UNA HERRAMIENTA DE META-EVOLUCIÓN PARALELA PARA LA ENTONACIÓN DE PROGRAMAS EVOLUTIVOS

■ Joel Rivas, Freddy Perozo

email: jrivas@uc.edu.ve, fperozo@uc.edu.ve

Departamento de Computación, Facultad Experimental de Ciencias y Tecnología Universidad de Carabobo, Valencia - Venezuela

■ Rosseline Rodríguez, Leonid Tineo

email: crodrig@usb.ve, leonid@usb.ve

crodrig@ldc.usb.ve, leonid@usb.ve, aurrutia@ucm.cl

Departamento de Computación, Universidad Simón Bolívar, Caracas – Venezuela

Fecha de Recepción: 20 de agosto de 2011

Fecha de Aceptación: 12 de marzo de 2012

RESUMEN

Se presenta una herramienta para entonar parámetros de control de programas evolutivos (probabilidad de cruce, probabilidad de mutación, tamaño de la población y gap generacional). Esta herramienta fue concebida con un enfoque de meta-evolución: en el nivel superior, un algoritmo genético optimiza a un programa evolutivo, colocado en el nivel inferior. Dado el volumen de cómputo que esto puede implicar, se usa una configuración paralela maestro-esclavos, implantada de manera transparente en un cluster con tecnología OpenMosix. Los resultados experimentales confirman la validez y generalidad de esta herramienta.

Palabras Clave algoritmos genéticos, programas evolutivos, meta-evolución, parámetros de control

A META-EVOLUTION PARALLEL TOOL FOR EVOLUTIONARY PROGRAMS TUNING

ABSTRACT

We present a tool for tuning control parameters of evolutionary programs (crossover probability, mutation probability, population size and generation gap). This tool was designed with a focus on meta-evolution: the top level, a genetic algorithm optimizes an evolutionary program in lower level. Given the amount of computation that this may mean is used master-slave parallel configuration, implemented transparently in a cluster with OpenMosix technology. The experimental results confirm the validity and generality of this tool.

Keywords genetic algorithms, evolutionary programs, meta-evolution, control parameters

1. INTRODUCCIÓN

En las técnicas de Computación Evolutiva se necesitan valores adecuados para parámetros de control como tasas de cruce y mutación, tamaño de la población, entre otros [6]. Estos parámetros permiten converger eficientemente a una solución adecuada. Si se quiere proveer una parametrización estándar que funcione en cualquier programa evolutivo, se cae en un problema complejo. Es una búsqueda exhaustiva cuyo objetivo es encontrar una combinación óptima de valores (a explorar) para los parámetros.

El diseñador de un programa evolutivo determinado debe definir la población inicial de individuos, los mejores operadores a utilizar (selección, cruce y mutación) así como también los parámetros de control que produzcan la convergencia del programa y arrojen los mejores resultados. Su decisión, respecto a los parámetros, debe basarse en el chequeo sistemático dentro de un rango de valores [7]. La elección inapropiada de los valores de estos parámetros puede incidir, de tal manera, que la convergencia del programa evolutivo fracase.

Esto se realiza mediante combinaciones de los valores de los parámetros de manera ordenada y sistemática. Por ejemplo, se va moviendo la probabilidad de cruce (*pc*) desde 0.1 hasta 1.0, en pasos de 0.1, lo que resulta en diez valores posibles para pc. Luego, para cada valor de *pc* se mueven los demás parámetros: probabilidad de mutación (*pm*), tamaño población (*pob*) y *gap* generacional. Si cada una de estas probabilidades se mueve en pasos de 0.1, se tendrían 10 valores para *pc* y 10 para *pm*, y supongamos 6 valores para el tamaño de población, por 10 valores para el *gap* generacional, lo cual produce 6000 combinaciones de parámetros a evaluar manualmente. Esto implica una inversión grande de tiempo. Si el programa evolutivo corre en 2 segundos, estas 6000 combinaciones se evaluarían en 3 horas y 20 minutos. Como cada combinación debería ser probada, por lo menos unas 10 veces para tener resultados estadísticos más o menos válidos, pues la población se inicializa aleatoriamente, produciría unas 33 horas invertidas por el desarrollador para encontrar una combinación de valores apropiados para los parámetros. Esto no es viable y además se introducirían muchos errores humanos, a menos que se decida realizar un programa que construya las 6000 combinaciones de valores y ejecute automáticamente el programa evolutivo, al menos 10 veces por cada combinación. Adicionalmente, por la restricción de limitar las variaciones de los parámetros a un paso específico (por ejemplo, cada 0.1 en las probabilidades), se tiene el inconveniente de no encontrar mejores resultados en otros valores del paso.

Se propone como solución al problema de entonar los parámetros de control, una herramienta automatizada enfocada en la entonación de tres parámetros que normalmente rigen la eficiencia de un programa evolutivo [5]: la probabilidad de cruce (*pc*), la probabilidad de mutación (*pm*) y el tamaño de población (*pob*); así como también incluir el *gap* generacional como otro parámetro importante para la dinámica del programa evolutivo. Esto se realiza con un enfoque de meta-evolución donde el algoritmo genético del nivel superior tiene la libertad de colocar cualquier combinación de valores de los parámetros en los rangos donde éstos se mueven, sin depender de un paso particular, para tratar de optimizar los resultados de programas evolutivos de nivel inferior. El objetivo de la herramienta es automatizar el proceso de búsqueda de los valores de estos parámetros con el fin de propiciar la convergencia del programa evolutivo considerado. Es importante hacer notar que no se está planteando la búsqueda de una parametrización estándar para programas evolutivos, entendiéndose ésta como la obtención de un conjunto de valores de los parámetros que tenga un comportamiento aceptable de manera general para cualquier programa evolutivo.

Este esquema de meta-evolución es implantada mediante una arquitectura paralela maestro-esclavos sobre un cluster con tecnología OpenMosix [27], que permite mejorar el tiempo de respuesta; pues las distintas instancias (con diferente combinación de valores de parámetros) de un mismo programa evolutivo, representan procesos independientes que no se comunican entre sí, de manera que se presenta el caso ideal de paralelismo. La arquitectura maneja, de manera transparente, el balanceo de la carga computacional. En este artículo se presenta la concepción, desarrollo y prueba de tal herramienta automatizada. Se realizaron pruebas con algunos programas evolutivos representativos, comenzando con un algoritmo genético (que tiene gap generacional) para la optimización de una función y prosiguiendo con programas evolutivos para la resolución del problema del agente viajero con 30 y 75 ciudades.

El resto de este artículo se ha estructurado como sigue: la segunda sección describe los Fundamentos Teóricos de este trabajo; en la tercera sección se presentan los Trabajos Relacionados como antecedentes de esta investigación; en la cuarta sección se presenta la Descripción de la Herramienta desarrollada; la quinta sección, se concentra en los Resultados Experimentales obtenidos; en la sexta sección se describen las Conclusiones y Trabajos Futuros que se esperan realizar dando continuidad a la investigación aquí reportada.

2. FUNDAMENTOS TEÓRICOS

La *Computación Evolutiva* es un conjunto de algoritmos estocásticos de búsqueda basados en abstracciones del proceso de evolución de Darwin [16]. En esta área se han propuesto distintos modelos computacionales denominados *algoritmos evolutivos* cuyo propósito es guiar una búsqueda estocástica, haciendo evolucionar un conjunto de estructuras (representativas de la solución al problema) y seleccionando de modo iterativo las más adecuadas. La mayoría de los algoritmos evolutivos son bastante simples en su concepción, pero suficientemente complejos, en su dinámica, como para proporcionar mecanismos de búsqueda robustos y potentes. Existen aplicaciones construidas con algoritmos evolutivos en diversos campos.

La *población de individuos* de un algoritmo evolutivo, representa un conjunto de candidatos a soluciones de un problema. En cada iteración del algoritmo, esta población es sometida a un proceso de selección para determinar los progenitores de la próxima *generación*, estando favorecidos los individuos más aptos de acuerdo a una *medida de adaptación*. Para generar nuevos individuos que constituyen la población hija, se aplican estocásticamente los *operadores genéticos* (cruce y mutación) sobre los progenitores. Si la población generada contiene los individuos deseados, se termina el proceso iterativo de evolución. También puede finalizar por una cota máxima de iteraciones establecidas.

Los parámetros externos de funcionamiento o *parámetros de control*, guían la dinámica evolutiva del algoritmo. Los más usados son el tamaño de población (*pob*), la probabilidad de cruce (*pc*), la probabilidad de mutación (*pm*), y el número de iteraciones.

Los tipos de algoritmos evolutivos existentes son: *Estrategias de Evolución* [26], *Programación Evolutiva* [11], *Algoritmos Genéticos* [12][17] y *Programación Genética* [19][20]. En este artículo se describen los algoritmos genéticos y los programas evolutivos por ser la base del trabajo reportado.

A pesar que los algoritmos evolutivos son eficientes, algunos problemas complejos requieren de gran cantidad de tiempo de procesamiento para alcanzar una solución satisfactoria. Las configuraciones paralelas permiten reducir el tiempo de respuesta, además, en muchas ocasiones obtienen mejores resultados que los algoritmos seriales.

2.1 Algoritmos Genéticos y Programas Evolutivos

En un algoritmo genético, los individuos son representados por cadenas de bits, llamadas *cromosomas*. Cada cromosoma consta de un número determinado de *genes*. Cada gen está compuesto de uno o varios bits, de acuerdo a la naturaleza del problema. La estructura de un individuo se deriva de la representación genética compuesta de cada uno de los genes que lo conforman. El contenido de un individuo corresponde a su codificación en cadena binaria. A la estructura se le denomina *genotipo* y al valor específico candidato del espacio de soluciones, asociado a esa estructura, se le denomina *fenotipo*. La población se inicializa de manera aleatoria.

El método de selección es proporcional a la *función de adaptación*, utilizando algún mecanismo aleatorio como sorteo, rueda de la ruleta o torneo.

El operador de cruce es el principal en esta técnica evolutiva para la reproducción, el cual realiza un intercambio estructurado de información (segmentos de bits). Actúa sobre parejas de padres y normalmente origina otro par de individuos que combinan características de sus progenitores. Los tipos de cruce más comunes son: el cruce de un punto, el cruce de dos puntos y el cruce uniforme. El operador de mutación, altera la información genética de un individuo, realizando alguna pequeña modificación sobre sus genes.

Para obtener la población hija se pueden seguir los siguientes criterios: reemplazo inmediato, donde todos los descendientes sustituyen a la población progenitora; reemplazo con factor de llenado, donde los descendientes sustituyen a los miembros de la población más similares a éstos; reemplazo por inserción, donde se sustituyen ciertos miembros de la población (ejemplo, los peores por los descendientes); y el reemplazo por inclusión, donde se agrupan los descendientes con los progenitores en una sola población y de ella se toman los miembros mejores.

Un criterio importante que se toma en cuenta al diseñar un algoritmo genético es el criterio de parada, que permite concretar las condiciones que indican que el algoritmo genético ha encontrado una solución aceptable, o en su defecto, ha fracasado en la búsqueda por lo que no tiene sentido continuar.

Los programas evolutivos [23] pueden definirse como algoritmos genéticos especializados [8] que incorporan conocimiento sobre el dominio del problema a estudiar, y donde, en general, es necesario representaciones genéticas no binarias. Éstos surgieron al resolver problemas de optimización paramétrica en los que se requería alta precisión; por lo que la representación binaria obligaba a utilizar individuos de tamaño muy grande, y esto incrementaba fuertemente los requerimientos de cálculo y almacenamiento. Los resultados mejoran significativamente [23] usando vectores de números reales donde cada componente representa un gen. En este tipo de programas, el operador de mutación consiste en alterar levemente el valor de una de las componentes del individuo y el operador de cruce produce dos individuos que de alguna manera promedian los valores de sus progenitores. Esto añade significado a los genotipos y a los operadores, lo cual incide favorablemente en la capacidad de procesar una mayor variedad de individuos útiles.

2.2 Meta-evolución

Entre las técnicas más avanzadas de la Computación Evolutiva están los enfoques de meta-evolución [2][10], que desarrollan un esquema de dos niveles para resolver un problema. El nivel superior contiene un algoritmo evolutivo que optimiza una población de algoritmos evolutivos ubicados en el nivel inferior. Esta población de individuos representa instancias de un algoritmo evolutivo que pretende solucionar un problema específico. Cada uno de los algoritmos de nivel inferior se ejecuta en forma independiente para producir una solución del problema particular. El valor de adaptación de esta solución es considerado en la operación del algoritmo de nivel superior. Los números de generaciones creadas en los dos niveles son independientes uno del otro. El algoritmo de nivel inferior con mayor valor de adaptación es considerado como el mejor algoritmo evolutivo para el problema específico.

La meta-evolución usualmente requiere una gran cantidad de cómputo, sin embargo, se pueden desarrollar implementaciones paralelas para que se ejecute en un tiempo razonable.

2.3 Configuración Paralela: Maestro-esclavos

En el enfoque maestro-esclavos, un proceso maestro controla procesos esclavos con las instancias de algún programa. El maestro determina en qué momento se ejecutan los esclavos. Al terminar éstos generan información útil para que el maestro realice nuevas operaciones. La meta-evolución aprovecha esta configuración paralela de manera que el maestro implementa el algoritmo genético de nivel superior, el cual optimiza instancias de un programa evolutivo, ubicadas en el nivel inferior.

En la actualidad, la computación paralela puede realizarse mediante tecnologías que permiten a múltiples computadoras trabajar juntas para resolver problemas, éste es el concepto de *Cluster Computing* ó cómputo grupal. Los clusters pueden clasificarse en dos paradigmas [4]. El primero consiste en la emulación del comportamiento de computadoras paralelas, mediante el uso de librerías externas, estos clusters requieren que las aplicaciones sean modificadas para aceptar instrucciones paralelas. El otro paradigma es el de sistemas altamente escalables, conocido como SSI, por sus siglas en inglés: *Single System Image*. En este caso, las aplicaciones no requieren cambiar para tomar ventaja del cómputo paralelo, pues el sistema realiza automáticamente la configuración de nodos heterogéneos y el balanceo de la carga entre nodos.

En este trabajo se usó un cluster SSI con tecnología OpenMosix [27]. Esta tecnología usa técnicas adaptativas para el balanceo de la carga computacional, donde los procesos en ejecución son migrados transparentemente a otros nodos del cluster para mejorar su desempeño. La velocidad a la cual los nodos pueden comunicarse está determinada por la velocidad de la red local que los conecta. OpenMosix no es capaz de paralelizar un simple proceso, pero si permite la ejecución paralela de diversos procesos de manera eficiente. Si una aplicación genera muchos procesos hijos, donde cada uno realiza un trabajo, entonces OpenMosix será capaz de migrar cada uno de estos procesos a un nodo apropiado dentro del cluster. Esto lo hace muy adecuado para una configuración maestro-esclavos. A pesar que el proyecto OpenMosix fue cerrado en marzo de 2008 [24], está todavía disponible para quieres quieran usarlo.

3. TRABAJOS RELACIONADOS

En la Computación Evolutiva, el problema de encontrar los valores óptimos de los parámetros de control ha sido objeto de estudio e investigación tanto teórica como experimentalmente [3][6]. Desde el punto de vista teórico, una parametrización óptima universal no existe, ya que los valores óptimos dependen en gran manera del problema de optimización para el cual el algoritmo es aplicado [15]. No obstante, algunos trabajos teóricos han propuesto, por ejemplo, que en un algoritmo genético, la probabilidad de mutación óptima es el inverso de la longitud del cromosoma [1].

Algunos experimentos se han enfocado en conseguir los parámetros óptimos para algoritmos genéticos, a través de la medición del desempeño de algoritmos probados con diferentes funciones objetivo con diversas características. Un primer estudio [7] obtuvo los siguientes valores pm = 0.001, pc = 0.6, $50 \leq pob \leq 100$, y un segundo estudio [25] produjo pm $\in$ [0.005,0.01], pc $\in$ [0.75,0.95], y $20 \leq pob \leq 30$. En el caso de los algoritmos genéticos, parametrizaciones que han mostrado experimentalmente que funcionan para la mayoría de los problemas se han adoptado como estándares [7].

También se han hecho experimentos de meta-evolución [2] para obtener valores óptimos de parámetros de algoritmos genéticos. Uno de estos experimentos [14] obtuvo los valores entonados: pc=0.95, pm=0.01, pob=30, gap=1.0. Este experimento se basó en el trabajo previo [7], usando las funciones de prueba para los algoritmos genéticos del nivel inferior y los valores de parámetros propuestos para el nivel superior. Otra experiencia previa [9] de meta-evolución con algoritmos genéticos, en ambos niveles, fue realizada para resolver un problema particular: optimización de pesos de una red neuronal artificial. Allí se consideró no sólo la optimización de parámetros, sino también de decisiones.

En el trabajo [2], el algoritmo de nivel superior combina estrategias de evolución y algoritmos genéticos. Los principales objetivos del experimento se centraron en probar cómo funcionaba la técnica híbrida para espacios de solución mixtos y buscar parámetros óptimos estándares que mejoraran la eficiencia de los ya existentes. Un aporte importante de esta experiencia [2] fue la idea de dejar semillas fijas para los algoritmos a optimizar. También propuso el uso de paralelismo basándose en el esquema de maestro-esclavos, debido al gran esfuerzo computacional requerido para evaluar la población de individuos, donde cada individuo es la representación de un algoritmo genético.

Se ha propuesto una metodología [18], que automatiza el proceso de entonación de los parámetros de un algoritmo genético. Ésta se basa en un enfoque de meta-evolución, con un algoritmo genético en el nivel superior, combinado con una regresión por soporte de vectores en su función de adaptación, la cual modela las interacciones de los parámetros de control del algoritmo genético de nivel inferior y predice el desempeño de este algoritmo ante diferentes combinaciones de los parámetros.

Un trabajo bien particular [22] usó meta-evolución con algoritmos genéticos para probar cuál operador de cruce es "mejor" en un TSP (*Travelling Salesman Problem*) y constatar si es mejor la utilización de diferentes operadores por generación en lugar de un único operador. Como resultado se obtuvo criterios para seleccionar operadores de cruce en este problema. Los resultados indicaron que en efecto, utilizar operadores diferentes en diferentes generaciones, es mejor que aplicar un único operador todo el tiempo.

En cuanto al uso de meta-evolución para optimizar programas evolutivos, es poco lo que se ha hecho previamente. Un trabajo anterior [21] presentó un híbrido entre algoritmos genéticos y estrategias de evolución para optimizar los parámetros de un programa evolutivo específico denominado PrismLens Applet.

Basado en estas experiencias previas, se ha construido una herramienta genérica para la entonación de parámetros de control de programas evolutivos. Ésta hace uso del enfoque de meta-evolución con un algoritmo genético de nivel superior. Su implementación se realizó con una configuración paralela maestro-esclavos. Esta herramienta constituye el aporte de la investigación reportada en este artículo.

4. DESCRIPCIÓN DE LA HERRAMIENTA

El corazón de la herramienta lo constituye un algoritmo genético de nivel superior que controla la meta-evolución. Los parámetros de control (*pc*, *pm*, *pob*, *gap*) y el número de iteraciones para este algoritmo se reciben como entrada mediante una interfaz web de usuario final, la cual se encarga de validarlos, de manera que cumplan con los rangos establecidos. Tanto estos parámetros, como los operadores genéticos son fijos durante la ejecución del algoritmo, además son independientes del programa evolutivo de nivel inferior.

El programa evolutivo de nivel inferior, a entonar, se le pasa a la herramienta como un archivo ejecutable, el cual debe ser correcto y compatible con la plataforma donde ella esté alojada. Dicho programa escribe en un archivo de texto su salida, a saber: el número de iteraciones realizadas y el valor de adaptación promedio de su última generación. Desde el shell del sistema operativo, una llamada al programa evolutivo sería de la forma `pe pc pm pob gap sem result`, donde `pe` es el nombre del ejecutable, `pc`, `pm`, `pob` y `gap` son valores numéricos correspondientes a los parámetros de control, `sem` es un valor entero largo sin signo correspondiente a la semilla y `result` es un string con la ruta y nombre del archivo donde el programa colocará su salida.

El objetivo del algoritmo genético de nivel superior es optimizar la función calculada por el programa de nivel inferior, correspondiente al valor de adaptación promedio de la última generación de la instancia del programa evolutivo. Ésta es la función de adaptación de la meta-evolución, es decir de un individuo del algoritmo genético de nivel superior.

Se usa una misma semilla que es pasada a todas las instancias del programa evolutivo, es decir, para todos los individuos de la población del algoritmo genético superior, de manera que ellos corran en condiciones de igualdad. Dicha semilla puede ser suministrada por el usuario o generada aleatoriamente por la herramienta.

Cada individuo del algoritmo genético de nivel superior es caracterizado por un cromosoma que codifica los parámetros de control (pc, pm, pob y gap) para el programa de nivel inferior.

Un cromosoma está compuesto de 32 bits, divididos en grupos que representan cada parámetro a entonar. La codificación genética de un cromosoma se muestra en la Figura 1. El primer grupo de siete bits contiene la probabilidad de cruce, un valor real con dos cifras significativas. El segundo grupo de diez bits contiene la probabilidad de mutación, un valor real con tres cifras significativas. El tercer grupo de ocho bits contiene el tamaño de la población, un valor entero. El cuarto grupo de siete bits contiene el gap generacional, un valor entero. Los rangos de valores de estos parámetros son: probabilidad de cruce ($0.00 \leq pc \leq 1.00$), probabilidad de mutación ($0.000 \leq pm \leq 1.000$), tamaño de la población ($6 \leq pob \leq 255$) y gap generacional ($5 \leq gap \leq 100$).

pc	*pm*	*pob*	*gap*
1110101	1010111101	11111101	0101110

Figura 1: Representación Genética de un Cromosoma

Para cada instancia del programa evolutivo se genera un nombre de archivo diferente donde éste coloca sus resultados, los cuales son leídos por el algoritmo genético de nivel superior. Finalmente, el algoritmo genético genera un reporte que también es escrito en un archivo de texto. El formato del archivo puede observarse en la Figura 2.

GENERACION ACTUAL: 500

Mejor adaptacion: 1.9331504106521606445312500 Adaptacion promedio: 1.9326910972595214843750000

No	Cromosoma	pc	pm	pob	gap	Adaptación	T.
1	11100000000001100011110010000110	0.88	0.013	124	9	1.9331504106521606445312500	109
2	11110100000001100011110010000110	0.96	0.013	124	9	1.9331504106521606445312500	119
3	11100110000001100011110100000110	0.91	0.013	126	9	1.9331504106521606445312500	131
4	11101000000001100011110100000110	0.91	0.013	126	9	1.9331504106521606445312500	131
5	11110100000001100011111100001000	0.96	0.013	130	10	1.9331504106521606445312500	132
6	11100100000001100011110010000110	0.90	0.013	124	9	1.9331504106521606445312500	133
7	11110100000001100011110100000110	0.96	0.013	126	9	1.9331504106521606445312500	133
8	11110100000001100011110110000110	0.96	0.013	126	9	1.9331504106521606445312500	133
9	01111100000001100011110100000110	0.49	0.013	126	9	1.9331504106521606445312500	163
10	11100000000001001011010010001101	0.88	0.010	108	14	1.9331502914428710937500000	71
11	11100110000001001011010010001101	0.91	0.010	108	14	1.9331502914428710937500000	97
12	11111100000001100011110100000110	0.99	0.013	126	9	1.9331502914428710937500000	103
13	11100100000001100011001110001010	0.90	0.013	106	12	1.9331502914428710937500000	103
14	11100100000001100011010010001101	0.90	0.013	108	14	1.9331502914428710937500000	109
15	11100100000001100011011100001001	0.90	0.013	114	11	1.9331502914428710937500000	112
16	11100100000001100011010100001010	0.90	0.013	110	12	1.9331502914428710937500000	115
17	11100100000001100011010110001010	0.90	0.013	110	12	1.9331502914428710937500000	115
18	11110100000001100011010100001010	0.96	0.013	110	12	1.9331502914428710937500000	116
19	00111010000001100011010100001010	0.24	0.013	110	12	1.9331502914428710937500000	122
20	00111100000001100011010100001010	0.24	0.013	110	12	1.9331502914428710937500000	122

Figura 2: Estructura y Contenido del Archivo de Resultados

En la primera línea, un entero que representa la generación final del algoritmo genético de nivel superior. En la segunda línea, dos números reales con 25 cifras decimales, los cuales representan la mejor adaptación y la adaptación promedio de los individuos de esa generación. Además, este archivo contiene para cada individuo de la última generación del algoritmo genético del nivel superior, el cromosoma, su decodificación en los correspondientes parámetros de control, el valor de adaptación de ese individuo y el número de iteraciones realizadas por el programa evolutivo para obtener ese valor de adaptación. Estos individuos están ordenados descendentemente por el valor de adaptación. La Figura 2 muestra sólo los primeros 20 individuos de un archivo de salida de ejemplo.

El mecanismo de selección utilizado por el algoritmo genético de nivel superior es el de rueda de la ruleta. La población inicial se genera aleatoriamente. Los operadores genéticos son cruce de un punto y mutación. Estas operaciones se realizan a nivel de bits utilizando la operación de *bitwase* provista por el lenguaje de programación. Se garantiza que no se pierde el mejor individuo debido al uso del gap generacional.

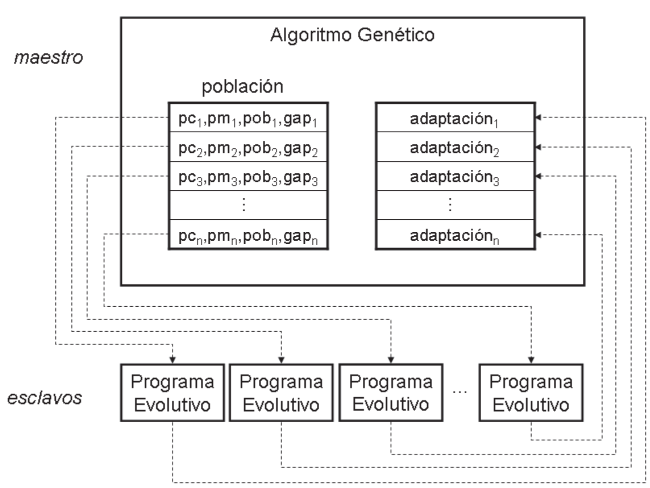

Figura 3: Diagrama de Configuración Paralela

La herramienta corre en un cluster con tecnología OpenMosix, la cual realiza automáticamente el balanceo de la carga computacional en los procesadores del cluster. Con OpenMosix no se necesita saber cuántos procesadores hay disponibles en el cluster para hacer el trabajo. Un proceso puede ser ejecutado en cualquier nodo del cluster y puede migrar de uno a otro de forma transparente.

La configuración paralela utilizada en la herramienta es la conocida como maestro-esclavos. El proceso maestro ejecuta el algoritmo genético de nivel superior de la meta-evolución. Los procesos esclavos son instancias del programa evolutivo, correspondientes a los diferentes individuos de la generación actual del algoritmo genético de nivel superior. Esto se muestra gráficamente en la Figura 3.

El maestro es síncrono, pues espera a que todos los individuos de la población se evalúen antes de seguir a la próxima generación. La comunicación de datos entre el maestro y los esclavos es a través de archivos como se describió anteriormente.

El proceso maestro realiza todas las operaciones del algoritmo genético menos la función de evaluación, la cual se realiza en los procesos esclavos de forma paralela. En la Figura 4 se muestra el algoritmo del proceso maestro. Se especifica el procedimiento evaluar población donde se levanta un proceso para cada instancia del programa evolutivo, correspondiente a cada individuo de la población. Es asimismo en este procedimiento que se espera por la culminación de los procesos esclavos. Los demás pasos del algoritmo genético se dan por sobrentendidos.

5. RESULTADOS EXPERIMENTALES

La herramienta se montó en un cluster que consta de 12 nodos, cada uno con las siguientes características: procesador Pentium IV 1.8 MHz, memoria de 512 MB y disco duro de 40 GB. Los nodos están conectados con Fast Ethernet (100 Mbit/seg). El cluster se encuentra ubicado en la Facultad de Ciencias y Tecnología de la Universidad de Carabobo.

Se escogieron tres programas evolutivos para probar la herramienta. Un programa evolutivo de Optimización de Funciones (Optf), con la función $f(x) = x + |\operatorname{sen}(32x)|$ en el intervalo [0,1], para el problema de buscar el máximo. Un programa evolutivo que resuelve el problema del agente viajero para 30 ciudades (TSP30) y otro para 75 ciudades (TSP75). Con cada uno de estos programas se realizaron dos experimentos.

El objetivo del primer experimento era observar que realmente la herramienta producía los valores adecuados además del efecto de las semillas de inicialización del generador de números pseudo-aleatorios de lenguaje C bajo linux, sobre estos valores. Para tal fin, con cada programa evolutivo se realizaron once corridas. Se usaron tres variantes de semillas de inicialización para el nivel superior y para cada una de éstas se probó con tres semillas distintas en el nivel inferior. En el segundo experimento se realizó un estudio de convergencia de la herramienta, con los programas evolutivos, basado en el promedio de diez corridas distintas. Los valores usados en los parámetros de control del algoritmo genético de nivel superior fueron 0.80 como probabilidad de cruce, 0.002 como probabilidad de mutación, 100 como tamaño de población, 10% como gap generacional [7], 500 iteraciones para el programa OptF, 200 iteraciones para los programas TSP30 y TSP75.

5.1 Programa OptF

Los resultados del primer experimento con el programa evolutivo Optf se muestran en la Tabla 1. Las columnas pc, pm, pob y gap, se refieren a los parámetros de control del programa evolutivo, que fueron hallados por el algoritmo genético de nivel superior. La columna Iteraciones corresponde el número de iteraciones realizadas por el programa evolutivo para producir la mejor adaptación.

Corrida	Semilla AG	Semilla PE	pc	pm	pob	gap	Mejor Adaptación	Iteraciones
01	1083101990	1082251078	0.90	0.010	126	10	1.9331504106521606445312500	107
02	1083101990	1081897119	0.92	0.021	128	7	1.9331504106521606445312500	190
03	1083101990	1080213991	0.78	0.002	124	21	1.9331504106521606445312500	54
04	2129989118	1082251078	0.89	0.006	130	11	1.9331504106521606445312500	113
05	2129989118	1081897119	0.72	0.003	132	16	1.9331504106521606445312500	50
06	2129989118	1080213991	0.80	0.022	134	7	1.9331502914428710937500000	137
07	3981121976	1082251078	0.43	0.031	98	8	1.9331501722335815429687500	190
08	3981121976	1081897119	0.99	0.015	64	15	1.9331495761871337890625000	130
09	3981121976	1080213991	0.88	0.013	124	9	1.9331504106521606445312500	109
10	3981121976	10000	0.48	0.003	130	35	1.9331504106521606445312500	65
11	10000	3981121976	0.20	0.002	140	20	1.9331496953964233398437500	111

Tabla 1: Resultados Experimentales con OptF

```
ag:
       procedimiento evaluar población P
       inicio
              para cada individuo i en P
                     ejecutar en paralelo el esclavo pe(i,s)
              fin para
              esperar que todos los esclavos terminen
       fin evaluar población
inicio
       t := 0
       generar semilla s
       inicializar población P(t) {Tomando ptos del espacio de soluciones}
       evaluar población P(t)
       mientras (no están los individuos deseados o no se cumpla condición de
terminación)
              seleccionar individuos más aptos de P(t)
              obtener P(t+1), mediante reproducción con individuos seleccionados
              evaluar población P(t+1)
              t:= t+1
       fin mientras
fin ag
```

Figura 4: Pseudo Código del Proceso Maestro

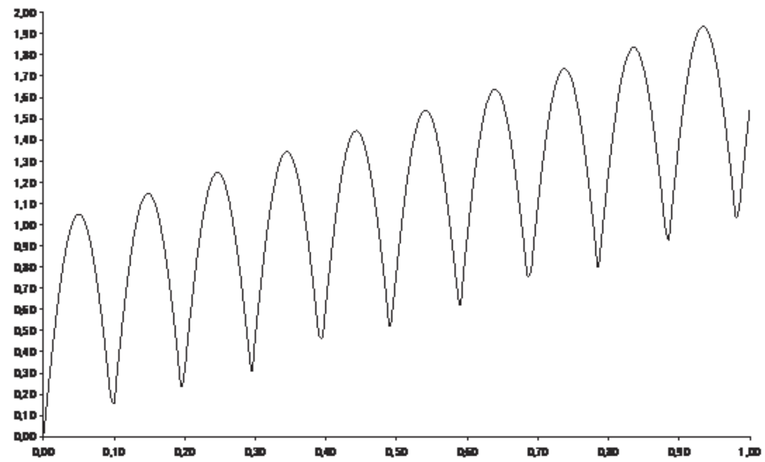

Figura 5: Función f(x) = x + | sen(32x)| en el intervalo [0,1]

Las corridas 1, 2, 3, 4, 5, 9 y 10 produjeron el mejor valor de adaptación encontrado por la herramienta. Para este caso es sencillo verificar que este valor es adecuado. De hecho se está optimizando la función $f(x) = x + |\operatorname{sen}(32x)|$ en el intervalo [0,1]. En la gráfica de la Figura 5 se observa que la mejor adaptación encontrada corresponde con el óptimo de la función.

Las corridas que obtuvieron la mejor adaptación, produjeron distintos valores para los parámetros de control. Esto significa que varias combinaciones de valores de los parámetros pueden permitir alcanzar resultados satisfactorios en un número de iteraciones entre 50 y 190. Ante todas estas combinaciones de valores disponibles para los parámetros, se escogería la que alcanza el mejor valor en el menor número de iteraciones posible; sin embargo, esto se debe a la influencia de las semillas utilizadas, que inicializan la población colocando los cromosomas en lugares del espacio de soluciones, que favorecen o no la rápida convergencia. Por lo tanto, podemos decir que cualquiera de esas combinaciones de parámetros es buena. Para este experimento, excluyendo la corrida 10, se puede concluir que los mejores valores, para los parámetros de control, están en los rangos

pc $\in$ [0.72, 0.92], pm $\in$ [0.002, 0.021],

pob $\in$ {124,..,132} y gap $\in$ {7,..,21}.

Dichos rangos corresponden con los valores de parámetros obtenidos en estudios previos [7]. En cuanto a la corrida 10, aunque se obtiene el mejor valor de adaptación, dos de los parámetros están fuera de estos rangos. Analizando este caso en detalle se pudo concluir que esto se debe al efecto de la semilla escogida para esta corrida.

Corrida	Semilla AG	Semilla PE	pc	pm	Mejor adaptación	Iteraciones
01	1083101990	1082251078	0.81	0.968	0.0023752963170409202575684	68
02	1083101990	1081897119	1.00	0.999	0.0023752963170409202575684	63
03	1083101990	1080213991	0.87	0.988	0.0023752963170409202575684	62
04	2129989118	1082251078	0.95	1.000	0.0023752963170409202575684	67
05	2129989118	1081897119	0.95	0.959	0.0023752963170409202575684	66
06	2129989118	1080213991	0.82	0.964	0.0023752963170409202575684	68
07	3981121976	1082251078	0.92	0.904	0.0023752963170409202575684	66
08	3981121976	1081897119	0.86	0.878	0.0023752963170409202575684	65
09	3981121976	1080213991	0.89	0.997	0.0023752963170409202575684	68
10	3981121976	10000	0.86	0.945	0.0023752963170409202575684	66
11	10000	3981121976	0.95	0.916	0.0023752963170409202575684	62

Tabla 2: Resultados Experimentales con TSP30

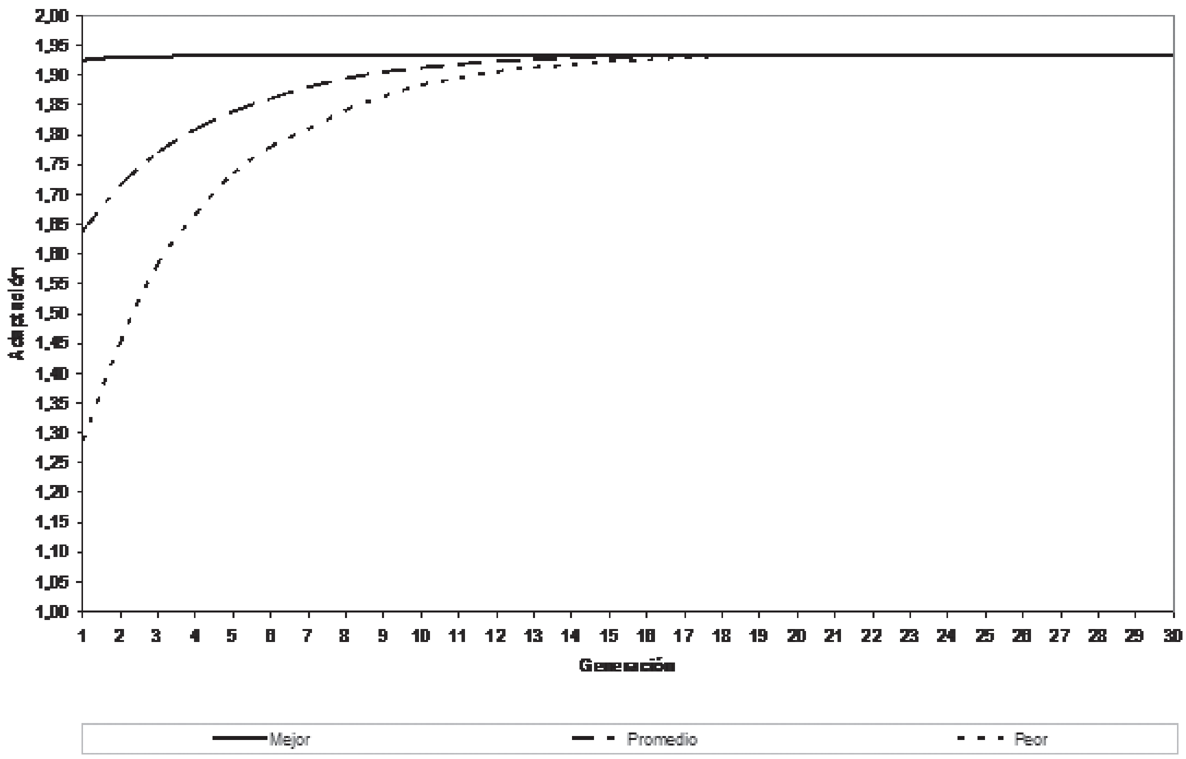

Figura 6: Gráfica de convergencia para OptF

Para el segundo experimento, correspondiente al estudio de convergencia de la herramienta, las corridas arrojaron que ésta efectivamente converge. En la Figura 6, se observan los valores de adaptación (mejor, promedio y peor) para las 30 primeras generaciones del algoritmo genético de nivel superior. En ella se evidencia que tanto la curva de la adaptación promedio, como la curva de peor adaptación se van acercando a la curva de la mejor adaptación. Este comportamiento se mantiene para el resto de las generaciones observadas.

5.2 Programa TSP30

Los resultados del primer experimento obtenidos con el programa evolutivo TSP30 se resumen en la Tabla 2. En este caso las once corridas arrojaron un mismo valor para la mejor adaptación. Es de hacer notar que este programa evolutivo corresponde a un algoritmo genético celular, con una rejilla regular de 12x12 casillas, obteniendo 144 individuos o cromosomas (población total fija); se utilizó la vecindad de Moore, equivalente a una población local de 9 cromosomas, para cada celda de la rejilla; el gap generacional no es utilizado porque en cada iteración el cromosoma de la celda central y el cromosoma de una casilla elegida aleatoriamente entre los vecinos, mediante rueda de la ruleta, son los padres y se sustituye el individuo central por el mejor hijo que resulte luego de cruce y/o mutación. Antes de este trabajo de investigación, en muchas corridas del algoritmo genético celular, el mejor valor hallado siempre había sido una ruta del viajero con distancia de 420 Kms, habiendo colocando los valores de los parámetros de manera manual y empírica, con pc=0.70, pm=0.400 y el número de iteraciones entre 100 y 200. La función de adaptación utilizada, para este programa evolutivo (por ser un problema de buscar el mínimo), es: $f(x) = 1/(1 + x)$, donde x corresponde a la distancia de la ruta; por lo tanto $f(420) = 0{,}00237529$.

Las combinaciones de valores de los parámetros, que logran el valor de la mejor adaptación, son todas distintas pero dentro en un rango entre 0,8 y 1.00. Para este experimento, se puede concluir que los mejores valores para los parámetros de control están en los rangos $pc \in [0.81, 1.00]$ y $pm \in [0.878, 1.000]$. Por ello se concluye que este programa evolutivo requiere de altas probabilidades de cruce y altas probabilidades de mutación, por la dificultad del problema. El número de iteraciones resultó en el orden de 70 generaciones.

Existe una diferencia apreciable en la probabilidad de mutación (pm) encontrada por la herramienta y la encontrada manualmente, mientras que la probabilidad de cruce es bastante similar. Las combinaciones de valores de los parámetros halladas por la herramienta convergen en menos iteraciones que la combinación manual. Con altas probabilidades de mutación, la herramienta explora más ampliamente el espacio de soluciones, desde el comienzo.

En la Figura 7 se observan las curvas de convergencia obtenidas en el segundo experimento. La curva de adaptación promedio se acerca rápidamente a la mejor adaptación manteniendo este comportamiento por el resto de las generaciones.

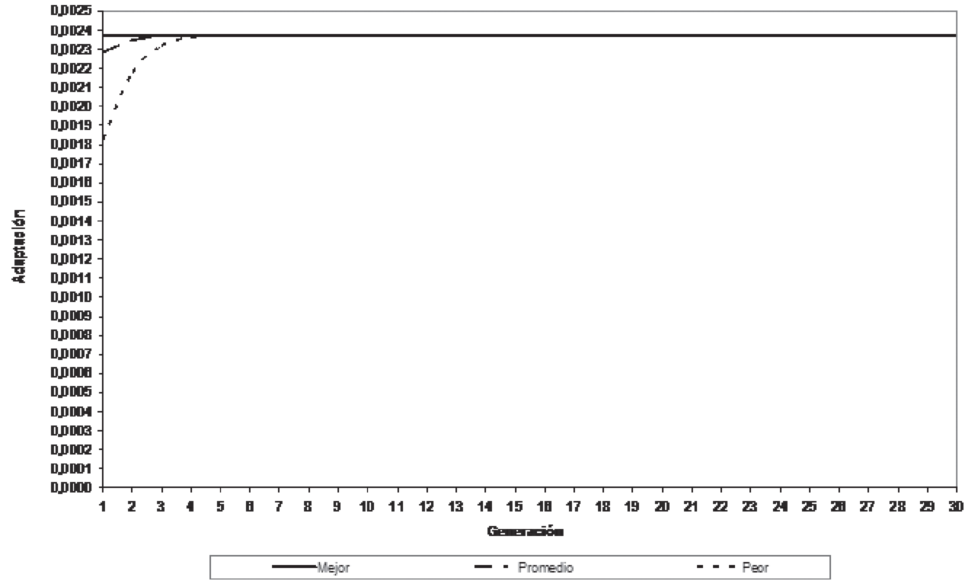

Figura 7: Gráfica de convergencia para TSP30

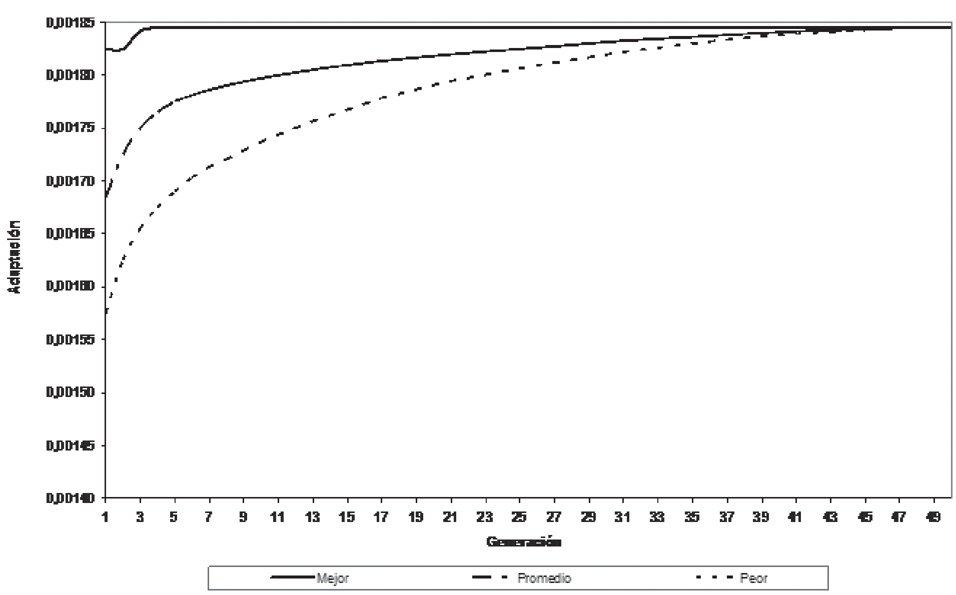

Figura 8: Gráfica de convergencia para TSP75

5.3 Programa TSP75

La Tabla 3 muestra los resultados para las once corridas realizadas en el primer experimento con el programa evolutivo TSP75. Este programa evolutivo corresponde al algoritmo genético celular descrito para el programa TSP30, con un archivo de entrada de 75 ciudades. De antemano, no se conoce el óptimo para esta instancia. Los resultados de la herramienta se compararon con los "mejores resultados" obtenidos manualmente: pc=0.70, pm=0.600 y un número máximo de iteraciones de 1000.

Aunque existen diferencias en los valores obtenidos por la herramienta para la mejor adaptación, éstos son mucho mejores, que los resultados obtenidos con los parámetros encontrados de forma manual (mejor adaptación igual a 0.00168350168350168350168835). En cuanto a los rangos de valores obtenidos, pc $\in$ [0.08, 0.21] y pm $\in$ [0.729, 0.993], se concluye que la probabilidad de cruce debe ser baja y la probabilidad de mutación alta. El problema del TSP con 75 ciudades resulta bastante complejo, por eso se requiere explorar muchas áreas del espacio de soluciones con una probabilidad de mutación alta. Lo cual se constató realizando corridas adicionales del programa. Las probabilidades de mutación más bajas (corrida 1 y 6), y la probabilidad de cruce con valor medio (corrida 10), se descartaron por razones estadísticas. En cuanto al análisis de convergencia realizado con el segundo experimento, se obtienen resultados similares a los obtenidos con los programas evolutivos precedentes. Las curvas pueden observarse en la Figura 8.

Como resultado de este trabajo de investigación se obtuvo la herramienta automatizada, para la entonación de parámetros de programas evolutivos. Esta herramienta usa técnicas avanzadas de la computación evolutiva, pues utiliza un enfoque de meta-evolución, para tratar de optimizar una técnica evolutiva (programa evolutivo) por otra técnica de este mismo tipo (algoritmo genético). Los valores obtenidos, en la entonación de los cuatro parámetros: *pc*, *pm*, *pob* y el *gap*, para los programas evolutivos probados, les permitieron converger eficientemente a una buena solución. En cada caso, la herramienta determinó los rangos de valores válidos, para los parámetros de control estudiados, que permiten un buen desempeño de los respectivos programas evolutivos. Estos rangos son dependientes del programa evolutivo particular.

Corrida	Semilla AG	Semilla PE	pc	pm	Mejor adaptación	Iteraciones
1	1083101990	1082251078	0.21	0.534	0.0018382368143647909164429	1000
2	1083101990	1081897119	0.10	0.987	0.0018248150590807199478149	1000
3	1083101990	1080213991	0.12	0.779	0.0018315019551664590835571	1000
4	2129989118	1082251078	0.10	0.939	0.0018382368143647909164429	1000
5	2129989118	1081897119	0.10	0.993	0.0018450181232765316963196	1000
6	2129989118	1080213991	0.10	0.689	0.0018416206585243344306946	1000
7	3981121976	1082251078	0.08	0.938	0.0018315485212951898574829	1000
8	3981121976	1081897119	0.08	0.837	0.0018304786644876003265381	1000
9	3981121976	1080213991	0.13	0.956	0.0018214910523965954780579	1000
10	3981121976	10000	0.53	0.812	0.0018450181232765316963196	1000
11	10000	3981121976	0.13	0.729	0.0018281524535268545150757	1000

Tabla 3: Resultados Experimentales con TSP75

6. CONCLUSIONES Y TRABAJOS FUTUROS

La herramienta tiene la ventaja de tratar de optimizar el desempeño de programas evolutivos, sin considerar la naturaleza del problema que resuelvan; el respectivo programa evolutivo sólo debe cumplir que se pueda ejecutar en la plataforma en la que está desarrollada la herramienta (linux), y que se pueda correr según los requerimientos del algoritmo genético del nivel superior, esto es, con la transferencia de los parámetros descritos (`pc pm pob gap sem result`) y con la colocación de sus resultados (adaptación promedio de la última generación y número de iteraciones de la corrida) en un archivo de salida con un formato pre-establecido.

Además, se observa que los resultados obtenidos son mejores y se producen en menor cantidad de iteraciones respecto a los generados manualmente por los diseñadores de los programas evolutivos probados. Todos los programas evolutivos considerados convergieron a una solución adecuada después de un número razonable de iteraciones.

Es importante resaltar que el diseñador del programa evolutivo debe realizar varias corridas en la herramienta, antes de decidir que rangos de valores son válidos para los parámetros de control de su programa. Esto se debe a la influencia de la aleatoriedad en la inicialización de la semilla, tanto para el algoritmo genético de nivel superior como el programa evolutivo de nivel inferior, lo cual incide directamente en los valores producidos por la herramienta.

El uso de una arquitectura paralela reduce significativamente el tiempo para obtener los resultados. Esta herramienta puede ser implementada en máquinas secuenciales, sin embargo, es claro que para programas evolutivos complejos, con mucha carga computacional, el tiempo de respuesta se incrementaría.

Para trabajos futuros se contempla aumentar el número de parámetros a ser entonados, entre los que estarían la cantidad de operadores de cruce, mutación y sus respectivas probabilidades de selección. Además, se podría estudiar cuáles operadores evolutivos son los más usados por los desarrolladores e integrarlos al proceso de entonación automatizada. Se puede mejorar la interfaz de la herramienta para dar mayor comodidad al diseñador.

Para estudiar mejor el desempeño de la herramienta sería necesario probar la meta-evolución en una arquitectura secuencial, y de esta manera, comparar de manera concreta, con los resultados del sistema paralelo.

7. AGRADECIMIENTOS

Los autores quieren agradecer a los estudiantes de la carrera de Computación en la Facultad Experimental de Ciencias y Tecnología de la Universidad de Carabobo, que han contribuido gentilmente con éste y otros proyectos realizados, particularmente a Andrés Barrios y Katherine Zaoral. Asimismo, agradecemos a Aquel que nos fortalece y conduce en ésta y todas las actividades de nuestra vida: Jesucristo (1 Timoteo 1:12).

8. REFERENCIAS BIBLIOGRÁFICAS

[1] B. Thomas. "The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. Parallel Problem Solving from Nature". PPSN II (Conference Proceedings), pp. 85-94. 1992.

[2] B. Thomas. "Parallel Optimization of Evolutionary Algorithms, Parallel Problem Solving from Nature". PPSN III (Conference Proceedings), pp. 418-427, 1994.

[3] B. Thomas, H.P. Schwefel. "An overview of Evolutionary Algorithms for Parameter Optimization", Evolutionary Computation, Vol. 1, No. 1, pp. 1-23, 1993.

[4] B. Moshe. *HPC Computing Applied to Business Aplications*, Qlusters Inc.

[5] Candidato Nro. 73390, "Measuring the effect of population size in the efficiency of a genetic algorithm", Curso de Vida Artificial, En: *Cognitive and Computing Sciences*, Universidad de Sussex, 2002.

[6] G. Cantor, "Asignación de Parámetros en los Algoritmos Evolutivos". RE'TAKVN, Vol. 1, No. 1, pp. 60-67. 2008.

[7] K. A. De Jong, *An analysis of the behaviour of a class of genetic adaptive systems*, PhD thesis, Universidad de Michigan, 1975.

[8] Programas de Evolución y Algoritmos Genéticos. A. Djordjalian, Indicart Carteles Electrónicos. 20 de agosto 2011, Disponible: <http://indicart.com.ar/ga/epyga.htm>.

[9] B. Freisleben, M. Härtfelder, "Optimization of genetic algorithms by genetic algorithms", Artificial Neural Nets and Genetic Algorithms, pp. 392-399, 1993.

[10] B. Freisleben. "Meta-Evolutionary Approaches". En: *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[11] D. B. Fogel. "An evolutionary approach to the travelling salesman problem". Biological Cybernetics. Vol. 60. No. 2. pp. 139-144, 1988.

[12] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, 1989.

[13] A. Gómez, M. Cárdenas. *Algoritmos Genéticos*. Ministerio de Ciencia e Innovación. Gobierno de España, Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas. 2010.

[14] J. J.Grefenstette, "Optimization of control parameters for genetic algorithms". IEEE Transactions on Systems, Man and Cybernetics. SMC. Vol. 16. No. 1. pp. 122-128. 1986.

[15] W. E. Hart, R. K..Bellew. “Optimizing an arbitrary function is hard for the genetic algorithm”. Proceedings of the 4th conference on Genetic Algorithms. pp. 190-195, 1991.

[16] J. Hernández, M. Ramírez, C. Ferri, *Introducción a la Minería de Datos*, Prentice Hall, 2008.

[17] J. H. Holland. *Adaptation in natural and Artificial Systems*. University of Michigan Press, 1975. ISBN-10: 0-262-08213-6

[18] J. Jaramillo. *Metodología de optimización de los parámetros de control de un algoritmo genético*. Universidad Nacional de Colombia, Departamento de Ingenieria Eléctrica, Electrónica y Computación. 2007.

[19] J. R. Koza, *Genetic programming: on the programming of Computers by means of natural selection*, MIT Press, 1992.

[20] J. R. Koza, *Genetic programming II: automatic Discovery of reusable proggrams*, MIT Press, 1994.

[21] Optimization of a Prism Lens Applet via a Biological Model of Evolution. J. Leung, K. Kern., University of Calgary. 20 de marzo de 2003. Disponible: <http://www.cpsc.ucalgary.ca/kern/533/Main.html>.

[22] M. Mernik, M. Crepinsek, V. Zumer. “A Meta Evolutionary Approach in Searching of the Best Combination of Crossover Operators for the TSP”. Neural Networks Nn’2000: Proceedings of the lasted International Conference. Pittsburgh, Pennsylvania. USA: ACTA Press. pp. 32-36. 2000.

[23] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1992.

[24] Open Mosix Project, OpenMosix Official Page, consultado el 19 de agosto de 2011, Disponible: <http://openmosix.sourceforge.net/>.

[25] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, R. Das. “A study of control parameters affecting online performance of genetic algorithms for function optimization”. Proceedings of the 3th conference on Genetic Algorithms, pp. 51-60, 1989.

[26] H.P Schwefel. *Evolution and Optimun Seeding*, Wiley Inc. 1995.

[27] OpenMosix Wikipedia. Wikimedia Project. Consultado el 19 de agosto de 2011. Disponible: <http://es.wikipedia.org/wiki/OpenMosix>.