Un algoritmo evolutivo simple para el problema de asignación de tareas a procesadores

Pablo Ezzatti
CeCal, Facultad de Ingeniería
Universidad de la República, Uruguay
pezzatti@fing.edu.uy

Sergio Nesmachnow
CeCal, Facultad de Ingeniería
Universidad de la República, Uruguay
sergion@fing.edu.uy

Resumen

Este trabajo presenta el análisis de una estrategia evolutiva de optimización basada en población para la resolución del Open Shop Scheduling, un problema de optimización combinatoria clásico que plantea la asignación de tareas a procesadores.

Sobre la propuesta original del algoritmo MOSES (Mutation or Selection Evolution Strategy) [3] aplicado al problema de asignación de tareas a procesadores, se analizan los resultados teóricos de convergencia en función de los parámetros del método y del problema. Se estudian dos variantes del algoritmo propuestas en la literatura y una tercera alternativa propuesta en este trabajo, analizando comparativamente su comportamiento y calidad de resultados.

El estudio contempla tres diferentes operadores de mutación, para los cuales se calcularon los diámetros de los grafos de exploración y se analizó empíricamente su vinculación con la calidad de resultados obtenidos sobre un conjunto de instancias de prueba.

Palabras clave: algoritmo evolutivo, MOSES, asignación de tareas a procesadores, OSSP.

Destinado al Workshop de Agentes y Sistemas Inteligentes

Introducción

Los algoritmos estocásticos de optimización buscan minimizar o maximizar una función objetivo, generando aleatoriamente soluciones en base a distribuciones de probabilidad. Esta estrategia es equivalente a la simulación de una evolución de un proceso estocástico.
Las técnicas de optimización evolutiva emulan los procesos de la selección natural sobre un conjunto de individuos con la idea de que el mejor de ellos, evaluado de acuerdo a una función de adaptación, corresponda a una buena solución aproximada del problema.

El comportamiento de los algoritmos evolutivos puede ser modelado utilizando cadenas de Markov definidas sobre el espacio de poblaciones de individuos solución del problema. La función probabilística de transición que define la cadena de Markov queda determinada por la formalización de los operadores evolutivos del algoritmo.
Los algoritmos evolutivos tradicionales que basan su funcionamiento en operadores de selección, cruzamiento y mutación, son complicados de modelar mediante cadenas de Markov. Esta dificultad ha conducido a proponer algoritmos alternativos de menor complejidad, para los cuales el estudio teórico de propiedades y condiciones de convergencia es más sencillo. El algoritmo MOSES [3] forma parte de esta clase de algoritmos evolutivos simples. Propone aplicar un mecanismo basado exclusivamente en la selección de individuos y su posterior mutación, sin tener en cuenta el operador de cruzamiento. La simplificación propuesta al mecanismo evolutivo permite obtener condiciones teóricas de convergencia genéricas, independientes de la función a optimizar.

El problema de asignación de tareas a procesadores plantea hallar una planificación que minimice el tiempo total de ejecución de un conjunto de *j* tareas sobre *p* procesadores. Cuando *j* es mayor que tres el problema es NP-difícil [7]. Al aumentar la complejidad del problema su resolución utilizando técnicas exactas se hace cada vez menos tratable, siendo la única opción realista la aplicación de heurísticas para hallar buenas soluciones aproximadas en tiempos razonables.

Este trabajo presenta el estudio de la estrategia de simulación evolutiva del algoritmo MOSES y sus variantes para la resolución de un problema de asignación de tareas a procesadores. La sección 2 presenta el algoritmo MOSES, su modelo mediante cadenas de Markov, las variantes estudiadas y sus características. La sección 3 describe el problema de optimización estudiado. La sección 4 explica la codificación utilizada para la resolución del problema y los operadores de mutación propuestos. Los resultados experimentales sobre las instancias de prueba consideradas se presentan y comparan en la sección 5. Por último, se ofrecen las conclusiones y propuestas de trabajo futuro.

1. El Algoritmo MOSES

Esta sección presenta el algoritmo MOSES y su modelo mediante cadenas de Markov, que permite calcular las propiedades teóricas de convergencia del algoritmo. Complementariamente se presentan dos variantes del algoritmo y se describen las características de sus mecanismos de búsqueda.

1.1 - Presentación

El algoritmo MOSES es un algoritmo evolutivo basado en población que utiliza solamente operadores de selección y mutación para la búsqueda de soluciones. Sus creadores, Cercueil y Francois lo enmarcan dentro del conjunto de algoritmos de simulación Monte Carlo [3].
Como diferencia fundamental con los algoritmos genéticos, otro caso de algoritmos evolutivos basados en exploración markoviana, MOSES tiene una formulación sencilla que permite deducir condiciones teóricas de convergencia, que no dependen de la función a optimizar.
El algoritmo MOSES trabaja sobre una población P de soluciones al problema (*individuos*) para optimizar una función *f*. Define una cadena de Markov X_n sobre la potencia E^k, siendo E el espacio de estados del problema y *k* la cardinalidad de P. Las transiciones de X_n están dadas por un operador de *mutación*, que define un grafo de búsqueda especificando las adyacencias entre individuos de la población y determina el modo de recorrer el espacio de estados del problema.

En cada generación se halla el mejor individuo alcanzado por el algoritmo (I_+). Para cada individuo de la población I_r se decide, de acuerdo a una probabilidad de mutación p_{MUT}, transformarlo en otro individuo I_q siguiendo una marcha al azar definida sobre el grafo de búsqueda, o transformarlo en el mejor individuo I_+. Esta estrategia introduce un grado de elitismo en el mecanismo de selección utilizado por el algoritmo MOSES. Una descripción del mecanismo evolutivo del algoritmo MOSES para un problema de minimización se ofrece en el Cuadro 1, basado en Ycart [12].

En cada generación
  Hallar el mejor individuo $I_+ = \text{Min}\ \{f(I_h),\ h = 1 \ldots k\}$
  Sortear un entero $Z \in (0, k]$ de acuerdo a una ley Binomial $(k, e^{-1/T})$
  Para individuos I_r , $r = 1 \ldots Z$
    cambiar $I_r = (i_1, i_2, \ldots, i_s)$ en $I_q = (i_1', i_2', \ldots, i_s')$
  Para individuos I_r , $r = Z+1 \ldots k$
    cambiar $I_r = (i_1, i_2, \ldots, i_s)$ en I_+

Cuadro 1: Descripción del mecanismo evolutivo del algoritmo MOSES

El algoritmo se complementa con una rutina de inicialización aleatoria para la población, usualmente diseñada tomando en cuenta las características del problema a resolver.

El movimiento entre individuos define la marcha aleatoria en el espacio de búsqueda y es crucial para el resultado teórico de convergencia del algoritmo. Una condición explícita sobre el diámetro del grafo conexo que define la recorrida aleatoria determina la convergencia de la cadena de Markov no homogénea $X_n^{T(n)}$. Considerando dos puntos del espacio de búsqueda como adyacentes si es posible transformar uno de ellos en el otro aplicando el operador de mutación y definiendo el diámetro del grafo de recorridas (D) como la mayor distancia entre dos nodos, la convergencia queda garantizada para poblaciones con mayor cantidad de individuos que el diámetro del grafo ($k>D$). Este resultado muestra la ventaja fundamental del algoritmo MOSES respecto a otros algoritmos evolutivos: su condición de convergencia no involucra a la función que se trata de optimizar, dependiendo únicamente del grafo definido para recorrer el espacio de búsqueda.

La decisión sobre el número de individuos participantes en la mutación se realiza siguiendo una ley binomial de parámetros k, la cantidad de individuos en la población, y $p_{MUT} = e^{-1/T}$, la probabilidad de mutación que define si cambiar un individuo o no. Para mantener la analogía con la notación utilizada para otros algoritmos de exploración markoviana, Ycart expresa p_{MUT} como una función de un parámetro T que juega el rol de la temperatura en un esquema de Simulated Annealing. Propone un esquema de decrecimiento escalonado para el parámetro T, definido por la Ecuación 1.

$$\forall q \in IN^+, \forall n \in \left(e^{(q-1)D}, e^{qD}\right)\ T(n) = 1/q$$

Ecuación 1: Esquema de decrecimiento propuesto para el parámetro T

1.2 - Variantes del algoritmo

En [3] Cercueil y Francois proponen tres variantes del algoritmo, que se describen a continuación.

A. MOSES.
   Corresponde a la versión original del algoritmo, donde el parámetro p_{MUT} se adapta de acuerdo al esquema de decrecimiento de la Ecuación 1. Los individuos se asumen etiquetados o numerados, lo cual define un orden fijo entre los individuos en la población P para efectuar la aplicación del operador de mutación sobre los "primeros" M individuos.

B. MOSES no ordenado.
   No se realiza el sorteo de un entero M para hallar los "primeros" individuos. Para cada individuo se aplica independientemente el operador de mutación con probabilidad p_{MUT} y con probabilidad 1- p_{MUT} se le reemplaza por el mejor individuo en la población.

C. MOSES ordenado.

No se realiza el sorteo del entero M sino que en cada generación se ordenan los individuos de acuerdo a su valor de fitness, creándose una lista $I^{(1)}, .. , I^{(k)}.$, Para cada individuo se aplica independientemente el operador de mutación con probabilidad p_{MUT} y con probabilidad 1- p_{MUT} se le reemplaza por un individuo mejor, tomando en cuenta la lista ordenada por fitness.

Las variantes tienen propiedades que influyen en su mecanismo de exploración. Las diferencias están dadas por los niveles de elitismo que determinan la presión de selección de cada algoritmo, definida por la relación entre el máximo de la función objetivo y su valor medio sobre la población.

En MOSES la presión de selección es débil para los individuos con los menores valores de etiquetado. La estructura jerárquica definida en la población permite que individuos con diferentes etiquetas realicen búsquedas locales a divergentes niveles. Los individuos con etiquetas menores realizan las caminatas aleatorias más largas en su exploración del espacio de búsqueda.

Respecto a las variantes propuestas, la versión no ordenada del algoritmo incrementa la presión de selección, al no existir estructura jerárquica definida sobre la población. La versión ordenada presenta el menor nivel de elitismo, ya que individuos que no mutan en una generación no son obligatoriamente reemplazados por el mejor individuo, y de este modo pueden permanecer en la población en la siguiente generación, independientemente de su valor de la función objetivo.

2. Problemas de asignación de tareas a procesadores

Los problemas de asignación de tareas a procesadores constituyen una subclase de los problemas de asignación de recursos con restricciones. Conocidos como *problemas de scheduling*, plantean hallar un ordenamiento (*planificación*) que minimice el tiempo total de ejecución de un conjunto de trabajos compuestos por tareas, sobre un número disponible de máquinas o procesadores. La *planificación* debe verificar que una misma tarea no se ejecute simultáneamente en dos máquinas diferentes y que una misma máquina no tenga asignada más de una tarea en cada unidad de tiempo.

2.1 - Open-Shop Scheduling Problem

Dentro del conjunto de problemas de asignación de tareas, el *Open-Shop Scheduling Problem* (OSSP) tiene como característica no suponer una ordenación para las tareas que conforman un trabajo. Como consecuencia, el OSSP tiene un espacio de búsqueda mayor que otras variantes de problemas de asignación. La aplicación de técnicas exactas para resolver el OSSP en tiempos razonables se hace cada vez más difícil al aumentar la complejidad del problema. En un problema de dimensión 4x4 (4 tareas para asignar a 4 procesadores) el espacio de búsqueda consiste en 16! ≈ 2.10^{13} planificaciones, lo cual muestra la complejidad del problema aún para este caso reducido.

La siguiente formulación matemática del problema se basa en el trabajo de Kahn y Crescenzi [8].

Sea un número $m \in Z^+$ de procesadores sobre los cuales se desea ejecutar un conjunto J de trabajos. Cada trabajo $j \in J$ está compuesto por m tareas $o_{p,j}$ que deben ejecutarse sobre el procesador p, con una duración $w_{p,j} \in N$. Una solución al problema de asignación queda determinada por un conjunto de funciones de planificación para cada procesador $f_p : J \to N,\ 1 \le p \le m$ que asigna a cada tarea $o_{p,j}$ el instante de tiempo en que comenzará a ser ejecutada en el procesador p, y tal que $f_p(j) > f_p(j')$ implica que $f_p(j) \ge f_p(j') + w_{p,j}$, tal que para cada $j \in J$ los intervalos $[f_p(j), f_p(j) + w_{p,j})$ son disjuntos.

El objetivo del problema de optimización es minimizar el tiempo total de ejecución de las tareas, es decir, elegir un conjunto de funciones f_p tales que minimicen el valor $\max_{1 \le p \le m,\, j \in J} f_p(j) + w_{p,j}$.

Métodos exactos como Branch & Bound involucran tiempos de ejecución considerables para hallar resultados razonables, aún para pequeñas instancias del problema. Varias propuestas aplican algoritmos genéticos a la resolución del problema, destacándose los trabajos de Khuri et al. [9] y Fang et al. [5]. Se utilizan codificaciones binarias y de permutaciones, trabajándose en general con representaciones y operadores derivados de la resolución del Traveling Salesman Problem (TSP).

Cuando el número de máquinas es mayor que 3, el problema de asignación OSSP se encuentra en la clase de problemas NP-difíciles, tal como se indica en el compendio de Garey y Johnson [7].

2.2 - Conjunto de problemas de prueba

Una serie de problemas OSSP han sido definidos para comparar los resultados obtenidos por diferentes investigadores. En este trabajo hemos utilizado instancias de prueba definidas por Taillard [11] que pueden generarse con el código libre disponible en el repositorio OR-Library [1], para evaluar los resultados de los algoritmos implementados.

	máq 1	***máq 2***	***máq 3***	***máq 4***	***máq 5***
Job 1	64	66	31	85	44
Job 2	7	69	68	14	18
Job 3	74	70	60	1	90
Job 4	54	45	98	76	13
Job 5	80	45	10	15	91

Tabla 1: OSSP de dimensión 5x5.

La Tabla 1 ofrece la formulación de un problema OSSP de dimensión 5x5, presentando las tareas de los diferentes trabajos y sus duraciones en las máquinas. Como ejemplo, la tarea 1 del trabajo 1 debe ocupar la máquina 1 por 64 unidades de tiempo, la tarea 2 del trabajo 1 debe ocupar la máquina 2 por 66 unidades de tiempo de procesamiento, y así sucesivamente. No se plantean restricciones sobre el orden de ejecución de tareas de un trabajo. Una solución válida al problema consistirá en una asignación que garantice la ejecución de todas las tareas. Una solución óptima al problema es aquella que minimiza el tiempo total de ejecución, se ofrece un ejemplo en la Figura 1.

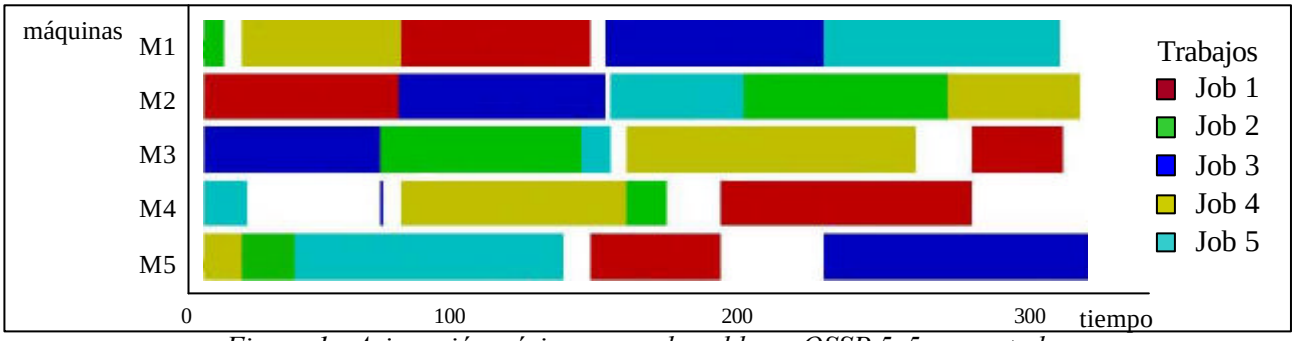

Figura 1 : Asignación mínima para el problema OSSP 5x5 presentado.

3. Codificación del problema y operadores de mutación

Esta sección presenta la codificación utilizada en la resolución del problema OSSP y sus características. Conjuntamente se presentan los operadores de mutación aplicados en la resolución del problema y se calculan los diámetros de los grafos de recorridas del espacio de búsqueda del OSSP. Finalmente se presenta una nueva variante del algoritmo MOSES que hemos propuesto para tratar de mejorar la calidad de la búsqueda evolutiva.

3.1 - Codificación del problema

En la codificación adoptada, una solución al problema OSSP queda representada por una permutación de enteros que indica el orden en que se atienden las tareas. Las tareas se numeran siguiendo un orden determinado por el identificador de trabajo, comenzando por las tareas del trabajo J_1, luego las tareas del trabajo J_2 y así sucesivamente hasta completar todas las tareas. Como ejemplo, para un problema de 5 máquinas y 5 trabajos, la primera tarea del trabajo J_1 se etiqueta con el número 0, la segunda con el 1, la primera del trabajo J_2 con 5 mientras que la ultima tarea del trabajo J_5 se etiqueta con el número 24. La cantidad de posiciones en el individuo corresponde al número de tareas a asignar. La Figura 2 muestra la codificación del individuo solución presentado en la Figura 1, para el problema OSSP ofrecido en la Tabla 1.

5	1	12	23	19	9	15	24	13	7	11	0	18	22	4	10	21	8	17	3	6	14	20	16	2

Figura 2: Codificación de una solución del problema OSSP 5x5 presentado.

Otras opciones son utilizables para la codificación de planificaciones para el problema OSSP. Es posible trabajar con individuos del mismo largo (*m.j)*, en el cual cada alelo es un valor de prioridad aleatorio entre 0 y 1. La planificación se interpreta ejecutando primero la tarea asociada a la posición en el vector de menor(mayor) prioridad, luego la segunda y así sucesivamente hasta considerar el total de tareas. En el ejemplo de 5x5, si la posición de menor valor en el vector es la 18 entonces la primera tarea a ejecutar es la T_{44}, calculada según *job = (pos div jobs)+1*, *maq = (pos mod jobs)+1*. Operadores de mutación para esta representación pueden modificar un valor al azar por una nueva prioridad, modificar *k* valores simultáneamente o tener en cuenta restricciones dependientes del planteo del problema. No hemos analizado codificaciones alternativas a la representación basada en permutaciones de enteros, pero su estudio se plantea como trabajo futuro.

3.2 - Operadores de mutación

De acuerdo al resultado teórico presentado en la sección 2, el modo de recorrer el espacio de búsqueda condiciona la eficiencia y la calidad de resultados del algoritmo. Por tal motivo, el diseño de operadores de mutación adecuados debe considerarse un objetivo de la investigación. En este trabajo hemos estudiado tres diferentes operadores de mutación aplicados al OSSP, que se basan en el intercambio de tareas en la permutación que codifica a una solución. Los operadores difieren entre sí por el mecanismo de selección de tareas a intercambiar. El Cuadro 2 presenta la descripción de los operadores, donde *cromosoma* refiere a la codificación de un individuo solución del OSSP.

- *Dos puntos al azar en el cromosoma (DPA):* Sortea dos valores entre 0 y el largo del cromosoma – 1 y se intercambian las tareas de las posiciones obtenidas.
- *Un punto al azar y el siguiente en el cromosoma (UPA):* Se sortea un valor entre 0 y el largo del cromosoma – 1 y se intercambia la tarea de la posición obtenida con la siguiente tarea en el cromosoma.
- *Un punto al azar y el siguiente en el camino critico (UPAC):* Se sortea un valor entre 0 y el largo del cromosoma – 1, que determina la primer posición de tarea a intercambiar. La segunda posición se obtiene buscando en el cromosoma, a partir de la posición ya elegida, una tarea que se encuentre en el camino critico de la tarea ya elegida. Dadas las posiciones especificadas, se intercambian las tareas correspondientes. Para el problema OSSP, el camino critico de una tarea está dado por las tareas del mismo trabajo y las de la misma maquina.

Cuadro 2: Operadores de mutación estudiados.

Los dos primeros operadores tienen formulación sencilla y figuran en la literatura como transiciones para resolver el TSP utilizando MOSES [12] o como mutación en propuestas de AG para el propio OSSP [5,9,10]. Utilizados sobre la codificación propuesta, presentan como desventaja que no garantizan que el resultado de su aplicación a un individuo produzca otro ejemplar que codifique una planificación diferente. Como caso de ejemplo de la dualidad de representaciones, la Figura 3 presenta dos codificaciones diferentes para la misma planificación de tareas.

El operador UPAC constituye nuestra propuesta para asegurar que el resultado de una mutación corresponderá a una planificación diferente a la original. El operador evita intercambiar tareas que no modifiquen sus respectivos caminos críticos, impidiendo la transición entre individuos que formen parte de la misma "clase de representaciones" para planificaciones.

El uso de varios operadores implica diferencias en los diámetros de los grafos de exploración del espacio de soluciones. Para el operador UPA, el diámetro para el problema de *j* trabajos y *m*

máquinas está dado por las permutaciones del etiquetado $D_{UPA} = \sum_{i=1}^{m.j-1} i = \frac{1}{2} mj.(jm-1)$. Para la mutación UPAC, el diámetro está dado por el número de permutaciones en el camino crítico, que corresponde a $D_{UPAC} = \sum_{i=1}^{m} \sum_{k=0}^{j-1} (i+k) = \frac{1}{2} mj.(m+j)$. Para el operador DPA, el diámetro del problema corresponde a $D_{DPA} = m.j - 1$, verificándose sencillamente por inducción completa.

1	**5**	**12**	**23**	**19**	**9**	**15**	**24**	**13**	**7**	**11**	**0**	**18**	**22**	**4**	**10**	**21**	**8**	**17**	**3**	**6**	**14**	**20**	**16**	**2**
19	**23**	**5**	**12**	**1**	**15**	**9**	**13**	**11**	**18**	**7**	**24**	**22**	**0**	**21**	**17**	**10**	**4**	**3**	**8**	**14**	**20**	**6**	**2**	**16**

Figura 3: Dos codificaciones diferentes que representan la misma planificación

3.3 - Algoritmo MOSES y sus variantes

Se trabajó sobre una implementación del algoritmo MOSES basada en la versión estándar codificada por Francois [6]. Sobre la implementación original se codificaron las variantes ordenadas y no ordenadas presentadas por Cercueil y Francois [3].

Luego de un análisis preliminar de resultados se decidió implementar una tercera variante, que denominamos algoritmo MOSES con reinicio. En ella, la probabilidad de mutación no disminuye contínuamente, sino que lo hace hasta un mínimo establecido $p_{REINICIO}$. Luego de alcanzado este valor mínimo, el algoritmo continúa la búsqueda asignando el valor inicial a la probabilidad de mutación. Esta variante pretende mejorar la exploración del espacio de búsqueda del problema, aumentando cíclicamente la cantidad de mutaciones luego de alcanzado el valor mínimo establecido. El Cuadro 3 presenta un seudocódigo de la variante propuesta, MOSES con reinicio.

```
En cada generación
    Hallar el mejor individuo I+ = Min {f(Ih), h = 1…k}
    Sortear un entero Z ∈ (0, k] de acuerdo a una ley Binomial (k, e^-1/T)
    Para individuos Ir , r = 1 … Z
            cambiar Ir = (i1, i2, …, is) en Iq = (i1', i2', …, is')
    Para individuos Ir , r = Z+1 … k
            cambiar Ir = (i1, i2, …, is) en I+
    Si ( e^-1/T > pMIN )
        Ajusto T
    Sino
        Reinicio el valor de e^-1/T
```

Cuadro 3: Seudocódigo del algoritmo MOSES con reinicio.

4. Resultados

Para la evaluación de los algoritmos implementados se trabajó sobre un subconjunto de los problemas de prueba propuestos por Taillard [11]. La Tabla 2 presenta las instancias utilizadas, indicando la dimensión de los problemas y las semillas de números seudoaleatorios para generar cada instancia con el código disponible en el repositorio OR-Library [1]. Las dos últimas columnas corresponden a los valores de la cota inferior (LowB), máximo de la suma de tiempos de ejecución por trabajos o por máquina y el mínimo conocido para los problemas considerados ($Min_{CONOCIDO}$), reportado en el reciente trabajo de Blum utilizando la metaheurística Ant Colony Optimization [2].

Problema	*Tareas*	*Máquinas*	*Semillas*	*LowB*	*$Min_{CONOCIDO}$*
P_{4x4}	4	4	1166510396, 164000672	186	193
P_{5x5}	5	5	527556884, 1343124817	295	300
P_{7x7}	7	7	609471574, 670843185	468	468
P_{10x10}	10	10	1344106948, 1868311537	637	637

Tabla 2: Descripción de las instancias de prueba utilizadas.

Cada versión de MOSES se ejecutó 100 veces sobre cada instancia de prueba. Se utilizó una probabilidad de mutación $p_{MUT} = 0.4$ y un tope inferior $p_{REINICIO} = 0.3$ en la variante con reinicio.
El estudio de las referencias no permitió determinar un criterio de finalización estándar para los algoritmos. Por simplicidad se decidió trabajar con esfuerzo prefijado, especificando un valor máximo de generaciones. Una cota adecuada es difícil de estimar, por lo cual se obtuvo un valor para el tope de generaciones en base a los experimentos de Cercueil y Francois [3]. Los autores trabajan con poblaciones donde se aplica la mutación a un número aproximado de 4 individuos y utilizan como límite 200.000 evaluaciones de la función objetivo. Traducido a generaciones, corresponde a un valor cercano a 50.000, utilizado como tope para nuestros algoritmos.

4.1 - Estudio de los operadores de mutación

Se estudiaron los operadores de mutación propuestos *UPA, UPAC y DPA*, ejecutando el algoritmo MOSES original y el MOSES con reinicio sobre las instancias de prueba P_{4x4} y P_{5x5}. Se utilizó una población superior al diámetro del espacio de búsqueda del operador UPAC (diámetro 64 para el problema P_{4x4} y 125 para el problema P_{5x5}). Los resultados se resumen en las tablas 3 y 4.

Algoritmo	*Mutación*	*Promedio*	*Desviación estándar*	*Núm. de ejecuciones en las cuales se alcanzó el Mínimo*
MOSES	UPA	199.38	4.47	8
MOSES	UPAC	194.61	1.50	39
MOSES	DPA	194.51	1.39	40
MOSES con reinicio	UPA	196.08	2.68	21
MOSES con reinicio	UPAC	194.3	1.21	44
MOSES con reinicio	DPA	194.06	1.14	52

Tabla 3: Resultados comparativos de los diferentes operadores de mutación para el problema P_{4x4} con población de 70 individuos y p_{MUT} =0.4.

Algoritmo	*Mutación*	*Promedio*	*Desviación estándar*	*Mejor resultado obtenido*
MOSES	UPA	325.15	7.95	305
MOSES	UPAC	314.09	6.65	301
MOSES	DPA	313.12	7.15	301
MOSES con reinicio	UPA	322.14	8.92	303
MOSES con reinicio	UPAC	312.34	7.08	301
MOSES con reinicio	DPA	311.41	6.11	300 (2 ejecuciones)

Tabla 4: Resultados comparativos de los diferentes operadores de mutación para el problema P_{5x5} con población de 125 individuos y p_{MUT} =0.4.

El operador DPA mostró consistentemente los mejores resultados para ambos problemas estudiados, tanto cuando se le utilizó en el algoritmo MOSES original como cuando se le utilizó en el algoritmo MOSES con reinicio. El operador UPAC tuvo un desempeño apenas menor al del operador DPA en términos de valor promedio, pero alcanzó en menos oportunidades el valor óptimo de cada problema. La diferencia entre valores promedio de ambos operadores estuvo por debajo de la desviación estándar de los resultados para ambos algoritmos y problemas estudiados.

El algoritmo MOSES con reinicio mostró mejores resultados que el algoritmo original, tanto para el valor promedio como para el número de ejecuciones en las cuales se alcanzó el óptimo conocido. Para el problema P_{4x4}, MOSES con reinicio alcanzó el óptimo en 52 ejecuciones con el operador DPA siendo éste el mejor resultado obtenido. Las mejoras que introdujo el algoritmo MOSES con reinicio para el problema P_{4x4} no se extendieron para el problema P_{5x5}. El operador DPA introduce un mayor grado de exploración del espacio de búsqueda, al producir cambios más radicales en la codificación de la planificación, mientras que el operador UPAC realiza modificaciones más certeras en la planificación, ya que solo intercambia valores dentro del camino crítico de la tarea seleccionada. Para un problema de espacio de búsqueda "pequeño", la estrategia de exploración del

operador DPA obtiene muy buenos resultados, hallando el óptimo en más de la mitad de las ejecuciones. La mejora se reduce en problemas con espacio de búsqueda mayor, como en el caso del problema P_{5x5} donde el operador orientado a la explotación obtiene resultados casi similares.

El operador UPA mostró los peores resultados, independientemente del algoritmo utilizado, para ambos problemas estudiados. La variación introducida por este operador no es eficaz comparada con la que introducen los otros dos operadores. La mutación UPA modifica la codificación intercambiando una tarea y la siguiente, pero no necesariamente modifica la planificación representada, de acuerdo a la dualidad de representaciones comentada en la sección 4.

El tamaño de la población utilizada para los problemas fue fijado de acuerdo al diámetro del espacio de búsqueda para el operador UPAC, inferior al del operador UPA. Aún incrementando el tamaño de la población de 70 a 120 individuos (diámetro del operador UPA para P_{4x4}) los resultados fueron de calidad inferior a los obtenidos con los otros dos operadores, tal como resume la Tabla 5.

Algoritmo	*Mutación*	*Población*	*Promedio*	*Desviación estándar*	*Núm. de ejecuciones en las cuales se alcanzó el Mínimo*
MOSES	UPA	70	199.38	4.47	8
MOSES con reinicio	UPA	70	196.08	1.50	21
MOSES	UPA	120	197.58	3.65	12
MOSES con reinicio	UPA	120	195.32	1.92	23

Tabla 5: Mejora obtenida al incrementar la población para el operador de mutación UPA para el problema P_{4x4} con p_{MUT} =0.4.

Como consecuencia de los resultados presentados, es posible concluir que el operador DPA presenta la mejor exploración del espacio de búsqueda entre los operadores de mutación estudiados para los problemas de asignación P_{4x4} y P_{5x5}. El operador UPAC es una alternativa para garantizar mejor explotación de soluciones intermedias de buena calidad, pero su capacidad de obtener buenos resultados finales está supeditada a un mecanismo para obtener buenas soluciones iniciales.
Por el motivo expuesto, se utilizará el operador DPA para el estudio comparativo de las diferentes variantes del algoritmo MOSES sobre los problemas de mayor dimensión.

4.2 - Estudio de las versiones del algoritmo MOSES

Para evaluar las diferentes versiones del algoritmo MOSES, se ejecutaron los algoritmos sobre el subconjunto de problemas de prueba de la Tabla 2. Los resultados se resumen en las Tablas 6 al 9, indicando el promedio de resultados sobre las 100 ejecuciones, desviación estándar y número de ejecuciones en las cuales se alcanzó el valor mínimo conocido reportado por Blum [2].

Algoritmo	*$Min_{CONOCIDO}$*	*Promedio*	*Desviación estándar*	*Núm. de ejecuciones en las cuales se alcanzó el Mínimo*
MOSES	193	194.51	1.39	40
MOSES Ordenado	193	193.10	0.44	95
MOSES No Ordenado	193	197.14	2.73	9
MOSES con reinicio	193	194.06	1.14	52

Tabla 6: Resultados comparativos de las versiones del algoritmo MOSES para el problema P_{4x4}, mutación DPA, población 70 y p_{MUT} =0.4 .

Algoritmo	*$Min_{CONOCIDO}$*	*Promedio*	*Desviación estándar*	*Núm. de ejecuciones en las cuales se alcanzó el Mínimo*	*Mejor resultado obtenido*
MOSES	300	313.12	7.15	0	301
MOSES Ordenado	300	301.64	1.10	9	**300**
MOSES No Ordenado	300	329.04	7.35	0	310
MOSES con reinicio.	300	311.41	6.11	2	**300**

Tabla 7: Resultados comparativos de las versiones del algoritmo MOSES para el problema P_{5x5}, mutación DPA, población 125 y p_{MUT} =0.4 .

Algoritmo	*$Min_{CONOCIDO}$*	*Promedio*	*Desviación estándar*	*Mejor resultado obtenido*
MOSES	468	511.81	11.09	491
MOSES Ordenado	468	488.03	6.12	**475**
MOSES No Ordenado	468	602.51	16.06	549
MOSES con reinicio	468	509.49	12.17	485

Tabla 8: Resultados comparativos de las versiones del algoritmo MOSES para el problema P7x7, mutación DPA, población 300 y p_{MUT} =0.4.

Algoritmo	*$Min_{CONOCIDO}$*	*Promedio*	*Desviación estándar*	*Mejor resultado obtenido*
MOSES	637	702.88	16.84	673
MOSES Ordenado	637	655.04	7.26	**640**
MOSES No Ordenado	637	841.27	36.79	768
MOSES con reinicio	637	701.90	17.51	673

Tabla 9: Resultados comparativos de las versiones del algoritmo MOSES para el problema P_{10x10},mutación DPA, población 1000 y p_{MUT} =0.4.

El algoritmo MOSES ordenado obtuvo los mejores resultados para cada uno de los problemas estudiados. Este comportamiento es consistente con resultados similares reportados para operadores de recombinación aplicados al problema OSSP, donde se indica que operadores que favorecen la exploración obtienen mejores resultados [10]. Además, debe tenerse en cuenta que el algoritmo MOSES ordenado no utiliza un esquema de reducción de la probabilidad de mutación, por lo cual, en cada generación, opera sobre un número mayor de individuos que el resto de los algoritmos.

Ninguno de los algoritmos propuestos fue capaz de alcanzar el valor mínimo conocido para los problemas P_{7x7} y P_{10x10} con el esfuerzo prefijado para las experimentaciones.

4.3 - Estudio de estrategias híbridas

Luego de establecer empíricamente las ventajas del operador que favorece la exploración del espacio de búsqueda, se decidió estudiar una variante en la metodología de búsqueda de las posibles soluciones para el problema OSSP. La nueva estrategia se diferencia con las anteriores en la utilización de diferentes operadores de mutación en distintas etapas de la ejecución del algoritmo. La estrategia híbrida posee una primera etapa en la cual se pondera la exploración utilizando el operador DPA y una segunda etapa en la cual a partir de la población intermedia de calidad aceptable se aplica el operador UPAC para privilegiar la explotación de buenas soluciones.

La estrategia híbrida fue probada sobre la instancia P_{5x5} utilizando los algoritmos MOSES y MOSES con reinicio. Los resultados obtenidos se resumen en la tabla 10.

Algoritmo	*Operador*	*Promedio*	*Desviación estándar*
MOSES	UPAC	314.09	6.65
MOSES	DPA	313.12	7.15
MOSES	DPA-UPAC	309.58	6.99
MOSES con reinicio.	UPAC	312.34	7.08
MOSES con reinicio.	DPA	311.41	6.11
MOSES con reinicio.	DPA-UPAC	308.73	6.73

Tabla 10: Resultados comparativos de las versiones del algoritmo MOSES para el problema P_{5x5}, con distintas estrategias de mutación, población 125 y p_{MUT} =0.4 .

El algoritmo híbrido ofreció promisorios resultados para la instancia de prueba considerada, mejorando los valores promedio respecto a los obtenidos con los operadores UPAC y DPA en solitario. Por motivos de tiempo no se ha completado el estudio de esta nueva estrategia sobre el resto de problemas de prueba, planteándose como una tarea a realizar en el futuro.

5. Conclusiones

Este artículo presenta el estudio del algoritmo MOSES y sus variantes aplicados al problema de asignación Open Shop Scheduling. El estudio contempla tres diferentes operadores de mutación, para los cuales se calcularon los diámetros de los grafos de exploración y se analizó empíricamente su vinculación con la calidad de resultados obtenidos sobre un conjunto de instancias de prueba.

El algoritmo MOSES posee propiedades teóricas de convergencia pero no existe un volumen importante de trabajo practico más allá del realizado por los propios creadores del algoritmo [3]. No existen referencias de aplicación del algoritmo a problemas de asignación en general, ni al OSSP en particular. En este sentido, los resultados experimentales presentados constituyen un aporte práctico para comprender el funcionamiento del algoritmo y su aplicabilidad a problemas de optimización.

Las variantes estándar y ordenada del algoritmo MOSES mostraron muy buenos resultados en los problemas de tamaño pequeño y resultados aceptables para los problemas de mediana dimensión. La variante propuesta en este artículo (MOSES con reinicio) tuvo un comportamiento similar, siendo superado en calidad de resultados únicamente por el algoritmo MOSES ordenado.

Respecto a los operadores de mutación propuestos, el operador DPA mostró los mejores resultados. Sin embargo, el operador UPAC obtuvo resultados de calidad similar y la mejora introducida por el operador DPA se encuentra por debajo de la desviación estándar del conjunto de resultados obtenidos, lo cual impide extraer afirmaciones concluyentes. La principal contribución del operador DPA consistió en incrementar el número de ocasiones en que se alcanza el mínimo teórico para los problemas P_{4x4} y P_{5x5}. El operador UPA mostró los peores resultados en los experimentos, comportamiento esperable de acuerdo al escaso nivel de diversidad que aporta al algoritmo.

La influencia de los diámetros de los grafos de exploración de los operadores se puso de manifiesto en los resultados comparativos ofrecidos en las tablas 3 y 4. Los mejores resultados correspondieron al operador de menor diámetro mientras que los peores corresponden al de diámetro mayor. No se pudo, sin embargo, verificar la condición teórica de convergencia ya que aún con poblaciones de tamaño mayor al diámetro del operador de mutación, no fue posible alcanzar los valores mínimos conocidos para los problemas P_{7x7} y P_{10x10}. Este resultado puede explicarse tomando en cuenta el criterio de esfuerzo prefijado utilizado para la detención de los algoritmos, que fue elegido arbitrariamente con valores deducidos de experimentos sobre otros tipos de problemas.

El propio criterio utilizado para la detención de los algoritmos determina que el estudio de eficiencia de los algoritmos sea poco útil. Trabajar con un esfuerzo prefijado impide extraer conclusiones sobre los tiempos de ejecución de los algoritmos, que tenderán a ser similares. Solo es posible estudiar los costos de ejecución analizando las características de los propios algoritmos. En el caso de las variantes del MOSES tan solo se puede argumentar que el algoritmo ordenado requiere un esfuerzo computacional mayor dado que no utiliza un esquema de reducción de la probabilidad de mutación, operando en cada generación sobre un número mayor de individuos que el resto de los algoritmos y dado que necesita ordenar la población de acuerdo a sus valores de fitness en cada generación.

6. Trabajo futuro

Una serie de aspectos sobre la aplicabilidad del algoritmo MOSES a los problemas de asignación de tareas a procesadores han quedado inconclusos y merecen mayor investigación en el futuro.

El conjunto de problemas de prueba debe ser extendido para estudiar la aplicabilidad de las variantes de MOSES a problemas de dimensión mayor. El propio conjunto de problemas de prueba de Taillard contempla problemas hasta de tamaño 20x20 para los cuales deberá estudiarse el comportamiento del algoritmo como tarea futura.

Los promisorios resultados obtenidos para la estrategia de exploración "híbrida" consisten en una línea a considerar en el futuro. Deberá verificarse su utilidad para el resto de los problemas de prueba y eventualmente mejorarse el algoritmo para decidir adecuadamente cuando dejar de ponderar la exploración para pasar a utilizar el operador que favorece la explotación. En la implementación actual utilizamos el operador DPA en la mitad de las generaciones y el operador UPAC en el resto. Evidentemente, este criterio no es el óptimo y debe mejorarse en el futuro.

La aplicación de la codificación basada en probabilidades y sus operadores asociados constituye otra línea importante a investigar en el futuro.

Los elevados tiempos de cómputo requeridos para la resolución de problemas de tamaño mediano indican que es de sumo interés el planteo de un algoritmo MOSES capaz de ejecutar en paralelo. Los algoritmos evolutivos distribuidos han sido extensamente estudiados en los últimos años y es posible indicar que una estrategia de distribución de la población y comunicación mediante migración es sencillamente implementable para el algoritmo MOSES. Conjuntamente, la extensión de los planteos teóricos sobre convergencia del algoritmo para el caso de poblaciones distribuidas, que involucrarían el estudio de la influencia del operador de migración, consistiría un importante aporte en el área.

8. Agradecimientos

Agradecemos el apoyo del Dr. Héctor Cancela del Departamento de Investigación Operativa, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay y la colaboración del Dr. Olivier Francois del Instituto Nacional Politécnico de Grenoble, Francia.

Referencias bibliográficas

[1] J.Beasley, *OR-Library: distributing test problems by electronic mail*, Journal of the Operational Research Society 41 (11), pp. 1069-1072, 1990.
Disponible online http://mscmga.ms.ic.ac.uk/info.html. Consultada junio 2003.

[2] C. Blum, *An ACO algorithm to tackle Shop Scheduling problems - Software and Results*, enviado a European Journal of Operational Research.
Disponible online. http://iridia.ulb.ac.be/~cblum. Consultado junio 2003.

[3] A. Cercueil, O. Francois. *Monte Carlo simulation and population-based optimization*. Congress on Evolutionary Computation (CEC01), pp. 191-198, Seoul, IEEE Press, 2001.

[4] L. Davis, *Job Shop Scheduling with Genetic Algorithms*, Proceedings of the 1st International Conference on Genetic Algorithms (ICGA), pp.136-140, 1985.

[5] H. Fang, P. Ross, D. Corne, *A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems.* Proceedings of 5th International Conference on Genetic Algorithms, pp 375-382, 1993.

[6] O. Francois. *Global Optimization with exploration/selection algorithms and simulated annealing*. The Annals of Applied Probability. Vol. 0 N° 0, pp. 1-24, 2001.

[7] M. Garey, D. Johnson, *Computers and Intractability*, W. Freeman & Comp., New York, 1979.

[8] V. Kahn, P. Crescenzi, *A compendium of NP Optimization Problems.* Disponible online http://www.nada.kth.se/~viggo/problemlist/compendium.html, consultado Junio 2003.

[9] S. Khuri, S. Miryala, *Genetic algorithms for solving open shop scheduling problems.* Lecture Notes in Artificial Intelligence: Progress in Artificial Intelligence, pp. 357-369, Springer, 1999.

[10] I. Labarere, V. Bernaudo, C. Salto, H. Alfonso, R. Gallard. *The role of different crossover methods when solving the open shop scheduling problem via a simple evolutionary approach*. VIII Congreso Argentino de Ciencias de la Computación, CACIC, Buenos Aires, 2002.

[11] É. Taillard, *Benchmarks for basic scheduling problems.* European Journal of Operational Research 64, pp. 278-285, 1993.

[12] B. Ycart, *Modèles et algorithmes markoviens*. Mathématiques et Applications, Springer, 2002.