37
ta006	0.51	0.36	0.19	0.70	0.36	0.54
ta007	0.83	1.04	1.02	1.25	0.65	0.98
ta008	0.49	0.32	0.24	0.66	0.49	0.74
ta009	0.48	0.34	0.25	0.70	0.50	0.74
ta010	0.50	0.34	0.25	0.66	0.50	0.75
Prom.	0.80	0.64	0.51	0.79	0.54	0.80

Tabla 81: Eficiencia incremental generalizada para la serie 20x5

9.7.2 Serie 20x10

	Tiempo según número de procesos					Eficiencia según número de procesos			
	2 en 1	3	6	9	12	3	6	9	12
ta011	1.21	0.69	0.26	0.26	0.25	0.59	0.79	0.51	0.41
ta012	88.33	40.84	18.49	20.27	19.22	0.72	0.80	0.48	0.38
ta013	39.46	3.40	3.11	2.93	3.14	3.87	2.11	1.49	1.05
ta014	479.55	102.41	214.68	268.53	311.18	1.56	0.37	0.20	0.13
ta015	1.95	1.11	0.89	0.84	0.78	0.58	0.37	0.26	0.21
ta016	0.39	0.23	0.17	0.23	0.23	0.57	0.37	0.19	0.14
ta017	91.00	31.55	29.80	14.65	5.40	0.96	0.51	0.69	1.41
ta018	202.23	154.66	5.23	5.66	2.64	0.44	6.44	3.97	6.38
ta019	39.44	1.29	1.29	1.12	1.13	10.16	5.10	3.92	2.90
ta020	6.65	1.76	1.76	1.77	1.80	1.26	0.63	0.42	0.31
Prom.	96.14	33.79	27.57	31.63	34.58	2.07	1.79	1.24	1.37

Tabla 82: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 20x10

20x10	3 a 6	3a9	3a12	6a9	6a12	9a12
ta011	1.35	0.87	0.69	0.65	0.53	0.79
ta012	1.10	0.67	0.53	0.61	0.53	0.79
ta013	0.55	0.39	0.27	0.71	0.47	0.70
ta014	0.24	0.13	0.08	0.53	0.43	0.65
ta015	0.62	0.44	0.36	0.71	0.54	0.81
ta016	0.65	0.33	0.24	0.50	0.50	0.75
ta017	0.53	0.72	1.46	1.36	1.36	2.04
ta018	14.78	9.11	14.63	0.62	1.07	1.61
ta019	0.50	0.39	0.29	0.77	0.49	0.74
ta020	0.50	0.33	0.24	0.66	0.49	0.74
Prom.	2.08	1.34	1.88	0.71	0.64	0.96

Tabla 83: Eficiencia incremental generalizada para la serie 20x10

9.7.3 Serie 20x20

	Tiempo según número de procesos					Eficiencia según número de procesos			
	2 en 1	3	6	9	12	3	6	9	12
ta021	28.66	26.14	11.83	18.77	7.54	0.37	0.40	0.17	0.32
ta022	10.23	2.30	1.91	1.58	1.62	1.48	0.89	0.72	0.53
ta023	39.52	47.89	54.79	7.89	11.77	0.28	0.12	0.56	0.28
ta024	0.96	0.57	0.58	0.59	0.58	0.56	0.28	0.18	0.14
ta025	155.85	21.65	23.19	25.44	29.30	2.40	1.12	0.68	0.44
ta026	57.42	39.26	28.10	44.67	25.46	0.49	0.34	0.14	0.19
ta027	84.63	17.97	16.46	1.95	1.63	1.57	0.86	4.81	4.32
ta028	154.18	47.89	35.20	19.50	12.60	1.07	0.73	0.88	1.02
ta029	32.67	10.97	6.31	4.33	2.43	0.99	0.86	0.84	1.12
ta030	120.29	56.93	44.11	39.11	49.56	0.70	0.45	0.34	0.20
Prom.	68.44	27.16	22.25	16.38	14.25	0.99	0.61	0.93	0.86

Tabla 84: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 20x20

	3 a 6	3a9	3a12	6a9	6a12	9a12
ta021	1.10	0.46	0.87	0.42	1.25	1.87
ta022	0.60	0.49	0.36	0.81	0.49	0.73
ta023	0.44	2.02	1.02	4.63	0.34	0.50
ta024	0.50	0.33	0.25	0.66	0.51	0.76
ta025	0.47	0.28	0.18	0.61	0.43	0.65
ta026	0.70	0.29	0.39	0.42	0.88	1.32
ta027	0.55	3.06	2.75	5.61	0.60	0.90
ta028	0.68	0.82	0.95	1.20	0.77	1.16
ta029	0.87	0.84	1.13	0.97	0.89	1.34
ta030	0.65	0.49	0.29	0.75	0.39	0.59
Prom.	0.65	0.91	0.82	1.61	0.65	0.98

Tabla 85: Eficiencia incremental generalizada para la serie 20x20

9.7.4 Serie 50x5

	Tiempo según número de procesos					Eficiencia según número de procesos			
	2 en 1	**3**	**6**	**9**	**12**	**3**	**6**	**9**	**12**
ta031	0.75	0.43	0.43	0.44	0.43	0.58	0.29	0.19	0.15
ta032	21.24	8.59	9.78	8.59	8.61	0.82	0.36	0.27	0.21
ta033	11.02	6.47	6.43	6.16	6.39	0.57	0.29	0.20	0.14
ta034	25.64	7.81	7.81	7.81	7.81	1.09	0.55	0.36	0.27
ta035	8.94	5.36	5.32	5.36	5.36	0.56	0.28	0.19	0.14
ta036	12.16	7.06	7.06	7.12	7.10	0.57	0.29	0.19	0.14
ta037	3.92	2.17	2.17	2.17	2.17	0.60	0.30	0.20	0.15
ta038	131.37	16.40	15.78	9.10	11.62	2.67	1.39	1.60	0.94
ta039	9.77	3.62	3.64	3.64	3.64	0.90	0.45	0.30	0.22
ta040	219.09	15.51	7.11	12.71	11.78	4.71	5.14	1.92	1.55
Prom.	44.39	7.34	6.55	6.31	6.49	1.31	0.93	0.54	0.39

Tabla 86: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 20x20

	3 a 6	3a9	3a12	6a9	6a12	9a12
ta031	0.50	0.33	0.25	0.66	0.50	0.76
ta032	0.44	0.33	0.25	0.76	0.50	0.75
ta033	0.50	0.35	0.25	0.70	0.48	0.72
ta034	0.50	0.33	0.25	0.67	0.50	0.75
ta035	0.50	0.33	0.25	0.66	0.50	0.75
ta036	0.50	0.33	0.25	0.66	0.50	0.75
ta037	0.50	0.33	0.25	0.67	0.50	0.75
ta038	0.52	0.60	0.35	1.16	0.39	0.59
ta039	0.50	0.33	0.25	0.67	0.50	0.75
ta040	1.09	0.41	0.33	0.37	0.54	0.81
Prom.	0.56	0.37	0.27	0.70	0.49	0.74

Tabla 87: Eficiencia incremental generalizada para la serie 50x5

9.7.5 Serie 50x10

	Tiempo según número de procesos					Eficiencia según número de procesos			
	2 en 1	3	6	9	12	3	6	9	12
ta041	285.49	37.76	86.77	12.14	23.28	2.52	0.55	2.61	1.02
ta042	69.26	107.43	9.91	13.64	9.20	0.21	1.16	0.56	0.63
ta043	266.83	49.79	49.88	26.29	48.74	1.79	0.89	1.13	0.46
ta044	45.20	18.48	7.38	4.07	4.47	0.82	1.02	1.23	0.84
ta045	48.64	15.83	18.94	13.99	17.29	1.02	0.43	0.39	0.23
ta046	354.87	6.03	6.02	6.02	6.02	19.61	9.82	6.55	4.91
ta047	68.62	728.65	693.83	37.11	31.39	0.03	0.02	0.21	0.18
ta048	1337.36	175.55	46.60	77.88	61.83	2.54	4.78	1.91	1.80
ta049	15.46	9.13	6.42	3.93	3.28	0.56	0.40	0.44	0.39
ta050	45.29	64.79	18.77	15.25	14.64	0.23	0.40	0.33	0.26
Prom.	253.70	121.34	94.45	21.03	22.02	2.93	1.95	1.53	1.07

Tabla 88: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 50x10

	3 a 6	3a9	3a12	6a9	6a12	9a12
ta041	0.22	1.04	0.41	4.76	0.26	0.39
ta042	5.42	2.63	2.92	0.48	0.74	1.11
ta043	0.50	0.63	0.26	1.26	0.27	0.40
ta044	1.25	1.51	1.03	1.21	0.46	0.68
ta045	0.42	0.38	0.23	0.90	0.40	0.61
ta046	0.50	0.33	0.25	0.67	0.50	0.75
ta047	0.53	6.55	5.80	12.47	0.59	0.89
ta048	1.88	0.75	0.71	0.40	0.63	0.94
ta049	0.71	0.77	0.70	1.09	0.60	0.90
ta050	1.73	1.42	1.11	0.82	0.52	0.78
Prom.	1.32	1.60	1.34	2.41	0.50	0.75

Tabla 89: Eficiencia incremental generalizada para la serie 50x10

9.7.6 Serie 50x20

	Tiempo según número de procesos					Eficiencia según número de procesos			
	2 en 1	3	6	9	12	3	6	9	12
ta051	4048.87	2413.21	1716.01	493.85	1056.39	0.56	0.39	0.91	0.32
ta052	798.90	453.68	165.40	208.87	126.71	0.59	0.81	0.43	0.53
ta053	315.53	484.47	110.49	144.86	121.32	0.22	0.48	0.24	0.22
ta054	1498.70	24774.29	31932.12	12434.48	7261.14	0.02	0.01	0.01	0.02
ta055	180.03	222.28	155.09	50.61	85.58	0.27	0.19	0.40	0.18
ta056	260.50	123.73	85.16	62.04	22.32	0.70	0.51	0.47	0.97
ta057	254.37	173.23	247.14	159.58	142.51	0.49	0.17	0.18	0.15
ta058	519.98	147.03	72.20	147.47	72.57	1.18	1.20	0.39	0.60
ta059	435.16	161.80	161.69	160.60	163.25	0.90	0.45	0.30	0.22
ta060	38.53	6.29	15.51	6.28	6.66	2.04	0.41	0.68	0.48
Prom.	835.06	2896.00	3466.08	1386.86	905.84	0.70	0.46	0.40	0.37

Tabla 90: Tiempo de ejecución y eficiencia de 3 a 12 procesos para la serie de 50x20

	3 a 6	3a9	3a12	6a9	6a12	9a12
ta041	0.70	1.63	0.57	2.32	0.23	0.35
ta042	1.37	0.72	0.90	0.53	0.82	1.24
ta043	2.19	1.11	1.00	0.51	0.60	0.90
ta044	0.39	0.66	0.85	1.71	0.86	1.28
ta045	0.72	1.46	0.65	2.04	0.30	0.44
ta046	0.73	0.66	1.39	0.92	1.39	2.08
ta047	0.35	0.36	0.30	1.03	0.56	0.84
ta048	1.02	0.33	0.51	0.33	1.02	1.52
ta049	0.50	0.34	0.25	0.67	0.49	0.74
ta050	0.20	0.33	0.24	1.65	0.47	0.71
Prom.	0.82	0.76	0.66	1.17	0.67	1.01

Tabla 91: Eficiencia incremental generalizada para la serie 50x20