Aplicación de las materias de Ingeniería de Computadores en la mejora de los Algoritmos Meméticos y Metaheurísticas en general

Fernando Palacios López, Jorge Chamorro Padial

Estudiantes del Grado en Ingeniería Informática

{ferpalacios, jorgechp}@correo.ugr.es

Resumen. En el presente artículo analizamos la aplicación de los conocimientos adquiridos en las diferentes asignaturas relacionadas con la Ingeniería de Computadores que componen el Grado en Ingeniería Informática con el uso de metaheurísticas que se imparten en la especialidad de Computación y Sistemas Inteligentes, haciendo especial enfoque en los Algoritmos Evolutivos. Nos centraremos en los Algoritmos Meméticos, comparando su rendimiento frente a los Algoritmos Genéticos. Finalmente, se comprobará la mejora del rendimiento de una metaheurística cuando se aplica Programación Paralela.

Palabras Clave: computación, metaheurísticas, algoritmos meméticos

Abstract. In this paper, we analyze how to apply the content of many subjects of Computer Engineering to those metaheuristic that are imparted in Computing and Intelligent Systems mention of Computing Engineering degree. The purpose of this analysis is to make emphasis on the importance of applying Computer Engineering techniques as a transversal knowledge of the degree. We focus on Evolutionary Algorithms, analyzing Memetic Algorithms and comparing their perfomance against Genetic Algorithms.

Keywords: computing, metaheuristics, memetic algorithms

1 Introducción

Metaheurísticas es una asignatura del tercer curso del grado Grado en Ingeniería Informática, enmarcada en la especialidad de Computación y Sistemas Inteligentes, siendo obligatoria para obtener la especialidad.

La asignatura, como su propio nombre indica, estudia diferentes metaheurísticas utilizadas en el ámbito de la Inteligencia Artificial para resolver problemas de diversa índole. El temario se estructura por temas, y en cada uno de estos temas se estudian diferentes algoritmos, según se expone a continuación [1].

- **Algoritmos de Búsqueda Local Básicos [2]:** Basan su estrategia en un estudio local del espacio de búsqueda, es decir, analizan las soluciones incluidas en el entorno de la solución actual. Dichas soluciones son denominadas soluciones vecinas.
  Estos algoritmos presentan el inconveniente de ofrecer como solución óptimos locales, que pueden llegar a estar muy alejados del óptimo global del problema.

- **Enfriamiento Simulado [3]:** Basados en la termodinámica, los algoritmos de Enfriamiento Simulado realizan una búsqueda por entornos en la que permiten movimientos a soluciones peores, con el fin de evitar finalizar en un óptimo local. Estos "movimientos de empeoramiento" se realizan de un modo controlado según un criterio probabilístico.

- **Búsqueda Tabú [4]:** Los algoritmos de Búsqueda Tabú generan entornos restringidos para evitar la exploración de zonas del espacio de búsqueda previamente visitadas. Además, implementan mecanismos de reinicialización para intensificar la búsqueda sobre zonas prometedoras y diversificar la búsqueda, visitando nuevas zonas. También permiten movimientos de empeoramiento para escapar de óptimos locales.

- **Métodos Basados en Trayectorias Múltiples [5]:** El funcionamiento de los algoritmos de este bloque consiste en la generación de una solución inicial y posteriormente en una etapa de mejora de dicha solución. Los algoritmos principales son GRASP, ILS y VNS.

- **Metaheurísticas Basadas en Poblaciones:** Estas metaheurísticas se encuentran dentro de la rama de la computación evolutiva, que recrea los mecanismos de la evolución biológica para obtener la solución a los problemas. Podemos destacar en esta clase los Algoritmos Genéticos, los Algoritmos Evolutivos [6] para optimización contínua, la Evolución Diferencial y la Programación Genética [7].

- **Metaheurísticas Basadas en Adaptación Social [8]:** Simulan el comportamiento colectivo de una sociedad para la resolución de problemas. Destacan los Algoritmos de Optimización Basados en Colonias de Hormigas y la Optimización Basada en Nubes de Partículas.

- **Metaheurísticas Híbridas:** Como su nombre indica, estas metaheurísticas combinan varios tipos de algoritmos. Podemos encontrar los Algoritmos Meméticos [9], que combinan Algoritmos Evolutivos con algoritmos de Búsqueda Local, y los algoritmos de Búsqueda Dispersa [10], un método evolutivo que también suele utilizar la Búsqueda Local como método de mejora de la solución actual.

- **Metaheurísticas Paralelas [11]:** Dado que las Metaheurísticas suelen realizar un cálculo intensivo, llega a ser necesario utilizar paralelismo para reducir el tiempo de ejecución. Este tema está centrado en el uso del paralelismo aplicado a metaheurísticas previamente estudiadas.

En este artículo, los autores nos proponemos aplicar el aprendizaje adquirido a lo largo de nuestra formación en las diferentes asignaturas de Ingeniería de Computadores (Arquitectura de Computadores, Ingeniería de Servidores, Estructura de Computadores y Tecnología y Organización de Computadores) a la implementación de metaheurísticas estudiadas en Computación y Sistemas Inteligentes, nuestra especialidad.

Para este fin, nos hemos decantado por la implementación y el análisis de un algoritmo memético y su aplicación a un problema de optimización.

La elección de un algoritmo memético en este artículo se debe a la cantidad de algoritmos que, de forma indirecta, van incluidos en él. Nos referimos a algoritmos de búsqueda local y Basados en Poblaciones. Entendemos que esta es la forma más efectiva de abarcar el mayor contenido de la asignatura Metaheurísticas en este artículo.

2 Características de los algoritmos meméticos

Los algoritmos meméticos son algoritmos basados en poblaciones y derivan de los genéticos. Estos algoritmos se basan en el concepto de *meme*. En la Antropología, un meme es la unidad mínima de información cultural que un individuo puede transferir a otro permitiendo así la preservación cultural entre generaciones.

El concepto de meme va asociado al de información. Los algoritmos meméticos tratan de utilizar el Dominio de conocimiento de un problema y preservarlo durante la búsqueda de una solución aceptable.

En los algoritmos genéticos, se generaban nuevas poblaciones a partir de las actuales, utilizando diferentes operadores (cruce, recombinación…) y posteriormente aplicando una mutación con una determinada probabilidad. Estas transformaciones no utilizan información del dominio de conocimiento para generar nuevos individuos.

Los algoritmos meméticos, en cambio, utilizan operadores que trabajan con información del dominio del problema. Estos operadores reciben el nombre de *híbridos* y traspasan atributos entre generaciones de individuos de una población.

A continuación, aclararemos el concepto de *traspaso de información* del dominio del problema entre individuos. En primer lugar, los algoritmos evolutivos son algoritmos que se han mostrado eficientes a la hora de explorar la diversidad en un espacio de soluciones. Esto nos permite tener algoritmos que hayan explorado diferentes zonas del

universo de soluciones. Sin embargo, los algoritmos evolutivos no se comportan adecuadamente a la hora de explotar un espacio de soluciones, esto es, dado una zona del espacio de soluciones, encontrar la mejor solución posible dentro de esa zona del espacio.

Por otro lado tenemos los algoritmos de búsqueda local, son algoritmos eficientes que son capaces de explotar una zona del espacio hasta encontrar la mejor solución posible. Sin embargo, son malos exploradores, es decir, no son algoritmos adecuados para buscar en diferentes zonas del espacio de soluciones.

Tenemos entonces un compromiso entre explotación y exploración. Existen algoritmos, como el de trayectorias múltiples, que comienzan a explotar desde diferentes zonas del espacio. Pero si lo que nos interesa es preservar las bondades que nos pueda ofrecer un algoritmo evolutivo a la hora de resolver nuestro problema, entonces podemos recurrir a los algoritmos meméticos.

Aún queda una cuestión pendiente, ¿nos aporta verdaderamente ventaja un algoritmo de Búsqueda Local?: Podemos considerar que un algoritmo de Búsqueda Local es un algoritmo experto en el dominio, que conoce, dadas un conjunto de soluciones, cuál es la mejor de todas. Wolpert y Macready enunciaron en 1997 el teorema "No Free Lunch"[12] por el cual, sabemos que para cada algoritmo, la ganancia en su rendimiento para un problema concreto es el coste que se paga en su rendimiento en otra clase de problema. Aplicar un algoritmo de Búsqueda Local sobre un problema determinado nos va a permitir aumentar los resultados del algoritmo en ese problema, a costa de perder eficacia en los otros.

La estructura general de un algoritmo poblacional es la siguiente [13]:

```
BÚSQUEDA POBLACIONAL
begin
        POBL <- GenerarPoblacionInicial()
        repeat
                NUEVAPOBL <- GenerarPoblacion(POBL)
                POBL <- ActualizarPobl(POBL,NUEVAPOBL)
                si POBL converge
                        POBLS <- reinicia(POBL)
                finsi
        until criterioParada
end
```

Podemos definir un algoritmo memético como "una población de agentes que alterna periodos de automejora (vía búsqueda local) con periodos de cooperación (vía recombinación), y competición (vía selección)" [14]

En los algoritmos meméticos es común hablar de agentes y no de individuos, ya que los primeros son considerados una extensión de los segundos.

Se pueden encontrar dos tipos de procesos de relación entre los agentes, los competitivos y los cooperativos. Entre los primeros tenemos la selección y la actualización, o reemplazo, mientras que entre los segundos encontramos la reproducción.

Existen dos operadores básicos de reproducción: la recombinación y la mutación. La recombinación se encarga de crear agentes nuevos haciendo uso de la información extraída de agentes recombinados, mientras que en la mutación se crean agentes nuevos mediante la modificación de otros existentes insertándoles información externa.

3 Análisis del algoritmo

En este artículo no queremos centrarnos en la construcción del algoritmo, ni en analizar cada una de sus partes. Tampoco vamos a implementar un algoritmo memético. Nos queremos centrar en aplicar paralelismo a un algoritmo memético como forma de casar el contenido de ambas especialidades del Grado.

Vamos a utilizar para las pruebas una implementación de un algoritmo memético realizada por Quang Huy Nguyen, de la Universidad Tecnológica de Nanyang. El algoritmo está publicado bajo una licencia GNU GPL v3 [15]. Es una implementación muy eficiente pero no aprovecha paralelismo. Utilizando OpenMP, añadiremos directivas necesarias para convertir el algoritmo en una solución paralela y posteriormente, compararemos resultados.

El algoritmo se ejecutará resolviendo un problema de ortogonalización utilizando los siguientes algoritmos. Como batería de pruebas, los datos de cada problema a procesar se generan de forma aleatoria en cada una de las comparaciones que se realizarán:

- Algoritmo de ortogonalización de Gram-Schmidt en un problema bidimensional [16]
- Algoritmo Genético utilizando la Función de Rastrigin [17]
- Algoritmo memético.

Tanto en el caso del algoritmo genético como en el memético, ambos estarán limitados a 50 generaciones. Gram-Schmidt no tiene restricciones y hallará un resultado óptimo.

Las baterías de pruebas anteriormente mencionadas se ejecutarán sobre dos máquinas cuyas características se muestran a continuación:

Tabla 1. Información de las máquinas utilizadas para la ejecución de las pruebas

Datos	Máquina 1	Máquina 2
Procesador	Intel Core 2 Quad CPU Q8400	AMD Athlon(tm) II X4 630
Cache	4Mb	1Mb
Frecuencia	2.66 GHz	2.8 GHz
Nº de núcleos	4	4
Memoria	4 GB	2 GB

El código de la implementación está escrito en C++, habiendo sido compilado utilizando g++ con los parámetros:

-g -Wall -O3 -D__LINUX__ -DEBUG -Wall -I. -c -w

En primer lugar, se hace una ejecución del algoritmo tal cual ha sido implementado por su autor (es decir, sin aprovechar paralelismo), obteniendo los siguientes resultados:

Tabla 2. Medición de tiempos sin paralelismo (en milisegundos)

Medición	Genético	Memético	Gram-Schmidt
MÁQUINA 1			
1	3950	3071	43425
2	4948	3036	56614
3	4705	3047	57225
4	5590	3241	49362
5	2426	4338	47424
6	2424	3780	50589
7	4420	3046	57809
Media	**4066**	**3366**	**51778**
Desv. Tipica	**1227**	**505**	**55591**
MÁQUINA 2			
1	1934	3110	339312
2	1930	3110	346738
3	1927	3256	329656
4	1942	3141	334336
5	1929	3099	337174
6	1939	3130	332181
7	1946	3110	340268
Media	**1935**	**3136**	**337095**
Desv. Tipica	**7,3**	**54,42**	**5701,98**

Hasta aquí hemos obtenido los resultados de un algoritmo que no aprovecha el paralelismo. Es decir, estamos utilizando el algoritmo de la misma forma que se explica en la asignatura Metaheurísticas. Ahora es el momento de ir más allá y de aprovechar el contenido impartido en asignaturas como Arquitectura de Computadores, estableciendo una forma rápida de paralelizar el algoritmo.

Haremos uso de OpenMP (http://openmp.org/wp/), una API que facilita la aplicación de concurrencia en C/C++. OpenMP divide en hebras de ejecución diferentes tareas. La granularidad de la división de tareas puede dejarse a la libre elección de OpenMP o bien puede ser especificada por el programador a nivel de funciones, bucles u otros bloques de código. OpenMP funciona por medio de directivas de compilación, lo que minimiza la invasión del código fuente. OpenMP se utiliza en las prácticas de Arquitectura de Computadores.

Hay muchas estrategias para paralelizar diferentes metaheurísticas. Reiteramos que no es nuestra intención hacer un análisis del algoritmo, por lo tanto, los tiempos se han medido utilizando los parámetros por defecto que establece el autor y tampoco hablaremos de los resultados obtenidos por el algoritmo (*fitness*). El objeto de este artículo no es mejorar la eficiencia de los algoritmos meméticos superando lo que se ha conseguido hasta la fecha, sino el de crear sinergias entre los contenidos de asignaturas de diferente especialidad dentro del Grado en Ingeniería Informática. Enrique Alba propone diferentes enfoques para paralelizar algoritmos evolutivos [18]. Haremos una paralelización básica y analizaremos la mejora de la eficiencia de ese algoritmo utilizando el conocimiento que hemos adquirido en Ingeniería de Servidores (ISE).

La sección de código seleccionada para ser paralelizada ha sido la función “evolve” de la clase “MemeticAlgorithm”. Esta función es la encargada de, como su nombre indica, evolucionar la población actual a un número de generación que se determina como parámetro. Por ello, es la función sobre la que recae la mayor parte de la carga computacional. Particularmente, se ha paralelizado el bucle más interno para que cada hebra se encargue de una porción del bucle:

```
#pragma omp parallel num_threads(nthreads)
{
        #pragma omp for
        for(unsigned j=0; j<v.size(); j++)
        {
                //cout << -gs->pop[v[j]]->fitness << " --> ";
                vector<double> ch = gs->pop[v[j]]->toDoubleVector();
                //cout << ls->search(ch) << " == ";
                ls->search(ch);

                // if lamarckian learning then copy back
                if (maLearningStrategy == maLSLamarckian) gs->pop[v[j]]->fromDoubleVector(ch);

                gs->pop[v[j]]->fitness = gs->evaluate(ch);

                //cout << -gs->pop[v[j]]->fitness << endl;
        }
}
```

Figura 1. Código paralelizado con directivas OpenMP

Tras compilar el algoritmo con OpenMP, obtenemos los siguientes resultados:

Tabla 3. Medición de tiempos con paralelismo (en milisegundos)

Medición	1 núcleo	2 núcleos	4 núcleos
MÁQUINA 1			
1	3071	2348	3590
2	3036	2503	3714
3	3047	2377	1964
4	3241	5114	3715
5	4338	2376	1862
6	3780	4528	2876
7	3046	2635	1782
Media	**3366**	**3126**	**2786**
Desv. Tipica	**505**	**1174**	**905**
MÁQUINA 2			
1	3110	2992	3011
2	3110	2962	2953
3	3256	2985	2950
4	3141	3039	3000
5	3099	3017	2965
6	3130	2993	2990
7	3110	3004	2925
Media	**3136**	**2999**	**2971**
Desv. Tipica	**54,42**	**24,49**	**30,87**

Podemos observar la mejora de la eficiencia del algoritmo cuando se realiza el reparto de tareas entre los diferentes núcleos.

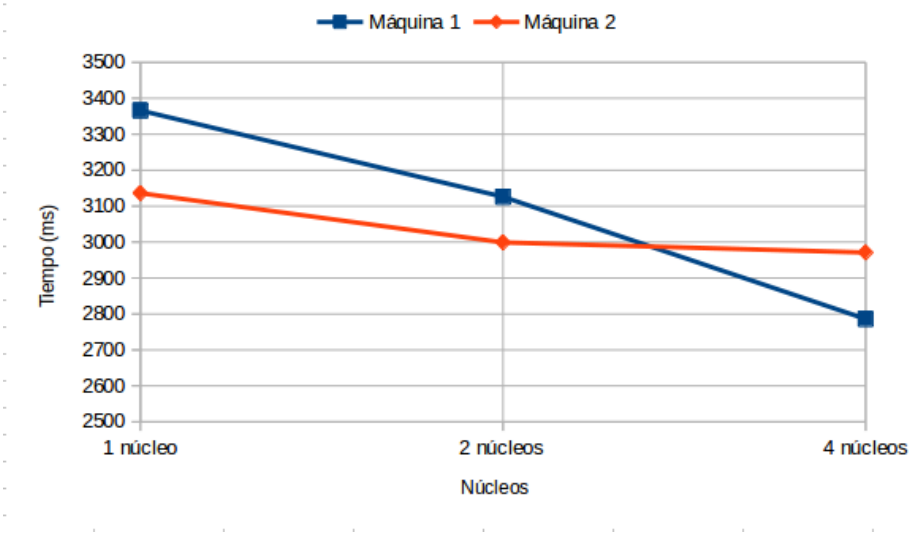

Figura 2. Evolución de tiempos de ejecución en ambas máquinas.

De la primera máquina, no podemos hablar de una mejora significativa cuando se paraleliza con dos núcleos, de acuerdo a la prueba t de student, obtenemos un p-valor de 0.6288 lo que no nos permite rechazar la hipótesis nula (es decir, que no haya diferencias significativas). Un panorama totalmente diferente nos ofrece la versión paralela de 4 núcleos, que muestra diferencias significativas con un p-valor de 0.1714, que nos permite rechazar la hipótesis nula.

Con respecto a la versión lineal, hemos obtenido un tiempo $\frac{3366}{3126} = 1.07$ veces mejor para dos núcleos y $\frac{3366}{2786} = 1.2$ en el caso de los cuatro núcleos.

En cuanto a la segunda máquina, podemos considerar que se ha producido una mejora de rendimiento respecto a la versión para cuatro núcleos rechazando la hipótesis nula con un p-valor de 0.2251.

Obtenemos un tiempo $\frac{3136}{2999} = 1.05$ veces mejor para dos núcleos y $\frac{3136}{2971} = 1.06$ en el caso de los cuatro núcleos.

4 Conclusiones

En la especialidad de Computación y Sistemas Inteligentes (CSI) se estudia de forma teórica gran cantidad de metaheurísticas, heurísticas y algoritmos y otras técnicas de gran utilidad en diferentes campos de las Tecnologías de la Información y la Comunicación (criptografía, visión por computador, aprendizaje automático, robótica, análisis de señales…). Pero, a la hora de enfrentarnos a un problema real, estos conocimientos son insuficientes sin el apoyo del conocimiento que proporcionan asignaturas como TOC, EC, AC o ISE y que, de un modo u otro, siempre están y han de estar presentes. Creemos indispensable profundizar en el temario de las mismas y que se transmita la importancia de estos contenidos a la hora de desarrollar soluciones de calidad.

En este artículo hemos comprobado la evolución en el rendimiento de un algoritmo memético cuya ejecución se ha paralelizado. Los resultados, aún sin haber realizado un análisis en detalle sobre la mejor configuración de sus parámetros o la mejor estrategia de paralización del algoritmo, muestran que hemos conseguido mejoras sustanciales.

Estos resultados reflejan la necesidad de transmitir a los estudiantes del grado la importancia, ya no solo de que sean impartidos, sino también de incidir en aquellos contenidos que se han de adquirir por primera vez durante el segundo año de carrera, pero que deben formar parte de la formación del estudiante en etapas posteriores.

Agradecimientos. Nos gustaría mostrar nuestros agradecimientos a Roberto Cogolludo Ayuso, graduado en Traducción e Interpretación por la Universidad de Granada, por su labor como revisor de este artículo y a Francisco Herrera Triguero, profesor de la asignatura de Metaheurísticas, por su labor como profesor y por la información, de gran utilidad, ofrecida para redactar este artículo.

Referencias

1. Material de la asignatura Metaheurísticas proporcionado por Francisco Herrera Triguero. 3º de Grado de Ingeniería Informática, perfil de Computación y Sistemas Inteligentes, 2013-2014. http://sci2s.ugr.es/docencia/metah/ en línea 30 de marzo de 2015.
2. E.-G. Talbi. Metaheuristics. From design to implementation. Wiley, 2009.
3. S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by Simulated Annealing, Science 220:4598 (1983) 671-680.
4. F. Glover, M. Laguna. Tabu Search. Kluwer Academic, 1997
5. F. Glover, G.A. Kochenberber. Handbook of Metaheuristics. Kluwer Acad., 2003. Cap 12: Multi-start Methods. (Rafael Marti), 355-368. Cap 8: Greedy Randomized Adaptive Search Procedure. (M.G.C. Resende, C.S. Ribeiro), 219-251. Cap 6: Variable Neighborhood Search, P.hansen, N.Mladenovic, 145-184, Cap: 11, Iterated Local Search, H.R. Lourenço, O.C. Martin, T.Stützle, 321-353.

6. D.B. Fogel (Ed.). Evolutionary Computation. The Fossil Record. (Selected Readings on the History of Evolutionary Computation). IEEE Press 1998.
7. J. R. Koza, Genetic Programming. MI Press, 1992.
8. E. Bonabeau, M. Dorigo, G. Theraulaz. Swarm Intelligence. From Nature to Artificial Systems. Oxford University Press, 1999.
9. P. Moscato, C. Cotta, "A Gentle Introduction to Memetic Algorithms". In: F. Glover, G.A. Kochenberber, (Eds.). Handbook of Metaheuristics. Kluwer Academics. (2003) 105-144, Kluwer, Boston MA, 2003.
10. M. Laguna, R.Martí. Scatter Search. Kluwer, 2002.
11. E. Alba (ed.), "Parallel Metaheuristics", John Wiley & Sons, 2005.
12. Wolpert, D.h., y W.g. Macready. "No Free Lunch Theorems for Optimization." IEEE Transactions on Evolutionary Computation (1997): 67-82.
13. Morales, Eduardo F. "Algoritmos Meméticos." Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), México, 2 Nov. 2004. Web. 22 Mar. 2015. <http://ccc.inaoep.mx/~emorales/Cursos/Busqueda/node117.html >
14. Moscato, P.A. (1989). On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Caltech Concurrent Computation Program Report 826, Catech, Pasadena, California.
15. Huy Nguyen, Quang. "Memetic Algorithm Package". 1 Ene. 2010. Web. 24 Mar. 2015. <https://code.google.com/p/memetic-algorithm/ >
16. I.V. Proskuryakov "*Encyclopedia of Mathematics*". Orthogonalization (ISBN: 1402006098) Web. 7 May. 2015. <http://www.encyclopediaofmath.org/index.php/Orthogonalization >
17. H. Mühlenbein, D. Schomisch y J. Born. "The Parallel Genetic Algorithm as Function Optimizer ". Parallel Computing, 17, páginas 619–632, 1991.
18. Alba, Enrique y Tomassini, Marco. "Parallelism and evolutionary algorithms" IEEE Transactions on Evolutionary Computation (2010): 443-462