VIII Congreso de Ingeniería de Organización
Leganés, 9 y 10 de septiembre de 2004

Implementación de meta-heurísticas en paralelo mediante LAM-MPI: Situación actual y perspectivas de futuro*

José Miguel León Blanco, José Manuel Framiñán Torres, Pedro Luis González Rodríguez, Rafael Ruiz-Usano

Departamento de Organización Industrial y Gestión de Empresas. Escuela Superior de Ingenieros. Universidad de Sevilla. Camino de los Descubrimientos, S/N. 41092 Sevilla. miguel@esi.us.es, jose@esi.us.es, pedroluis@esi.us.es, usano@us.es.

Resumen

La implementación de métodos aproximados para obtener soluciones cercanas al óptimo en problemas NP-completos es un área de gran interés en la ingeniería de organización. La resolución en paralelo de estos métodos mediante computación presenta ventajas, no solo en la reducción del tiempo necesario para obtener soluciones de calidad sino en la misma calidad de las soluciones. Esta se ve mejorada debido a la exploración más exhaustiva del espacio de soluciones. La implementación en paralelo en redes de ordenadores de bajo coste mediante técnicas de paso de mensajes es una de las líneas más prometedoras en la resolución práctica de este tipo de problemas.

Palabras clave: Procesamiento paralelo, MPI, LAM, metaheurísticas

1. Introducción

La implementación de métodos aproximados o heurísticas que permiten obtener soluciones cercanas al óptimo para problemas de optimización combinatoria NP-completos es un área de trabajo de gran interés en la Ingeniería de Organización. Existen gran número de investigaciones en este campo, y muchas de ellas se orientan a la resolución en paralelo de estos métodos aproximados. En este trabajo se presenta una breve revisión de las tendencias tanto actuales como futuras en la materia. Además se realizan una serie de implementaciones mediante la interfaz de paso de mensajes (MPI).

2. Metaheurísticas y procesamiento paralelo

Entre los métodos más conocidos de resolución aproximada de problemas de optimización combinatoria, destacan las metaheurísticas, en las que una solución inicial, típicamente ofrecida por alguna heurística, es mejorada mediante la exploración de las soluciones cercanas (ver, por ejemplo Díaz *et al.* 1996 para una descripción de metaheurísticas).

* Este trabajo se deriva de la participación de sus autores en un proyecto de investigación financiado por CICYT con referencia DPI-2001-3110 titulado "Sistemas Híbridos para un control Integrado de la Producción".

2.1. Mejoras en el tiempo de ejecución

El aumento exponencial del número de soluciones posibles con la dimensión de las soluciones en los problemas NP-Completos ha llevado a que la primera preocupación en la búsqueda de soluciones aproximadas sea la reducción del tiempo de computación. Esta reducción de tiempo será proporcional de forma lineal al número de procesadores empleados una vez se haya refinado lo suficiente el algoritmo de búsqueda en paralelo.

$$aceleración(p) = \frac{\text{tiempo\_algoritmo\_secuencial}}{\text{tiempo\_algoritmo\_paralelo(p)}} \quad (1)$$

Otros autores, Cung, V.D. *et al.* (2001), hablan de eficiencia ρ como el ratio entre la aceleración y el número de procesadores:

$$\rho = \frac{aceleración(p)}{p} \quad (2)$$

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

2.2. Mejoras en la robustez del algoritmo

La segunda observación sobre la resolución en paralelo es que esta permite para un tiempo similar al del algoritmo secuencial, explorar de forma mucho más exhaustiva el espacio de soluciones, con lo que se obtiene una mejora en la calidad de las soluciones obtenidas. La mejora en robustez del algoritmo paralelo frente al algoritmo secuencial se refiere a la capacidad de encontrar soluciones de calidad ante diferentes instancias del mismo problema, ya que la resolución mediante meta-heurísticas, es muy sensible a cada problema concreto.

Existen numerosas aplicaciones en las que se demuestra que la resolución en paralelo de problemas mediante metaheurísticas no solo mejora los tiempos de resolución, sino que mejora la robustez de las soluciones obtenidas. (Cung *et al.* (2001), Crainic *et al.* (2002))

3. Estrategias de resolución en paralelo

3.1. Algoritmos de pasada única o de múltiples pasadas

La resolución en paralelo de problemas mediante metaheurísticas puede seguir varias estrategias. Estas pueden clasificarse atendiendo a diferentes criterios, por ejemplo, la estrategia de exploración de soluciones por cada uno de los procesos. Los algoritmos de pasada única o *single walk*, realizan una sola exploración en cada procesador, devolviendo la mejor solución encontrada. En el caso de que cada uno de los procesos explore varios caminos a partir de soluciones que no tienen porqué coincidir de una pasada a la siguiente, se habla de paralelismo de múltiples pasadas o *multiple walk*.

3.2. SIMD y MIMD

Son dos tipos de arquitectura para la resolución en paralelo que se diferencian en la forma en que reparte el trabajo a los procesadores. SIMD (Single Instruction Stream, Multiple Data Stream) carga 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 cada una por un proceso distinto.

La arquitectura MIMD (Multiple Instruction Stream, Multiple Data Stream) carga conjuntos de instrucciones diferentes en cada procesador. Cuando los cálculos que se pretende realizar son complejos, puede ser adecuado dividir las tareas en segmentos diferentes que se ejecutan por separado. Se conoce también como paralelismo funcional, que trata de explorar cada parte del espacio de soluciones con diferentes estrategias. Por un lado, se mejora enormemente la calidad de las soluciones por emplear no solo diferentes puntos de partida en la exploración sino diferentes métodos. Por otro lado, se dificulta en gran medida la observación de la mejora frente al algoritmo secuencial, dado que las características de la exploración mediante uno u otro método son muy diferentes.

3.3. Paralelismo síncrono y asíncrono

Cuando existen secciones de código que se ejecutan en paralelo, suele ser ventajoso que los diferentes procesos o hilos no trabajen de forma independiente, sino que intercambien información. Esta información puede ser en el caso de los problemas que nos ocupan, por ejemplo, sobre 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 primer caso, los procesos realizan los intercambios a intervalos de tiempo prefijados, mientras que en el segundo, se producen cuando se dan ciertas condiciones. El paralelismo síncrono es más adecuado para arquitecturas y procesos homogéneos. En caso contrario, es más interesante la comprobación de condiciones para el intercambio.

De forma esquemática, en la figura 1 pueden distinguirse los diferentes tipos de hilos de ejecución citados por Crainic y Toulouse (2002) de la siguiente forma:

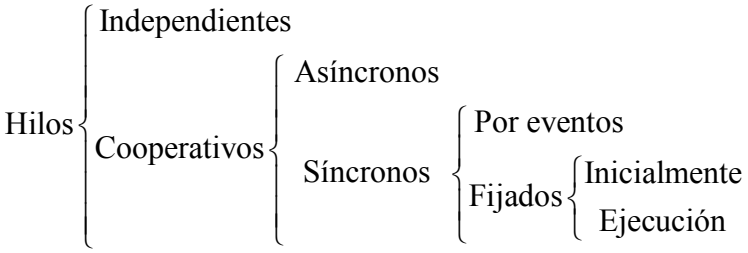

Figura 1. Tipos de hilos de ejecución en paralelo

3.4. Otras clasificaciones

Se han realizado otras muchas clasificaciones de los algoritmos de resolución en paralelo Cung *et al.* (2001) Crainic *et al.* (2002). En algunas se atiende a la relación entre el tiempo de

computación y el tiempo de comunicación, para distinguir entre algoritmos de grano fino o de grano grueso (mayor tiempo de computación). En otras, se tiene en cuenta el modelo distribuido o maestro/esclavo. Otra de las clasificaciones atiende a la arquitectura física de la plataforma de computación. Así se distinguen entre memoria centralizada y distribuida.

3.5. Consideraciones sobre las estrategias de programación en paralelo

La división del espacio de soluciones puede acarrear dificultades añadidas a la implementación en paralelo, ya que debe generarse alguna métrica para distinguir unas partes de otras en el 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. Sobre todo complejo en el caso de procesos independientes, donde deberán diseñarse de forma más cuidadosa las estrategias de exploración.

También se ha mencionado antes la complejidad de una comparación directa entre el algoritmo secuencial y el paralelo. En este caso, la comparación se realiza atendiendo a la bondad de las soluciones según el valor de la función objetivo para instancias diferentes del mismo problema.

4. Plataformas paralelas de computación

Existen gran variedad de estrategias para implementar programas en paralelo. Entre ellas se distinguen varias arquitecturas hardware y dentro de ellas distintas herramientas software.

4.1. Arquitecturas hardware

Dentro de los primeros, se pueden emplear costosos sistemas multiprocesador, disponibles por los diferentes fabricantes de equipos como por ejemplo, los sistemas 9000 de HP, Fire de Sun o iSeries de IBM. Otra posibilidad es emplear clústeres de ordenadores bien de bajo costo o bien soluciones proporcionadas por los principales fabricantes de hardware como los sistemas Blade de IBM o de Sun. Dentro de los primeros existen dos tendencias principales, Mosix y Beowulf. Son los dos paradigmas de computación distribuida (Mosix) y paralela (Beouwlf). En Mosix, el paralelismo es más difuso, ya que es el sistema operativo el encargado de repartir las tareas, mientras que en un cluster Beowulf, las tareas deben separarse en la programación de las mismas. Otras posibilidades como las arquitecturas masivamente paralelas o redes de transputers son poco aplicables a la empresa actual, bien por su coste o bien por su escasa difusión actual. Un tema cercano a la computación paralela y con bastante interés en la actualidad es la computación distribuida, en la que no entramos en este estudio.

4.2. Soluciones software

En cuanto a soluciones software, se encuentran diversas plataformas que pueden hacer uso de las arquitecturas hardware citadas. Así, se puede emplear programación multihilo, empleando lenguajes orientados a objetos como Java o C++, que proporcionan métodos y funciones para facilitar la tarea de subdividir la ejecución en tareas más o menos independientes. PVM o Parallel Virtual Machine es una biblioteca para paso de mensajes entre procesos, que permite

emplear nodos heterogéneos en la comunicación. Siguiendo esta misma filosofía, se desarrolló la Interfaz de Paso de Mensajes o MPI.

5. Interfaz de paso de mensajes (MPI)

Esta es una de las posibilidades de implementación 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 añaden 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 cluster.

Entre sus prestaciones, aparte de las normales de envío y recepción de mensajes entre procesos, dispone de características adicionales para difusión de mensajes o *broadcast*, para realizar operaciones matemáticas y estadísticas con el contenido de los mensajes, para detener o reiniciar procesos entre otras. En todas estas operaciones, 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.

Existen varias implementaciones de MPI, las más conocidas, MPICH y LAM.

6. Multiordenador de área local (LAM)

Una de las implementaciones más extendidas de MPI es LAM (Local Area Multicomputer www.lam-mpi.org), 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. Actualmente puede obtenerse, además de en su página web, como parte de las distribuciones de Linux, por ejemplo, Red Hat. Aunque existen otras implementaciones de MPI, como MPICH, en nuestro caso, nos encontramos trabajando en la implementación mediante LAM. La disponibilidad de implementaciones de las librerías MPI y la facilidad de su puesta en funcionamiento son factores que inclinan la decisión hacia esta plataforma.

Una de las dificultades de emplear la arquitectura MPI es el esquema de memoria que se emplea. El esquema de memoria que puede emplearse es únicamente el de memoria distribuida, sin beneficiarse por tanto 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 dificultades. Una forma de que todos los procesos accedan a toda la memoria del sistema, es que cada uno emplee parte del tiempo en procesamiento y parte en hacer disponible la memoria a 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, pueden dedicarse una parte de los procesos a servicio de memoria y otra parte a cálculos. Por estas dificultades, el tipo de paralelismo en MPI sigue el esquema de memoria distribuida.

7. Aplicación a la resolución de problemas de secuenciación

Uno de los problemas clásicos en la secuenciación de trabajos es el de minimización del tiempo máximo de terminación (makespan) en un entorno de flujo uniforme sin restricciones de capacidad. Este problema se reduce, bajo la hipótesis de no adelantamiento de los trabajos, en procesar n trabajos en el mismo orden en m máquinas. Este problema es NP-completo para un número de máquinas mayor de 2, 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.

En esta comunicación, emplearemos un esquema maestro/esclavo para implementar la búsqueda de soluciones en este problema mediante la paralelización de un algoritmo de búsqueda local. En la tabla 1 se presenta un pseudocódigo muy resumido de la aplicación. Los procesos esclavos, al menos en esta versión de la aplicación, funcionan como hilos independientes, y sólo intercambian información a través de un proceso principal o maestro. Este proceso mantiene una lista con 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.

Tabla 1. Esquema básico de la aplicación para búsqueda en paralelo

Principal		
Lectura de datos del problema		
Maestro		**Esclavos**
Para un nº de iteraciones		
Genera estrategia		
Genera solución		
Envía solución + estrategia	→	Recibe solución inicial + estrategia
		Ejecuta búsqueda local con memoria
Recibe las B mejores soluciones encontradas por i	←	Devuelve las B mejores soluciones
Tareas finales		

El algoritmo empleado es una versión paralela de búsqueda local completa con memoria, Ghosh y Sierksma (2002). En este algoritmo, cada proceso mantiene tres listas de soluciones, LIVE, DEAD y NEWGEN. Partiendo de una solución inicial, enviada por el proceso maestro, realizan una búsqueda local de todas las soluciones que superan un umbral también enviado por el proceso maestro. Según un parámetro, se toma una solución de la lista LIVE, se exploran todas sus vecinas, y las que pasan el umbral se guardan en NEWGEN. Una vez encontradas todas las vecinas de la solución inicial, hasta el tope marcado por el tamaño de la memoria, pasan a la lista LIVE. El proceso se repite de forma iterativa hasta que se cumpla alguna condición de parada. En ese momento, se realiza un postproceso de las soluciones que quedan en la lista LIVE, y pasa la mejor solución a la lista DEAD. La condición de parada que se está empleando es el tiempo que pasa el proceso sin obtener mejora. El proceso reenvía la mejor solución de la lista DEAD junto con el valor del C_{max} al proceso maestro, para obtener una nueva solución de partida y comenzar así la búsqueda. Los parámetros como el tamaño de la memoria, el valor del umbral, el número de soluciones que se toman de LIVE para cada exploración o el tiempo máximo de búsqueda sin mejora son enviados por el proceso maestro en cada iteración, debidamente corregidas.

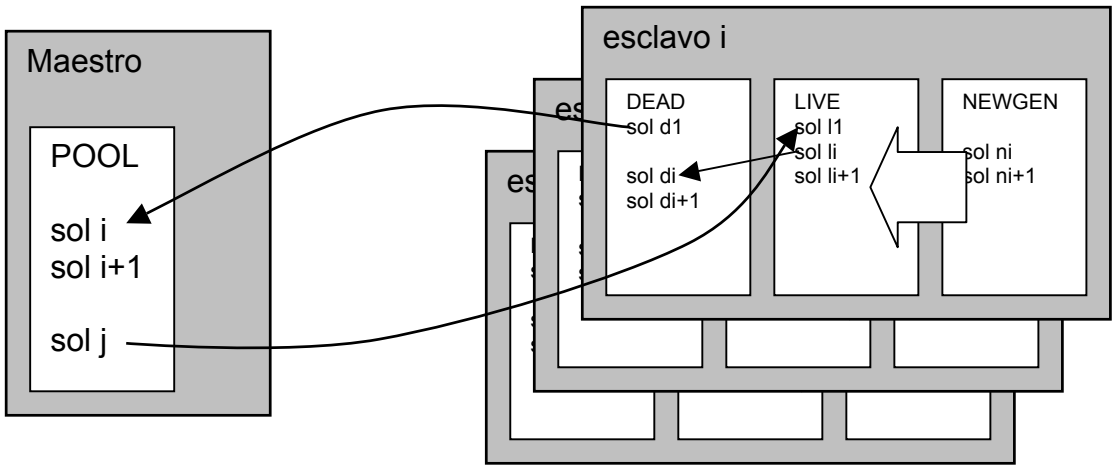

Figura 2. Esquema de funcionamiento en paralelo

El proceso maestro tiene la responsabilidad, no solo de mantener una memoria central de soluciones, POOL, sino de recibir las enviadas por los esclavos, procesarlas y reenviarles nuevas soluciones hasta alcanzar una condición de parada. Las nuevas soluciones se envían según el valor de makespan alcanzado y la proximidad a otras soluciones. Esta estrategia se basa en el trabajo de Crainic y Gendreau (2002). Si la solución recibida por el proceso maestro es la mejor hasta el momento, se la reenvía al mismo proceso para que inicie una búsqueda a partir de ahí. Si la solución iguala a la mejor hasta el momento, se le reenvía la misma si el número de iteraciones sin mejora no supera un valor. Si pasa de este valor, la solución enviada se toma al azar de entre las que superan un umbral dado. Esta estrategia se basa en la estrategia descrita en Reeves (1999). En este trabajo se expone el hecho de que las soluciones en problemas de optimización combinatoria como el que nos ocupa, siguen una topología similar a grandes valles, en los que los óptimos locales pueden conducir al óptimo global. Todo esto depende del operador de generación de soluciones vecinas y la métrica con la que se calculan las distancias entre las distintas soluciones.

Si las soluciones devueltas por los procesos esclavos están cercanas según el valor de makespan al mejor valor global hasta la fecha, se les devuelve la mejor solución encontrada por ese proceso concreto. En caso de no haber obtenido mejora en varias iteraciones, se le devolverá una solución aleatoria de entre las más prometedoras encontradas hasta la fecha.

7.2. Implementación

En la actualidad, hemos implementado la resolución de este problema mediante una red Fast Ethernet con 7 PCs, con la versión 7.0.2 de LAM sobre la distribución Red Hat 9 de Linux.

7.2.1. Notas sobre la compilación de LAM

Para la comunicación ha debido compilarse LAM para emplear el protocolo SSH v.2 para la autentificación de los nodos en la red. Dadas las condiciones de inseguridad actuales en Internet y que nuestra red es accesible actualmente desde el exterior, se ha optado, además de por asignar direcciones IP privadas a cada uno de los nodos, emplear la versión 2 del protocolo SSH, mucho más segura que emplear telnet o RSH.

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 nos permitirá monitorizar en un

futuro 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.

7.3. Experimentación y resultados

Para comprobar tanto la corrección como la bondad del algoritmo seleccionado, se han comparado los valores obtenidos con el algoritmo paralelo con los obtenidos para una batería de problemas bien conocida como es la Taillard, (1993), de la que se dispone, para diferentes configuraciones del problema de secuenciación de trabajos, de los valores más cercanos al óptimo obtenidos hasta la fecha por diferentes equipos de investigadores. De esta forma se comparan tiempos de ejecución para distinto número de procesadores y cercanía al óptimo. Así se tienen las dos medidas fundamentales de la bondad de estos algoritmos como son la aceleración con el número de procesadores y la robustez ante distintas configuraciones del problema.

8. Comentarios y líneas futuras de investigación

Deben perfeccionarse numerosos aspectos del algoritmo presentado. Entre ellos cabe citar los siguientes:

- La cooperación entre procesos se encuentra en una fase muy primitiva, ya que únicamente se produce mediante el intercambio de soluciones con el proceso maestro y no mediante el intercambio directo de soluciones entre procesos esclavos. En la aplicación actual debe emplearse paralelismo de grano grueso o *coarse grained* debido a la lentitud de la red de comunicaciones, luego sería deseable su mejora para poder aumentar la comunicación entre procesos sin restar eficiencia a la paralelización.

- El riesgo de estancamiento en mínimos locales se ha manifestado a lo largo de toda la implementación de este algoritmo, con lo que deben buscarse alternativas que permitan generar soluciones para salir de dichos mínimos y evolucionar favorablemente. Entre estas mejoras se encuentra la modificación de las estrategias de búsqueda de los procesos no solo mediante distintos parámetros sino mediante distintos algoritmos. La implementación mediante LAM permitirá hacerlo sin demasiadas dificultades.

- Se ha trabajado todavía poco en el campo de la sincronización de los procesos, existiendo tiempos muertos en los mismos y procesos saturados. Del mismo modo, tampoco se ha optimizado el flujo de información por la red, lo que sin duda resultará de gran utilidad cuando se emplee el algoritmo en problemas de mayores dimensiones que los actuales.

Referencias

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.
Díaz, A.; Glover, F.; Hassan, M. G.; González, J. L.; Laguna, M.; Moscazo, P.; Tseng, F. T. (1996). *Optimización heurística y redes neuronales.* Paraninfo.
Crainic, T.G. ; Gendreau, M. (2002). Cooperative Parallel Tabu Search for Capacitated Network Design. *Journal of Heuristics*, No. 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.
Cung, V.D.; Martins, S.; Ribeiro, C.; Roucariol, C. (2001) *Strategies for the parallel implementation of Metaheuristics. Essays and Surveys in Metaheuristics*. C.C. Ribeiro and P.
Ghosh, D.; Sierksma, G.; (2002). Complete Local Search with Memory. *Journal of Heuristics*, No. 8, pp. 571-584. Kluwer.
Hansen, editors), Kluwer Academic Publishers.
Gropp, W.; Lusk, E.; Skjellum, A. (1999) *Using MPI, Portable Parallel Programming with the Message-Passing Interface*. 2nd ed. The MIT Press.
LAM-MPI Parallel Computing. www.lam-mpi.org
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.
Reeves, C.R. (1999). Landscapes, operators and heuristic search. *Annals of Operations Research* No. 86*,* pp. 473-490. J.C.Baltzer AG, Science Publishers.
Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, Vol. 64, pp. 278-285.