Particle Swarm Optimization para un Problema de Optimización Combinatoria

Leticia Cecilia Cagnina y Susana Cecilia Esquivel

Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC)[1].
Universidad Nacional de San Luis
Ejército de los Andes 950, D5700HHW - San Luis - Argentina
Tel: +54 (2652) 426747 - Fax: +54 (2652) 430224

{esquivel, lcagnina}@unsl.edu.ar

Workshop de Agentes y Sistemas Inteligentes

Resumen

En este artículo se presenta una versión del algoritmo de *Particle Swarm Optimization* que ha sido hibridizado con un operador dinámico de mutación y que implementa el modelo conocido como *local best* (l-best). El algoritmo se aplica al problema de *scheduling* de máquina única siendo la función objetivo a optimizar la de *Total Weighted Tardiness*.
El algoritmo propuesto es validado usando instancias tomadas de la *OR-Library* y los resultados son comparados con los obtenidos por un algoritmo evolutivo multirecombinado que incluye conocimiento acerca del problema y con otra versión de un algoritmo *Particle Swarm Optimization* que implementa el modelo *global best* (g-best) cuyos resultados han sido reportados en publicaciones recientes.
Los resultados obtenidos son muy promisorios, sobre todo si se considera que este paradigma casi no ha sido utilizado para problemas de optimización combinatoria.

Palabras Claves: *Particle Swarm*, Optimización, Técnicas de Vecindarios, *Scheduling*

[1] El LIDIC es financiado por la Universidad Nacional de San Luis y la ANPCYT (Agencia Nacional para la Promoción de la Ciencia y la Tecnología).

1. Introducción

El algoritmo *Particle Swarm Optimization* (PSO) fue propuesto por James Kennedy y Rusell Eberhart en 1995 [2, 5] como una heurística de búsqueda poblacional basada en el comportamiento social y cognitivo de ciertas especies. En ella cada partícula representa una solución potencial dentro del espacio de búsqueda y está caracterizada por una posición, una velocidad y una memoria de su comportamiento pasado.
En cada ciclo de vuelo, se evalúa la función objetivo para cada partícula en su posición actual. El valor obtenido mide la calidad de la partícula para el problema que se esté considerando.

En el PSO original las partículas vuelan a través del espacio de búsqueda influenciadas por dos factores: a) la mejor posición que la partícula alcanzó hasta ese momento (según está registrada en su memoria) y b) la mejor posición alcanzada por cualquier partícula de la población o *swarm*.

La actualización de la posición de cada partícula se realiza a través de las siguientes dos ecuaciones:

$$vel_{i,j} = vel_{i,j} + c_1 * r_1 * (p_{i,j} - part_{id}) + c_2 * r_2 * (p_{g,j} - part_{id}) \quad (1)$$

$$part_{id} = part_{i,j} + vel_{i,j} \quad (2)$$

donde $vel_{i,j}$ es la velocidad de la partícula i en la j-ésima dimensión, c_1 y c_2 son pesos, conocidos con el nombre de factores de aprendizaje individual y social, respectivamente aplicados a la influencia de las mejores posiciones obtenidas hasta el momento por la partícula i y la mejor partícula del *swarm* g; r_1 y r_2 son valores aleatorios distribuidos uniformemente en el intervalo [0,1].
Luego que la velocidad se actualiza usando la ecuación (1) se calcula la nueva posición de la partícula i en su dimensión j-ésima a través de la ecuación (2). Este proceso se repite para cada dimensión de la partícula i y para todas las partículas de la población.

En un trabajo posterior Shi y Eberhart [13] introdujeron el concepto de factor de inercia, w, cuya finalidad es controlar la cantidad de la velocidad previa de la partícula a ser mantenida en el ciclo actual. En consecuencia, la ecuación (1) se reformula como:

$$vel_{i,j} = w * vel_{i,j} + c_1 * r_1 * (p_{i,j} - part_{id}) + c_2 * r_2 * (p_{g,j} - part_{id}) \quad (3)$$

En trabajos posteriores, Kennedy y Eberhart [2, 3, 4, 19] estudiaron los efectos de la inclusión de vecindarios en el PSO. En el presente trabajo sólo se incluye la técnica de vecindario que considera al vecindario en términos de los índices de las partículas, de esta manera cada partícula es afectada por la mejor performance de sus k vecinos inmediatos, siendo $k + 1$ el tamaño del vecindario. En este modelo la ecuación (3) se expresa como sigue:

$$vel_{i,j} = w * vel_{i,j} + c_1 * r_1 * (p_{i,j} - part_{id}) + c_2 * r_2 * (p_{l,j} - part_{id}) \quad (4)$$

donde $p_{l,j}$ representa a la mejor partícula en el vecindario de la partícula i en lugar de la mejor partícula global de todo el *swarm*.

Recientemente otros autores han propuesto una hibridación de PSO con conceptos de Algoritmos Evolutivos [17]. Algún tiempo antes, Angeline propuso un algoritmo híbrido que usa el PSO original combinado con el concepto de selección [18]. El algoritmo aquí propuesto hibridiza al PSO con un operador dinámico de mutación.

La principal meta del presente trabajo es evaluar la *performance* del algoritmo propuesto cuando se aplica a la resolución de un problema de *scheduling* de máquina única. Para esto, se usaron *benchmarks* bien conocidos y los resultados son comparados con un algoritmo evolutivo multirecombinado con inclusión de conocimiento del problema (MCMP-SRI-IN) [11] y un algoritmo híbrido (HPSO) basado en el *global model* (g-best) [1].

El resto del artículo está organizado como sigue. La sección 2 describe brevemente el problema de *scheduling* tratado. La sección 3 describe los lineamientos generales del algoritmo propuesto y de los algoritmos utilizados para comparación. La descripción experimental se realiza en la sección 4. Los resultados se muestran y discuten en la sección 5. En la sección 6 se presentan las conclusiones y se bosquejan algunas de las líneas de investigación futuras.

2. El Problema de Scheduling de Total Weighted Tardiness

McNaughton [6] presentó por primera vez un problema de *scheduling* cuyo objetivo era el de minimizar el costo total de penalización por demoras con respecto a la fecha de terminación de los *jobs*. También probó que existe una solución óptima sin la necesidad de dividir el procesamiento de cada *job*, por lo tanto sólo se necesitan considerar *schedules* de permutaciones de la misma cantidad de *jobs* que el problema original.
De esta manera, el problema de *scheduling* de *Total Weighted Tardiness* (TWT) para máquina única [12] puede plantearse así: *n jobs* se procesan sin interrupción en una única máquina. Ésta sólo puede procesar un único *job* a la vez. Todo *job j* (j = 1, ..., n) está disponible para su procesamiento en tiempo *0*, requiere un tiempo de procesamiento p_j, tiene un peso positivo w_j, y un tiempo de entrega d_j en el cual, idealmente, el *job* debería finalizar. Para un orden de procesamiento de *jobs* establecido, el tiempo de completitud C_j y el *tardiness* $T_j = \max(C_j - d_j, 0)$ del *job j* puede calcularse fácilmente. El problema radica en encontrar el orden de procesamiento de los *jobs* con el mínimo *total weighted tardiness*

$$\sum_{j=1}^{n} w_j T_j$$

Aunque su formulación es simple, este modelo se traduce en un problema de optimización que es NP-Hard [12].

3. Descripción de Algoritmos

En su concepción original, PSO trabaja eficientemente con problemas de optimización en espacios continuos de búsqueda. Así como la versión previa de PSO híbrido, HPSO, el nuevo algoritmo $HPSO_{vecin}$ trata de preservar tal eficiencia en los problemas de secuenciamiento. Por tal razón, para trabajar con problemas de permutaciones, se seleccionó la representación *Random Key* propuesta por Bean [14]. Esta representación codifica una partícula en números reales; dichos valores son usados como claves ordenadas para codificar la solución. En el problema de *scheduling* elegido, cada componente de la partícula o dimensión se corresponde con un *job*. Estas componentes son generadas aleatoriamente al inicio del algoritmo con valores pertenecientes al

rango [0,1]. La transformación al espacio de soluciones se produce al establecer un orden ascendente sobre dichos valores. Por ejemplo, para un problema de 8 *jobs*, la partícula:

	0.22	0.09	0.99	0.97	0.35	0.57	0.18	0.85
Jobs	1	2	3	4	5	6	7	8

representa al *schedule*: 2 7 1 5 6 8 4 3. Esta secuencia luego es evaluada para determinar el valor de la función objetivo.

Es importante remarcar que debido a la redundancia de la representación elegida, muchos vectores *random keys* pueden conducir a la misma secuencia de *jobs*, por tal motivo se adoptó un operador de mutación dinámica para mantener la diversidad en la población. Este operador se aplica para cambiar una partícula con una probabilidad p_m, la cual depende del número total de ciclos de vuelo y del ciclo corriente:

$$pm = max\_pm - \frac{max\_pm - min\_pm}{max\_cycle} * current\_cycles$$

De esta forma la mutación se aplicará más frecuentemente al comienzo del proceso de búsqueda y decaerá a medida que el número de ciclos aumenta. La partícula sólo se modifica si el valor objetivo de la partícula mutada es mejor que el valor anterior. Las características arriba detalladas son compartidas por ambos algoritmos HPSO y $HPSO_{vecin}$.

Un aspecto importante, que es el fundamento para proponer el algoritmo $HPSO_{vecin}$, es la convergencia prematura del algoritmo HPSO. En un trabajo previo [1] se observó que HPSO puede converger a un óptimo local, provocando el estancamiento del algoritmo para algunas instancias difíciles.
A través de la utilización de vecindarios, se buscó reducir la velocidad de convergencia y el estancamiento en óptimos locales, pudiendo así encontrar en la mayoría de los casos, los óptimos globales.

En esta nueva versión del algoritmo, cada partícula además de estar influenciada por el mejor valor encontrado por ella misma, es influenciada por el mejor valor alcanzado por alguno de sus vecinos. Ésto produce que cada partícula sea afectada por la mejor *performance* de un grupo más pequeño de partículas, y no por la mejor *performance* de la mejor partícula de la población como sucedía en HPSO. Teniendo esto presente ambos algoritmos difieren en la forma en que actualizan la velocidad de cada partícula. HPSO lo hace usando la ecuación (3) mientras que $HPSO_{vecin}$ usa la ecuación (4).

Finalmente, a los efectos de comparar la *performance* de $HPSO_{vecin}$ se trabajó también con un algoritmo evolutivo multirecombinado (es decir que realiza múltiples operaciones de *crossover* sobre múltiples padres) al cual se le incorporan *random* inmigrantes y conocimiento del problema por medio de la inclusión, en la población inicial, de semillas generadas con otras heurísticas, tal algoritmo es conocido como MCMP-SRI-IN y ha mostrado una muy buena *performance* sobre el tipo de problema en estudio, para un análisis más detallado de la operación de este algoritmo consultar [11].

4. Descripción de Experimentos

Los algoritmos trabajaron con diez instancias de problemas de 40 *jobs* y con diez instancias de problemas de 50 *jobs* extraídos de la *OR-Library* [15]. En todos los experimentos se realizaron 30 corridas para cada instancia.

Parámetros de MCMP-SRI-IN

El algoritmo evolutivo se ejecutó en todas las corridas por 200 generaciones, el tamaño de población se fijó en 100 individuos, la probabilidad de *crossover* fue de 0.65 y la de mutación de 0.05. Para la multirecombinación el número de operaciones de *crossover* múltiple sobre cada par de padres fue de 14 y el número de padres a ser recombinados fue de 16. La población inicial se construye como el vecindario de 3 semillas aplicando el operador de inversión [11].

Parámetros comunes para los algoritmos PSO

El tamaño de la población fue fijado igual al número de *jobs*, como fue sugerido por Clerc [16]. Para el operador dinámico de mutación las probabilidades fueron: *pm_min = 0.1* y *pm_max = 0.4*.

Parámetros propios de cada algoritmo PSO

Parámetros	**HPSO**	**$HPSO_{vecin}$**
ciclos de vuelo	6000 (40 jobs) 9000 (50 jobs)	15000 (40 jobs) 30000 (50 jobs)
factor de inercia (w)	0.3	0.5
factores aprendizaje ($c_1 = c_2$)	1.3	1.5
tamaño vecindario (k)	-	4

Como fue expresado anteriormente el HPSO tiende a una convergencia prematura, en tal sentido el número de ciclos es el que requiere para obtener las mejores soluciones, aumentarlo no mejora la *performance* del algoritmo. Por otro lado, $HPSO_{vecin}$ reduce la velocidad de convergencia en consecuencia requiere de un mayor número de ciclos. El factor de inercia, los factores de aprendizaje y el tamaño de vecindario fueron determinados experimentalmente como los más aptos luego de un conjunto de corridas previas.

Para evaluar la *performance* de los algoritmos se definieron las siguientes variables:

- **Mejor**: indica el mejor valor encontrado para la instancia.
- **MEv** o **Mean Evaluation**: número medio de evaluaciones necesarias para encontrar el *upper_bound* o *benchmark* (mejor valor conocido) de cada instancia.
- **HR** o **Hit Radio**: denota el porcentaje de corridas en las cuales el algoritmo alcanza el *Benchmark*. HR será 1 si el *Benchmark* se encontró en todas las corridas, 0 si no se encontró en ninguna, o un valor entre estos dos en otro caso.

5. Resultados

A continuación se muestran los resultados obtenidos con los tres algoritmos usados en el análisis: HPSO, MCMP-SRI-IN y $HPSO_{vecin}$. La tabla 1 se corresponde con las instancias de 40 *jobs*, mientras que la tabla 2 con las de 50 *jobs*. La primera columna identifica el número de la instancia, la segunda el *upper-bound* o valor óptimo conocido aportado por la *OR-Library*. Las siguientes tres columnas muestran los mejores valores obtenidos por cada algoritmo.

Inst	**Upp. Bnd**	**HPSO**	**MCMP-SRI-IN**	**$HPSO_{vecin}$**
1	913	913	913	913
11	17465	17465	17465	17465
21	77774	77785	77774	77774
31	6575	6575	6575	6575
46	64451	64457	64451	64451
56	2099	2099	2099	2099
71	90486	90616	90486	90486
91	47683	47870	47683	47683
101	0	0	0	0
116	46770	46905	46770	46770

Tabla 1: Performance para instancias de 40 jobs.

Inst	**Upp. Bnd**	**HPSO**	**MCMP-SRI-IN**	**$HPSO_{vecin}$**
1	2134	2134	2134	2134
11	51785	51812	51785	51785
21	214546	214744	214555	214547
31	9934	9934	9934	9934
46	157505	157682	157505	157505
56	1258	1258	1258	1258
71	150580	150936	150580	150580
91	89298	90469	89448	89323
101	0	0	0	0
116	35727	36278	35727	35728

Tabla 2: Performance para instancias de 50 jobs.

La tabla 1 muestra una mejora interesante en el comportamiento de $HPSO_{vecin}$ con respecto a HPSO ya que sólo en el 50% de las instancias HPSO fue capaz de encontrar el valor óptimo conocido. El nuevo algoritmo $HPSO_{vecin}$, al igual que MCMP-SRI-IN logró alcanzar los *upper-bound* en las 10 instancias.

Un resultado similar puede ser observado en las instancias de 50 *jobs* (tabla 2). Mientras que sólo el 40% de los *upper-bounds* fueron encontrados con HPSO el porcentaje ascendió a 70% con el nuevo algoritmo $HPSO_{vecin}$ y en 2 instancias la diferencia entre el valor del *benchmark* y el encontrado por $HPSO_{vecin}$ puede considerarse no significativa (una unidad). Vale la pena destacar que si bien MCMP-SRI-IN no encuentra el óptimo conocido en solamente 2 instancias y $HPSO_{vecin}$ en 3 instancias, la diferencia entre los valores de los *benchmarks* y el mejor valor encontrado por el algoritmo es mucho mayor en MCMP-SRI-IN que en $HPSO_{vecin}$, lo cual indica que el error obtenido con este último algoritmo es menor que el del algoritmo evolutivo.

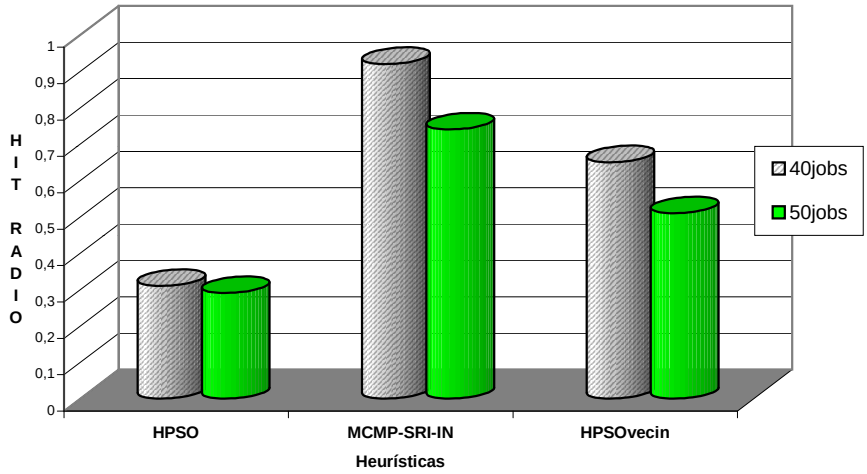

Figura 1: Evaluación de la performance con HIT RADIO.

A primera vista, si se consideran los valores obtenidos por los algoritmos, la *performance* de MCMP-SRI-IN es igual a la de $HPSO_{vecin}$ en las instancias de 40 *jobs* y muy similar en las instancias de 50. Sin embargo, se puede realizar un estudio más profundo evaluando la *performance* con las otras variables definidas en la sección anterior. La figura 1 muestra el Hit Radio (HR) promedio de las 10 instancias de 40 y las 10 de 50 *jobs* obtenidos con los tres algoritmos.
A simple vista es notable el incremento del HR obtenido por $HPSO_{vecin}$ respecto de HPSO, también puede observarse que si bien el promedio de HR de MCMP-SRI-IN es superior que el de $HPSO_{vecin}$ tanto para las instancias de 40 como para las instancias de 50 *jobs*, los valores alcanzados por este último son lo suficientemente buenos si se tiene en cuenta que $HPSO_{vecin}$ realiza una búsqueda ciega mientras que el algoritmo evolutivo incluye conocimiento del problema.

La figura 2 muestra el promedio de evaluaciones (MEv) para cada uno de los algoritmos

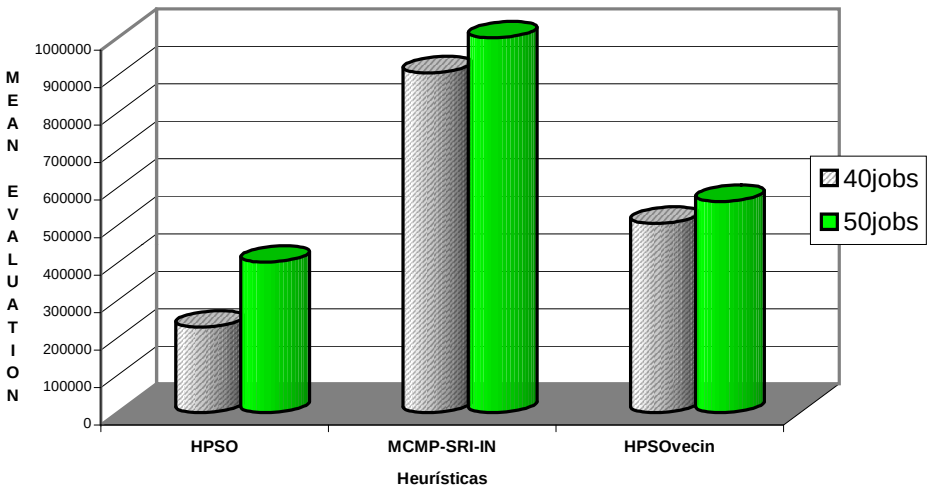

Figura 2: Evaluación de la performance con MEAN EVALUATION.

estudiados, considerando las diez instancias de 40 y las 10 de 50 *jobs*.
Con respecto a HPSO puede decirse que, en promedio, es el que menor número de evaluaciones realiza, pero es importante también destacar que para el problema de tamaño 40 sólo logra alcanzar el *upper-bound* en el 50% de las instancias mientras que para el caso de 50 *jobs* sólo lo hace en el 40% de los casos.

Más interesante resulta comparar MCMP-SRI-IN con $HPSO_{vecin}$ puesto que su *performance* es muy similar, en este caso resulta evidente que para las instancias de 40 *jobs* el promedio de evaluaciones realizadas por MCMP-SRI-IN es aproximadamente un 40% mayor que las que realiza $HPSO_{vecin}$, mientras que en el caso de 50 *jobs* MCMP-SRI-IN duplica, en promedio, la cantidad de evaluaciones realizadas.

6. Conclusiones

Diferentes heurísticas [8] incluyendo los Algoritmos Evolutivos y los de Colonia de Hormigas han sido aplicados exitosamente en la resolución de problemas duros de *scheduling* [9, 10, 11] mientras que sólo existen dos trabajos presentados simultáneamente en el mismo año [1, 7] que apliquen el paradigma de Particle Swarm Optimization a este tipo de problemas de *scheduling*.

En un trabajo previo se incursionó por primera vez en este tipo de aplicación desarrollando un algoritmo PSO hibridizado con un operador dinámico de mutación. Si bien los resultados obtenidos fueron alentadores se observó que el HPSO tendía a converger prematuramente.
En el presente trabajo se presentó un nuevo algoritmo, $HPSO_{vecin}$, que fue diseñado para mejorar la *performance* lograda por su antecesor HPSO a través de la reducción de la velocidad de convergencia.

Los algoritmos fueron aplicados a un problema de optimización combinatoria, tal es el problema de *scheduling* de Máquina Unica, con la función objetivo *Total Weighted Tardiness*.

Si bien $HPSO_{vecin}$ comparte algunas características con HPSO tales como la representación de las partículas y el operador de mutación dinámica, el primero introduce el uso de vecindarios.
Para evaluar la *performance* de $HPSO_{vecin}$, se lo comparó con HPSO y con MCMP-SRI-IN, un algoritmo evolutivo que a través de la multirecombinación balancea la exploración con la explotación guiando la búsqueda a través de la inserción de conocimiento específico.

De los resultados discutidos en la sección 5 se puede concluir que $HPSO_{vecin}$ supera a HPSO en el caso de las instancias estudiadas y es comparable en cuanto a la calidad de los resultados con MCMP-SRI-IN y lo logra con un menor costo en número de evaluaciones. Pero también se debe señalar que MCMP-SRI-IN tiene un *hit radio* superior al de $HPSO_{vecin}$ lo cual indica que encuentra los *benchmarks* en mayor cantidad de corridas lo cual se explica por la información acerca del problema que incorpora con el uso de las semillas.

Estos resultados son prometedores y en la actualidad el trabajo continúa en las siguientes direcciones: a) minimizar los posibles efectos de la redundancia que surgen en respuesta de la representación utilizada, a través del empleo de otro tipo de codificación de las partículas, b) el ajuste dinámico de los parámetros del algoritmo y c) la inclusión de información del problema en el algoritmo mediante la utilización de buenas semillas (buenas soluciones provistas por otras heurísticas) para guiar la búsqueda ciega que realiza $HPSO_{vecin}$.

7. Reconocimientos

En memoria del Dr. Raúl H. Gallard por quién sentimos el mayor respeto y gratitud.

8. Referencias

[1] Cagnina L., Esquivel S., Gallard R., "Particle Swarm Optimization for Sequencing Problems: A Case Study", Congress on Evolutionary Computation, pags 536-541, 2004, Portland, Oregon, USA.
[2] Eberhart R., Kennedy J., "A New Optimizer Using Particles Swarm Theory", 6th International Symposium on Micro Machine and Human Science, 1995, Nayoga, Japan.
[3] Peer E., van den Bergh F., Engelbrecht A., "Using neighborhoods with the guaranteed convergence pso". Swarm Intellegence Symposium (Indianapolis, Indiana), pags 235-242, Piscataway, NJ, 2003. IEEE Service Center.
[4] Eberhart R., Kennedy J., Shi Y., "Swarm Intelligence", Morgan Kaufmann Publishers, 2001, USA.
[5] Eberhart R., Kennedy J., "Particle Swarm Optimization", Proc. of IEEE Int'l Conf. on Neural Networks, 1995, Piscataway, NJ, USA, pp 1942-1948.
[6] McNaughton R., "Scheduling with deadlines and loss finctions", Management Science, vol. 6, N° 1, pp 1-12, 1959.
[7] Tasgetiren F., Sevkli M., Liang Y., Gencyilmaz G., "Particle Swarm Optimization algorithm for Single Machine Total Weighted Tardiness Problem", Congress on Evolutionary Computation, pags. 1412-1419, 2004, Portland, Oregon, USA.
[8] Morton T., Pentico D., "Heuristics Scheduling Systems", Wiley Series in Engineering and Technology Management, John Wiley and Sons, INC, 1993.
[9] Avci S., Akturk M., Storer R., "A Problem Space Algorithm for Single Machine Weighted Tardiness Problems", IIE Transactions, Vol. 35, pp. 479-486, 2003.
[10] den Besten M., Stüsle T., Dorigo M., "Ant Colony Optimization for the Total Weighted Tardiness in Paralell Problem Solving from Nature", PPSN VI International Conference, 1999.
[11] De San Pedro M., Pandolfi D., Villagra A., Lasso M., Vilanova G., Gallard R., "Adding problem-specific knowledge in Evolutionary Algorithms to Solve W-T Scheduling Problems", Proceedings del VIII Congreso Argentino de Ciencias de la Computación, pp 343-353, UBA, 2002, Argentina.
[12] Pinedo M., "Scheduling: theory, algorithms and systems", First Edition, Prentice Hall, 1995.
[13] Eberhart R., Shi Y., "A modified Particle swarm Optimizer", International Conference on Evolutionary Computation, Anchorage, AK, Piscataway, NJ, IEEE Service center, 1998.
[14] Bean J., "Genetic Algorithms and Random Keys for Sequencing
[15] OR Library, htpp://mscmga.ms.ic.ac.uk/
[16] Clerc M., "Discrete Particle Swarm Optimization. Illustred by the Traveling Salesman Problem", 2000. htpp://www.mauriceclerc.net.
[17] Higashi N. and I.H., "Particle Swarm Optimization with gaussian mutation". Swarm Intelligence Symposium (Indianapolis, Indiana), pags 72-79. Piscataway, NJ, April 2003. IEEE Service Center.
[18] Angeline P., "Using Selection to Improve Particle Swarm Optimization". International Conference on Evolutionary Computation, Piscataway, NJ, 1998. IEEE Service Center.
[19] Kennedy J., Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance". CEC'99, USA, vol 3, pags 1931-1939, Piscataway, NJ, July, 1999. IEEE Service Center.