DISEÑO DE FILTROS ANALÓGICOS PASIVOS BASADOS EN PROGRAMACIÓN GENÉTICA

Chouza, M., Rancán, C., Clúa, O., García-Martínez, R.

Laboratorio de Sistemas Inteligentes. Facultad de Ingeniería. UBA
Grupo de Sistemas Distribuidos Heterogéneos. Facultad de Ingeniería. UBA
Centro de Ingeniería de Software e Ingeniería del Conocimiento. UBA

mmc@chouzakeil.com.ar, crancan@itba.edu.ar, oclua@acm.org, rgm@itba.edu.ar

Resumen

El diseño de filtros analógicos pasivos es una tarea que presenta alta complejidad por la cantidad de factores que intervienen en su desarrollo, relacionados principalmente con la topología del circuito y los distintos valores que pueden tomar sus componentes. La Programación Genética, que consiste en la aplicación de un procedimiento evolutivo similar al de los algoritmos genéticos a programas en lugar de a representaciones fijas de las posibles soluciones, constituye una herramienta poderosa para automatizar esa tarea, permitiendo alcanzar altos grados de eficiencia en el diseño. No obstante ello, también estas herramientas deben emplearse con cuidado en razón del elevado crecimiento de las poblaciones de individuos y el consecuente aumento del esfuerzo computacional. Por tal razón, en el presente trabajo se proponen técnicas tendientes a controlar estos aspectos negativos comprobándose su efectividad en experimentos controlados que demuestran su eficiencia.

Palabras clave: Diseño de Filtros Analógicos – Programación Genética

1. INTRODUCCIÓN

Las técnicas convencionales de diseño de filtros asumen una topología particular del circuito, limitando el espacio de diseño y, por lo tanto, la optimalidad de los resultados obtenidos. La aplicación de la programación genética a la resolución de este problema de diseño [Koza *et al.*, 1997] permite explorar, no solo sobre el espacio dado por los valores de los componentes, sino sobre las distintas topologías posibles. Esto es lo que le ha posibilitado obtener resultados que pueden ser considerados competitivos con los alcanzados por diseñadores humanos.

En la resolución de problemas reales, uno de los inconvenientes principales que presenta la utilización de la programación genética es el crecimiento del tamaño de los individuos evaluados y su consiguiente efecto en el rendimiento [Streeter, 2003]. Algunas técnicas para resolver este problema se han explorado en dominios restringidos [Ryan *et al.*, 2003], pero su escalabilidad y aplicabilidad no han sido demostradas.

El diseño de filtros analógicos pasivos es un campo particular dentro del dominio del diseño de filtros analógicos. Dentro de este campo, las complejas interacciones no lineales entre los valores de los componentes y las características del filtro dificultan la aplicación de técnicas convencionales en topologías generales.

En este contexto, este trabajo explora la aplicación de tres técnicas de optimización al diseño de filtros analógicos pasivos utilizando programación genética. El estado de la cuestión se presenta en la sección 2, tanto en el diseño de filtros analógicos pasivos (sección 2.1) como en cuanto a la

programación genética aplicada a este problema (sección 2.2); los problemas tratados en este trabajo se indican en la sección 3; las técnicas propuestas para resolver estos problemas en la sección 4; los resultados experimentales se muestran en la sección 5 y algunas conclusiones y futuras líneas de investigación se indican en la sección 6.

2. ESTADO DE LA CUESTIÓN

2.1. Diseño de Filtros Analógicos Pasivos

2.1.1. Definición y Clasificación

Se denomina filtro a un dispositivo que procesa señales con el objeto de alterar su espectro. La forma en que realiza esa alteración, para una gran variedad de filtros, puede describirse dando su función de transferencia. Este trabajo se concentra en los filtros analógicos pasivos. Estos son llamados analógicos por trabajar con señales de esta clase, presenten en una gran gama de aplicaciones incluyendo prácticamente todas las formas de interacción con seres humanos. El término pasivo se refiere a la clase de componentes que se utilizan para construirlo; en nuestro caso la utilizamos para indicar que está constituido por resistores, capacitores e inductores [Paarmann, 2003].

2.1.2. Procedimientos Manuales de Diseño

Los procedimientos manuales típicos se utilizan para un conjunto preestablecido de transferencias y suponiendo una topología "ladder" para el circuito. Estos se basan en la utilización de una tabla en la que las entradas son la forma y orden de aproximación del filtro a diseñar y las salidas son los valores normalizados de sus componentes. Como las tablas se calculan para una frecuencia de trabajo fija, estos valores deben "desnormalizarse" para llevar el filtro a la frecuencia de trabajo deseada [Paarmann, 2003].
La disponibilidad de herramientas automatizadas ha disminuido la utilización de estos procedimientos en la práctica, pero siguen utilizándose en la enseñanza por su carácter pedagógico [Koller & Wilamowski, 1996].

2.1.3. Herramientas Automatizadas Convencionales

La gran mayoría de las herramientas convencionales son desarrollos más o menos directos de los procedimientos "históricos" [Koller & Wilamowski, 1996] [Nuhertz, 2008]. Las entradas son el orden de aproximación, la forma de aproximación y la topología a utilizar (donde "ladder" es una de las opciones). La salida es el diseño completo del circuito, incluyendo los valores de los componentes. Una contribución de esta clase de herramientas es que permiten realizar un análisis de sensibilidad respecto a variaciones en los valores de los componentes, ayudando a detectar diseños no satisfactorios, pero sus limitaciones respecto a la incapacidad de generar nuevas topologías persisten incluso en sistemas sofisticados [Koza *et al*., 1997].

2.2. Programación Genética y su Aplicación al Diseño de Filtros Analógicos

2.2.1. Definición

En este trabajo continuamos explorando la aplicación de la programación genética [Koza, 1992] al diseño de filtros analógicos [Koza *et al*., 1997]. Esta técnica consiste en la aplicación de un procedimiento evolutivo similar al de los algoritmos genéticos a programas en lugar de a representaciones fijas de las posibles soluciones.
En algunos casos el mismo programa puede ser la solución buscada, sin embargo en la gran mayoría de los casos, la solución buscada es alguna clase de estructura, no un algoritmo. La forma en la que se resuelve este conflicto consiste en hacer que el programa desarrollado construya la solución como efecto de su ejecución, en forma análoga al proceso de desarrollo embrionario [Gruau, 1992].

2.2.2. Estructuras de los Programas

Los programas sobre los que opera el proceso de desarrollo podrían tomar múltiples formas, pero en la práctica se han utilizado dos variantes: la estructura en árbol y la estructura lineal. La estructura en árbol fue la utilizada en los trabajos originales en el área [Koza, 1992] y está basada en la empleada por el lenguaje Lisp. Por otro lado, la estructura lineal emula a los lenguajes imperativos convencionales y fue desarrollada unos años más tarde. A pesar de un creciente interés en las estructuras lineales, la gran mayoría de los trabajos en el área siguen efectuándose utilizando estructuras en árbol [Brameier, 2004].

2.2.3. Representaciones de Circuitos

Para el caso de la aplicación de la programación genética al diseño de filtros analógicos, una de las decisiones a tomar es la elección de cómo se representará al circuito. Las dos variantes que han sido utilizadas exitosamente en esta materia han sido la representación convencional [Koza *et al.*, 1997] (ver figura 1) y la representación utilizando "bond graphs" [Hu *et al.*, 2005] (ver figura 2).
La representación convencional de un circuito consiste en un grafo, donde las aristas simbolizan a los componentes y los nodos las interconexiones de los mismos. La principal ventaja de esta representación es la existencia de un gran número de herramientas de simulación compatibles, tales como el SPICE [Quarles, 1989].

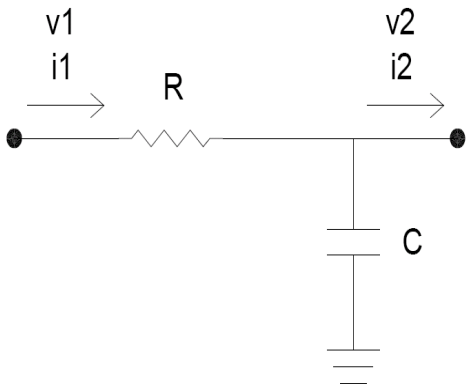

Fig. 1. Representación convencional de un filtro pasabajos.

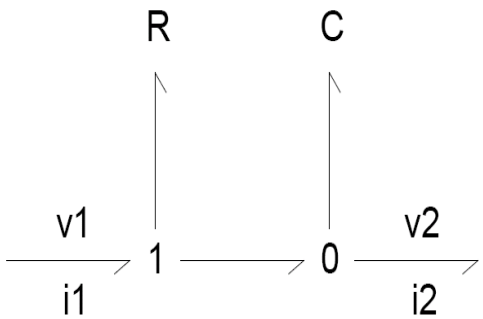

Fig. 2. Representación de un filtro pasabajos como un "bond graph".

3. PROBLEMAS

3.1. Crecimiento del Tamaño de los Individuos

En la aplicación práctica de la programación genética suele producirse un crecimiento del tamaño de los individuos [Streeter, 2003]. Este aumento del tamaño trae dos inconvenientes:

- **Mayor consumo de memoria:** Como es natural, un mayor tamaño de los individuos lleva, si se mantiene una población constante, a mayores requerimientos de memoria para poder almacenarla.
- **Mayores tiempos de evaluación:** En una gran variedad de problemas, un aumento en el tamaño de los individuos lleva a un tiempo de evaluación mayor.

Ambos aspectos limitan la escalabilidad de la programación genética, al impedir un aumento del rendimiento proporcional al crecimiento de los recursos computacionales.

3.2. Convergencia Prematura

El otro problema que suele presentarse es uno habitual en las técnicas de búsqueda: la convergencia prematura. Este problema ocurre cuando la variabilidad en la población disminuye sin haberse llegado a una solución aceptable, dejando al sistema atrapado en una solución sub-óptima. Al igual que el problema anterior, disminuye la calidad de las soluciones para una cantidad dada de recursos.

4. SOLUCIÓN PROPUESTA

La forma más directa de resolver el problema mencionado en la sección anterior sería penalizar a los individuos de mayor tamaño. Sin embargo, esto trae inconvenientes en el desarrollo del proceso evolutivo, ya que la presencia de estos individuos es requerida para actuar como "puentes" entre distintas configuraciones. De no encontrarse presentes, la población quedaría atrapada en mínimos locales.
Para resolver este problema han sido planteadas múltiples estrategias, tal como puede verse en [Ryan *et al.*, 2003]. Exploraremos la aplicación de dos de las técnicas mencionadas en la referencia anterior, además de una tercera técnica capaz de ser aplicada en conjunto con cualquiera de éstas, al diseño de filtros utilizando programación genética.

4.1. Plagas para Disminuir el Esfuerzo Computacional

Como se mencionó anteriormente, uno de los problemas más importantes de la programación genética es el aumento del tamaño de los individuos a medida que se desarrolla el proceso evolutivo, lo que se conoce como "bloat". Esto ocasiona, de mantenerse constante la cantidad de individuos, un aumento progresivo del tiempo de procesamiento por generación y de la cantidad de memoria que el proceso requiere.
Si en lugar de medir los resultados obtenidos respecto a la cantidad de generaciones, se lo hace en relación al esfuerzo computacional realizado, se observan resultados positivos al suprimir una cierta

cantidad de individuos (los que presenten peor puntaje) por cada generación. Este proceso ha sido denominado por sus autores como plaga [Fernández *et al.*, 2003].

4.2. Ajuste Dinámico de la Función de Evaluación

Una de las formas simples de controlar el tamaño de los individuos es la de penalizarlos según el valor de este. Sin embargo, tal como se describió anteriormente, esto tiene consecuencias indeseables sobre el proceso evolutivo. Este método, descrito en [Poli, 2003], se basa en crear "agujeros" en la función de evaluación en forma dinámica.
Este proceso consiste en seleccionar en forma aleatoria un cierto número de individuos cuyo tamaño supere un cierto valor preestablecido (por lo general el tamaño promedio) y se los elimina. Este método no impedirá a la población aumentar su tamaño promedio, si esto fuera necesario para mejorar su ajuste, pero proporciona un freno al crecimiento del mismo. Otras variantes posibles de esta estrategia incluyen el hacer la probabilidad de eliminación dependiente del puntaje de los individuos y el seleccionar automáticamente como integrantes de la próxima generación a los individuos que sean menores de cierto tamaño mínimo.

4.3. Utilización de un Caché de Evaluación

Podemos denominar a las dos técnicas descritas anteriormente como "técnicas de control de poblaciones", ya que buscan operar en forma más o menos directa sobre la constitución de la misma. La técnica que vamos a ver en esta sección opera de un modo que, idealmente, no modificaría el desarrollo del proceso evolutivo.
La utilización de un caché de evaluación consiste en almacenar el puntaje resultante de aplicar el proceso normal de evaluación a cada individuo, de modo que las posteriores evaluaciones de ese mismo individuo puedan resolverse mediante la simple lectura del puntaje almacenado previamente. Esta técnica, también denominada "memorización", ha sido aplicada previamente a los algoritmos genéticos [Povinelli & Feng, 1999] y en general a un gran número de problemas donde la evaluación de una función es costosa y pueden repetirse los valores a evaluar.
Por razones de eficiencia se decidió la utilización de una tabla *hash* para almacenar los puntajes, utilizando *como clave* un *hash* de 64 bits del individuo evaluado. Como los puntajes ingresados ya poseen una distribución de claves uniforme, no resulta necesario realizar un paso posterior de *hashing* para distribuir los elementos en la tabla.
Al tratarse de un caché, no es necesario un manejo sofisticado de las colisiones. En caso de producirse alguna, el nuevo par *hash*-puntaje sobrescribe al anteriormente almacenado sin producirse error alguno. El único error posible estaría dado por la presencia de individuos distintos con idénticos *hashes.* Pero, suponiendo el uso de una función de *hash* de 64 bits razonablemente efectiva, esta probabilidad está dada aproximadamente por

$$p(n) \approx 1 - \exp\left(-\frac{n^2}{2 \cdot 2^{64}} \right),$$

donde n representa el número de individuos insertados [van Oorschot & Wiener, 1994]. Como la cantidad de individuos en cuestión es notablemente inferior a 2^{32}, la probabilidad de que se produzca una "colisión" de *hashes* es muy baja.

5. VERIFICACIÓN EXPERIMENTAL DE LA PROPUESTA

5.1. Generalidades

Todas las comparaciones se efectuaron utilizando una población de 1000 individuos, 100 generaciones y 10 ejecuciones independientes por cada método. Se realizó una selección por ranking elitista, con una distribución de probabilidad exponencial sobre todos los individuos no automáticamente seleccionados, de modo que la probabilidad de seleccionar al individuo *i*-ésimo sea de $P(i) \approx \lambda e^{-\lambda i}$. Se utilizó $\lambda = 0.002$, equivalen-te a $\lambda = 2$ sobre los rankings normalizados al intervalo [0, 1). Las probabilidades de cruza, mutación y modificación de valores (una variante de mutación que solo afecta valores numéricos) fueron optimizadas empíricamente, obteniéndose los valores de 0.1 y 0.2, respectivamente.
Para la aplicación de plagas se eliminó a los 10 individuos con menor puntaje de cada generación en la que hubiera más de 10 individuos; de este modo el proceso evolutivo dejaba de aplicarse después de la generación 99. La probabilidad de eliminar a los individuos con tamaño por encima del promedio en el método de ajuste dinámico de la función de evaluación fue elegida en 0.2. Ambos valores fueron optimizados en forma empírica.
Para todas las pruebas que lo requerían se utilizó un caché con 2^{20} elementos que, considerando que se utilizan 8 bytes para el *hash* del individuo y otros 8 bytes para el puntaje, representan un consumo de 16 MB de RAM para este fin. Este valor fue seleccionado debido a que, al trabajar con un máximo de 100000 individuos distintos, no existían ventajas significativas en utilizar un caché de mayor tamaño. La función de *hash* elegida fue *MurmurHash* 2.0 [Appleby, 2008], debido a su buen desempeño en las pruebas realizadas.
La utilización de un caché de evaluación es esencialmente ortogonal a la aplicación de las otras dos técnicas mencionadas, ya que solo afecta el proceso de evaluación en si. Por lo tanto, no solo se lo evaluó por separado, sino que también se realizaron pruebas en combinación con los métodos de aplicación de plagas para reducir la carga computacional y de ajuste dinámico de la función de evaluación.
El problema elegido para las pruebas fue la obtención de un filtro pasabajos con una frecuencia de corte de 10 kHz. Al ser un filtro pasivo, tiene especial importancia la selección de la impedancia de la fuente de señal y de la impedancia de carga. Como el filtro iba a ser diseñado exclusivamente con el objeto de comparar distintas técnicas de optimización, se les asignó en forma arbitraria un valor puramente resistivo de 1 kΩ a ambas impedancias. La función de evaluación seleccionada, consistió en efectuar la suma de las diferencias al cuadrado entre la transferencia real y la ideal en un total de 50 puntos, distribuidos logarítmicamente entre 1 Hz y 100 kHz.

5.2. Relación entre el Tamaño de los Individuos y el Tiempo de su Evaluación

El primer análisis realizado fue con el objetivo de determinar la relación entre el tiempo empleado en evaluar cada generación y el tamaño promedio de sus individuos. Fue realizado sobre el caso base, sin aplicar ninguna de las técnicas para controlar el crecimiento de los individuos. En caso de aplicarlas no cabría esperar un cambio en los resultados, ya que solo afectarían el tamaño de los individuos y no la correlación de este con el tiempo de evaluación.
En la figura 3 podemos observar la relación entre el tamaño promedio de los individuos evaluados en una generación y el tiempo demorado en evaluar toda la población.
Realizando un ajuste a los datos con una expresión de la forma $A \cdot (tam_{indiv})^B$ se obtuvo como resultado un exponente B de 2.03, lo que nos indica que el tiempo de evaluación depende en forma

aproximadamente cuadrática del tamaño de los individuos. Esto, junto a la observación de que algunas generaciones demoraron más de 600 segundos en evaluarse, subraya la importancia de controlar el tamaño de los individuos.

5.3. Comparación de Resultados Obtenidos por las Distintas Técnicas

En la figura 4 podemos observar un análisis comparativo de los resultados finales obtenidos en cada una de las ejecuciones y aplicando cada una de las posibles combinaciones de las técnicas descritas en la sección 4. Los símbolos indican los valores promedios de puntaje y duración de cada ejecución, mientras que las barras indican el rango observado en cada uno de estos parámetros.
En primer lugar, se observa una marcada diferencia entre las técnicas que se combinan con el uso del caché de evaluación (marcadas en negro) y sus contrapartidas que no lo hacen (marcadas en gris). El uso del caché se acompaña por una reducción de los tiempos de ejecución de alrededor de un orden de magnitud, sin observarse una disminución en la optimalidad de los resultados obtenidos (recordar que menores puntajes indican filtros mejor adaptados).

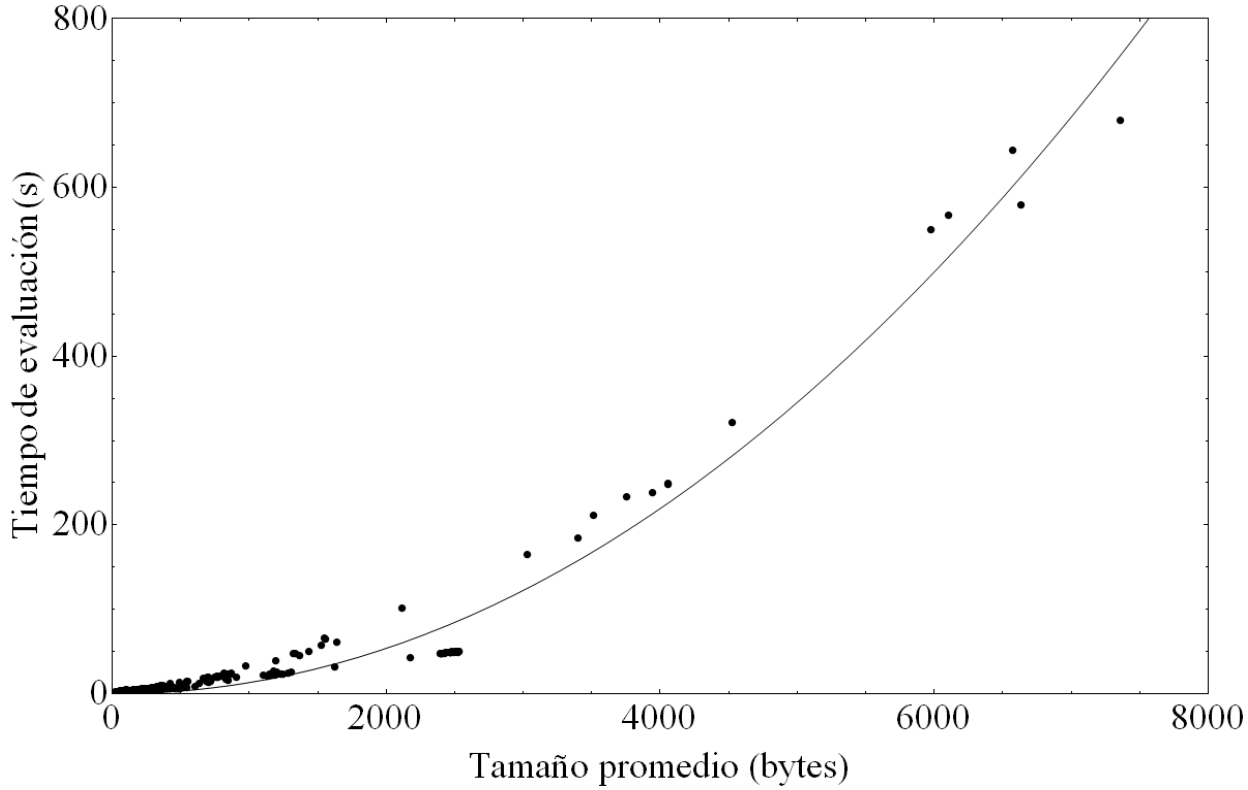

Fig. 3. Crecimiento del tiempo de evaluación de una generación en función del tamaño promedio de los individuos.

La otra diferencia notable que se observa es entre los tiempos obtenidos aplicando plagas para controlar el crecimiento del esfuerzo computacional y los que se logaron aplicando el ajuste dinámico de la función de evaluación o sin aplicar ninguna de las técnicas de control de poblaciones. Si se toman en forma combinada, la utilización de plagas y la aplicación de un caché de evaluación permiten un desempeño casi dos órdenes de magnitud superior al obtenido en el caso base, sin aplicar ninguna técnica de optimización.

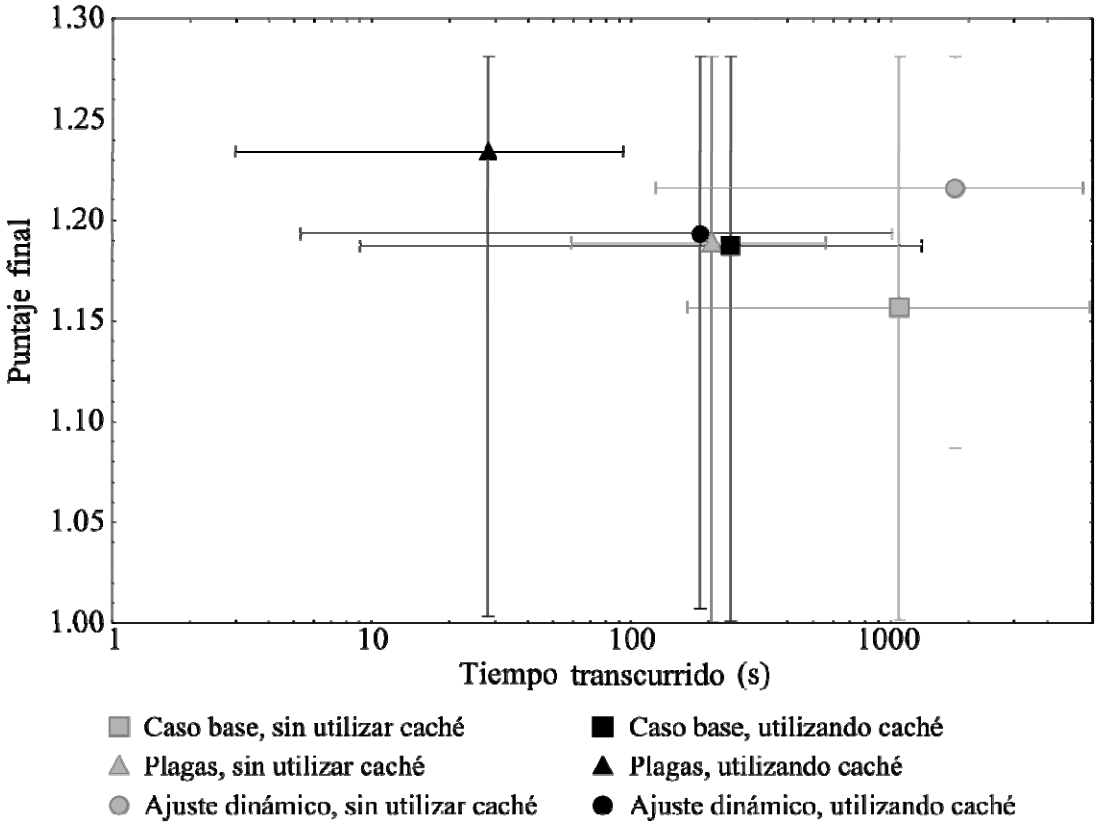

Fig. 4. Comparación entre el desempeño de las distintas técnicas de optimización.

6. CONCLUSIONES

Una limitación de la aplicación de la programación genética a la resolución de problemas reales radica en la gran cantidad de recursos computacionales requeridos para afrontar problemas con una complejidad apreciable. En este trabajo se presentó una posible aproximación para mitigar este problema, en la que se evaluaron tres técnicas de optimización para reducir la carga computacional del proceso evolutivo: uso de plagas para controlar el esfuerzo computacional, ajuste dinámico de la función de evaluación y aplicación de un caché de evaluación. La última de estas técnicas permite además ser aplicada en combinación con cualquiera de las dos anteriores.

Encontramos que tanto la aplicación de un caché de evaluación como el uso de plagas permitía mejoras notables en los tiempos de evaluación, reduciéndolos en casi un orden de magnitud sin afectar a la calidad de los resultados obtenidos. Operando en conjunto, permiten reducir el tiempo de evaluación en casi dos órdenes de magnitud respecto al caso base. El ajuste dinámico de la función de evaluación tuvo un desempeño similar al del caso base, sin mostrar una diferencia significativa en los tiempos de ejecución.

7. REFERENCIAS

Appleby, A. (2008). *MurmurHash 2.0*. http://murmurhash.googlepages.com/. Último acceso 10 de junio de 2008.

Brameier, M. (2004) *On Linear Genetic Programming*. Tesis doctoral. Universität Dortmund, Dortmund, Germany.

Fernandez, F., Vanneschi, L., Tomassini, M. (2003). *The Effect of Plagues in Genetic Programming: A Study of Variable-Size Populations*. Lectures Notes in Computer Science 2610: 317-326.

Gruau, F. (1992) *Cellular encoding of Genetic Neural Networks.* Informe técnico 92-21, Laboratoire de l'Informatique du Parallilisme. Ecole Normale Supirieure de Lyon, France

Hu, J., Zhong, X., Goodman, E. (2005). Open *Ended Robust Design of Analog Filters Using Genetic Programming*. Proceedings of the 2005 ACM Conference on Genetic and Evolutionary Computation: 1619-1626

Koller, R., Wilamowski, B. (1996). *LADDER. A Microcomputer Tool for Passive Filter Design and Simulation*. IEEE Transactions on Education 39(4): 478-487.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Koza, J., Bennett, F., Andre, D., Keane, M., Dunlap, F. (1997). *Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming*. IEEE Transactions on Evolutionary Computations 1(2): 109-128.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press.

Nuhertz Technologies. (2008). *Passive Filters Solutions*. Disponible al 20 de abril de 2008. http://www.filter-solutions.com/passive.html

Paarmann, L. (2003). *Design and Analysis of Analog Filters*. Kluwer Academic Publishers.

Poli, R. (2003). *A Simple but Theoretically-Motivated Method to Control Bloat in Genetic Programming*. Lectures Notes in Computer Science 2610: 204-217.

Povinelli, R., Feng, X. (1999). *Improving Genetic Algorithms Performance by Hashing Fitness Values*. Proceedings of the Artificial Neural Networks in Engineering Conference (ANNIE) '99: 399-404

Quarles, T. (1989). *The SPICE3 Implementation Guide*. Memorandum Nro. UCB/ERL M89/44. Electronics Research Laboratory. University of California at Berkeley.

Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., Costa, E. (editores) (2003). *Proceedings Euro Genetic Programming 2003 Euro GP 2003*. Lectures Notes in Computer Science 2610.

Streeter, M. (2003). *The Root Causes of Code Growth in Genetic Programming*. Lectures Notes in Computer Science 2610: 443-454

van Oorschot, P., Wiener, M. (1994). *Parallel collision search with application to hash functions and discrete logarithms.* Proceedings of the 2nd ACM Conference on Computer and communications security: 210-218.

.