Distribución equilibrada del esfuerzo de cómputo en Algoritmos Genéticos

José Ignacio Hidalgo
Universidad Complutense de Madrid
C/ Profesor García Santesmases s/n
28040, Madrid - Spain
hidalgo@dacva.ucm.es

Francisco Fernández de Vega
Grupo de Evolución Artificial –
Dept. de Informática
Centro Universitario de Mérida
C/ Sta Teresa de Jornet, 38. 06800 Mérida
http://atc.unex.es/pacof
fcofdez@unex.es

Resumen

Es difícil encontrar un equilibrio entre el número de ejecuciones, el tamaño de las poblaciones y las generaciones a computar cuando se utiliza un Algoritmo Evolutivo y se precisa ahorrar recursos de computación. Podemos encontrar en la literatura del dominio trabajos que estudian el tamaño de las poblaciones y el número de éstas, pero es difícil encontrar artículos que estudien los tres parámetros anteriores simultáneamente. En este artículo estudiamos todos ellos a la vez. Consideramos el esfuerzo de computación para evaluar diferentes alternativas. Los resultados experimentales confirman algunas de las conclusiones descritas en trabajos previos que consideraban dos de los tres componentes que aquí se consideran. A la vez ofrecemos algunas ideas para distribuir adecuadamente los recursos de computación al diseñar implementaciones paralelas de los Algoritmos Genéticos.

Palabras clave— Algoritmos Genéticos, Esfuerzo de computación, paralelismo.

1. Introducción

Uno de los aspectos más característicos de los algoritmos evolutivos es su facilidad para ser paralelizados. Dado que la evolución en la naturaleza actúa sobre una población completa, podemos considerarla un proceso altamente paralelo [14].

Se ha establecido que la eficiencia de los Algoritmos Genéticos (AGs) está influida notablemente por el tamaño de la población. Con poblaciones grandes, la diversidad genética se incrementa, y es más probable que el algoritmo consiga encontrar el óptimo global. Sin embargo, una población grande también requiere mayor cantidad de memoria que una pequeña, y a la vez mayor tiempo de cálculo y a veces de convergencia. Uno de los primeros pasos consiste en decidir el tamaño de la población y número de poblaciones que se van a utilizar.

Existen muchos trabajos que estudian el tamaño de las poblaciones y el número óptimo a utilizar. Uno de los últimos, presentado por Cantú-Paz y Goldberg [7] presentó un estudio analítico sobre los beneficios de la utilización de una sola población de gran tamaño, frente a múltiples poblaciones aisladas, que pueden considerarse ejecuciones independientes, de tamaño más pequeños. Los autores establecieron un límite fijo de esfuerzo de cómputo y analizaron la calidad esperada en las soluciones. La conclusión es que para problemas difíciles, es mejor utilizar una sola ejecución, del tamaño más grande posible, en lugar de múltiples ejecuciones (múltiples poblaciones aisladas) con poblaciones más pequeñas.

En trabajos como el anterior, los investigadores tratan de establecer un equilibrio entre el número de ejecuciones y el número de individuos totales a utilizar. Por otro lado, también en ocasiones, los investigadores tratan de equilibrar el número de generaciones a calcular, y el número de ejecuciones (que podrían considerarse de nuevo poblaciones aisladas que trabajan en paralelo). Este ha sido el caso de un trabajo reciente realizado en PG por Luke [17].

Actas del IV Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB2005, pp.365-372
ISBN: Vol I:84-9732-440-4 Vol II: 84-9732-467-6

En este artículo tratamos de estudiar todos estos elementos a la vez: considerando un límite fijo de esfuerzo de computación disponible, estudiamos como equilibrar el número de ejecuciones, número de individuos por ejecución y número de generaciones que vamos a calcular, y tratamos de observar experimentalmente la mejor combinación de los tres parámetros simultáneamente, empleando un para de problemas de test.

El resto de este artículo se estructura del siguiente modo: la sección 2 presenta una revisión de trabajos previos en el área. La sección 3 explica los experimentos que se realizan y la forma en que hemos medido el esfuerzo de computación. Las secciones 4 y 5 contienen un análisis experimental de los resultados, y finalmente la sección 6 muestra nuestras conclusiones y trabajo futuro.

2. Trabajos previos

Existen varios trabajos en la línea descrita en la sección previa, algunos de ellos contradictorios, que tratan de establecer un número óptimo de ejecuciones y el número de individuos tanto en los modelos secuenciales como paralelos de los algoritmos evolutivos. En esta sección revisamos los trabajos más relevantes.

Goldberg y algunos otros investigadores han publicado varios trabajos que estudian el tamaño de la población y algunos parámetros relacionados para los Algoritmos Genéticos [5, 8, 13, 15 y 20]. Algunas de las conclusiones alcanzadas en estos trabajos son las siguientes:

- El tamaño de la población que maximiza el número de esquemas procesados suele ser pequeño cuando el fitness se evalúa secuencialmente.
- Cuando un problema requiere evaluar soluciones en paralelos para ganar tiempo, es preferible utilizar tamaños de población grandes.
- Es posible estimar el tamaño de la población estudiando la variación del fitness.
- Es posible obtener un límite conservador sobre la calidad de la convergencia en los Algoritmos Genéticos.
- La calidad de las soluciones depende de la probabilidad de elegir los *bloques de construcción* correctos.
- Si se eligen *bloques* incorrectos en la primera generación, podría ser imposible para un AG llegar a una solución de calidad.
- Puede utilizarse un modelo para predecir el ritmo de convergencia de un AG basándonos en el tamaño de la población. Incluso puede decidirse el tamaño para conseguir una determinada calidad para problemas como el *one-max* y *función trampa* [15].

También se han publicado trabajos que emplean un algoritmo *microgenético*, consistente en una población de tamaño muy pequeño en un algoritmo genético. Para conseguir buenos resultados, lo que hace es reemplazar a menudo la población completa (excepto el mejor individuo), con una nueva población cuando la anterior ha convergido. Esta técnica se conoce también como técnica de elitismo y reinicio. Adoptando una reducción de tamaño de 100 a 5 individuos, es posible reducir el número de evaluaciones requeridas más de un orden de magnitud [4].

Una técnica similar ha sido utilizada por Hidalgo para implementar un operador de *regeneración* y reducir el tamaño de la población [12]. Con este nuevo operador, el algoritmo detecta si el mejor individuo es el mismo durante 10 generaciones consecutivas, y en este caso se elimina la cuarta parte de la población, que es sustituida por nuevos individuos generados al azar.

Gao ha mostrado que el problema del tamaño de la población puede estudiarse mediante técnicas tomadas de la teoría del aprendizaje computacional. Utilizando estas ideas, es posible establecer dos límites inferiores del tamaño de población, que permiten utilizar el principio de paralelismo implícito de los Gas, y relacionar el tamaño de la población con otros parámetros de control [19, 21].

En otros trabajos, Hernández-Aguirre y otros, han desarrollado un esquema denominado *PAC-learning*. La idea es inducir el aprendizaje mediante un conjunto de ejemplos. Los autores utilizan PAC para derivar el tamaño de la población que con mayor probabilidad contendrá al menos un individuo cercano a la solución objetivo [1].

Hay pocos trabajos sobre AGs con tamaños de población variables. Eiben y otros, evaluaron un tamaño de población variable. Sus conclusiones indican que incorporando estos mecanismos

basados en el tiempo de vida en los AE, se pueden mejorar los resultados [22]. Costa y otros han desarrollado un estudio experimental sobre la variación aleatoria dinámica del tamaño de las poblaciones [16]. También se ha presentado un estudio experimental con poblaciones de tamaño variable [2], en el que se utiliza el concepto de *edad* para determinar el tiempo de vida de cada individuo. Los individuos son eliminados cuando la edad alcanza un límite permitido. Otro modelo, denominado *Adaptive Polulation size GA (APGA)* utiliza una variante del anterior con modelo de estado estacionario, y el tiempo de vida del mejor individuo no se modifica [3]. Por otro lado, un artículo reciente describe poblaciones que eliminan individuos a medida que se evalúan generaciones, ahorrando de ese modo esfuerzo de cálculo [11].

También podemos encontrar estudios sobre tamaño de poblaciones en Programación Genética. Tomassini y otros han estudiado la diversidad en modelos de varias poblaciones en PG. Han presentado un estudio experimental sobre el comportamiento de PG distribuido en islas, que permite ahorrar tiempo de cálculo y mejorar la calidad de las soluciones [18]. Por otro lado, [17] ha estudiado la relación entre múltiples ejecuciones cortas y una única ejecución larga con PG.

Varios trabajos en PG Paralela presentaron con el modelo paralelo de PG conclusiones similares que las obtenidas al utilizar un modelo de poblaciones independientes [9, 10]. Todos observaron que los resultados mejoran con el número de poblaciones utilizadas, aunque empeoran cuando estas reducen su tamaño. También en PG el concepto de *plagas* (eliminación de individuos malos de la población) ha sido aplicado a diferentes problemas, incluso con anterioridad a los AGs, y se han conseguidos ahorros de tiempo de cálculo muy significativos [11].

Para intentar cubrir el mayor número de áreas en esta revisión, también en las estrategias de evolución, se ha analizado el tamaño de la población [6]. Mediante un análisis teórico, se ha estudiado la influencia del tamaño de la población y ciertas técnicas aplicadas a problemas dinámicos, basados en operadores de mutación. El autor concluye que en problemas dinámicos, el tamaño de la población puede ser un parámetro crítico, y es necesario continuar la investigación.

En resumen, es difícil establecer un equilibrio entre el número de ejecuciones, el tamaño de la población y el número de generaciones necesarias para obtener soluciones de calidad cuando se utilizan los Algoritmos evolutivos y se quiere ahorrar esfuerzo de cálculo. Los experimentos y resultados presentados en las siguientes secciones muestran un estudio experimental con AGs, empleando problemas bien conocidos en el dominio. Aunque trabajamos con el concepto de múltiples ejecuciones, debemos tener en cuenta que el modelo es equivalente al modelo paralelo de poblaciones aisladas. El interés del análisis experimental que mostramos, es que en lugar de considerar dos de los tres parámetros importantes (tamaño de población, número de ejecuciones y número de generaciones) como ha sido el caso en estudios previos, consideramos a la vez los tres parámetros y confirmamos los resultados que se habían presentado hasta la fecha con los estudios parciales.

3. Experimentación

Aunque tradicionalmente los resultados en estudios con AEs se presentan comparando la calidad con el número de generaciones necesarias para conseguirla, nosotros utilizamos aquí una medida diferente. Cuando las condiciones del experimento cambian de una generación a otra, o se trata de comparar resultados de dos experimentos realizados en diferentes condiciones, las medidas hay que estudiarlas de otro modo. Por ejemplo, no podemos comparar resultados de diferentes experimentos cuando utilizamos diferentes tamaños de población. En Programación Genética, el problema es todavía mayor dado que los individuos crecen a medida que los experimentos van desarrollándose. En [6, 23] se muestra una nueva forma de comparar los resultados en estas condiciones. Nosotros empleamos aquí una medida similar, que utiliza tanto la calidad de las soluciones como el esfuerzo de cálculo requerido para conseguir esa calidad, medido este esfuerzo como el número total de individuos evaluados hasta una determinada generación. Para calcular esta medida estrictamente creciente, debemos primero calcular el esfuerzo parcial de cálculo en cada generación

y luego calcular el esfuerzo acumulado hasta una generación mediante la suma de los esfuerzos de todas las generaciones previas. Aunque esta medida depende del problema, ya que el coste de evaluación de un individuo es diferente para diferentes problemas, es muy útil para comparar diferentes resultados obtenidos en experimentos de un mismo problema.
A continuación describimos los problemas de prueba que hemos utilizado.

3.1. One Max

El problema OneMax es un problema muy simple que trata de maximizar el número de *unos* en una cadena de bits. Formalmente, este problema puede describirse como la búsqueda de una cadena x={x1,x2,…,xN} con xi∈{0,1}, que maximiza la siguiente ecuación:

$$F\left(\vec{x}\right) = \sum_{i=1}^{N} x_i$$

Para definir una instancia del problema necesitamos solamente decidir el tamaño *n* de la cadena. Dado este valor *n* el óptimo se encuentra cuando el número de *unos* en la cadena es *n*, es decir, cuando todos los bits de la cadena son unos.

3.2. Función trampa

Para realizar los experimentos, se ha trabajado con un problema muy conocido en el dominio de los AGs por su dificultad [8] la función trampa. Es una función cuyos valores dependen del número *u* de bits con valor *1* en el subconjunto *k* de bits de la cadena de entrada. El valor de aptitud mejora a medida que más bits del subconjunto poseen valor *0*, hasta alcanzar el óptimo local; pero el óptimo global máximo se encuentra en el extremo opuesto, cuando todos los bits del subconjunto son *1*. El orden *k* se define como:

$$f(x) = \begin{cases} k - u - d & if \quad u < k \\ k & if \quad u = k \end{cases}$$

Siendo *d* la diferencia de valor de aptitud de los dos picos, que en nuestro caso se establece con valor *1*. La función trampa es cada vez más difícil, según el número de bits k utilizados para evaluar subcadenas, y también a medida que se reduce el valor d. En nuestros experimentos hemos utilizado valores de k=3 y k=4. La longitud de los individuos la hemos establecido en *l=25*k* bits. Como herramienta de pruebas se ha utilizado GAlib, con una configuración que permite utilizar el modelo de islas que corre sobre un único procesador.

4. Experimentos con onemax

En nuestros experimentos utilizamos selección por torneo con reemplazo, cruce de un punto con probabilidad 100%, y no utilizamos por tanto mutación. Todos los resultados que presentamos en esta sección son un promedio de 200 ejecuciones. En primer lugar trabajamos con la función *one-max* con cadenas de longitud 25, 100 y 400. En los experimentos hemos tratado de analizar la mejor forma de invertir esfuerzo de computación, equilibrando tres parámetros en estudio: el número de generaciones, número de individuos en la población y número de ejecuciones. Considerando que disponemos de un esfuerzo de computación fijo (1024 evaluaciones en total), hemos realizado varias configuraciones para invertir este esfuerzo, teniendo en cuenta el número de ejecuciones que vamos a realizar el experimento (por ejemplo 2 veces), número de individuos a utilizar (por ejemplo 8) y número de generaciones que podemos evaluar para que en total no superemos el esfuerzo de computación establecido (64 para este ejemplo).

#Ejecuciones	#Individuos	#Generaciones
1	8	128
1	16	64
1	32	32
2	8	64
2	16	32
4	8	32
4	16	16
8	8	16
8	16	4

Tabla 1. Listado de experimentos realizados empleando en cada caso: #ejecuciones x #individuos x #generaciones = 1024 evaluaciones disponibles.

En la Tabla 1 mostramos la serie de experimentos que realizamos. Cada una de las configuraciones se repite 200 veces para obtener promedios. Las figuras 1, 2 y 3 muestran los resultados, promediados como ya se ha dicho sobre 200 repeticiones independientes, de cada experimento resumido en la tabla 1, y para el problema *one-max* con 25, 100 y 400 bits respectivamente.

Lo primero que debemos notar es que la utilización de un número pequeño de individuos en cada una de las ejecuciones es una mala elección. Incluso cuando gastamos esfuerzo en un número alto de generaciones, los resultados son normalmente peores que los obtenidos con otras configuraciones (comparado por ejemplo con los resultados obtenidos cuando se realizan 2 ejecuciones con una población de 8 individuos). Por otro lado, los mejores resultados se han obtenido cuando se utiliza 1 sola ejecución, y el tamaño de población más grande posible, por supuesto, y justamente por eso, con el menor número de generaciones evaluadas, dado los recursos fijos de computación establecidos.

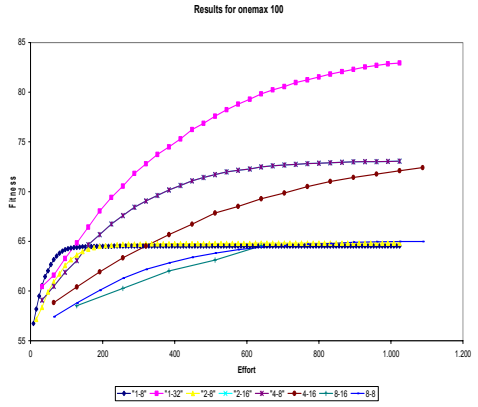

Figura 1. One-max problem con 100 bits.

Sin embargo, es interesante que nos fijemos en que el segundo mejor resultado de cada una de las gráficas se ha obtenido con experimentos tales que el producto del número de ejecuciones y número de individuos coincide con el número de individuos totales que obtiene los mejores resultados (por ejemplo, para 100 bits, el mejor resultado se obtiene con 1 ejecución de 32 individuos, y el segundo mejor resultado con 2 ejecuciones y 16 individuos). Además, debemos indicar, que en el caso de que utilizáramos más de un procesador, obviamente este segundo mejor resultado se convertiría automáticamente en el mejor, ya que las ejecuciones se realizarían en paralelo, empleando por eso la mitad de tiempo.

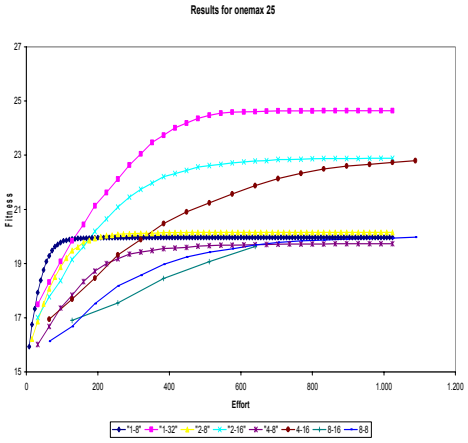

Figura 2. One-max problem con 25 bits.

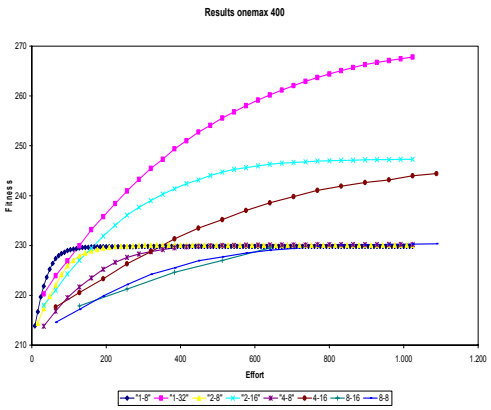

Figura 3. One-max problem con 400 bits.

Según lo anterior, es preferible ahorrar esfuerzo de computación en generaciones, y emplearlo utilizando poblaciones más grandes, o más ejecuciones del experimento. Este resultado que se ha obtenido considerando los tres parámetros simultáneamente, coincide con los descritos

anteriormente en la literatura [7] que solamente tuvo en cuenta el número de ejecuciones y tamaño de las poblaciones, y similarmente con los resultados para Programación Genética [17] que consideraba solamente el número de ejecuciones y generaciones evaluadas.

5. Experimentos con la función trampa

Los siguientes experimentos realizados han utilizado la función trampa utilizando los valores *m=25* y *k=3 y 4*. De nuevo, el esfuerzo máximo por cada experimento se estableció en 1024, y las pruebas que hemos desarrollado se resumen en la Tabla 2.

#Ejecuciones	#Individuos	#Generaciones
1	128	64
1	64	128
1	256	32
2	64	64
2	128	32
4	64	32
4	128	16
8	64	16
8	128	8

Tabla 2. Resumen de experimentos utilizando #ejecuciones x #individuos x #generaciones = 1024 evaluaciones.

De nuevo, los resultados obtenidos para cada experimento, y mostrados en las gráficas, han sido promediados sobre 200 repeticiones independientes. Hemos seguido los estudios previos sobre la función trampa [6, 15] y no utilizamos operador de mutación, además de utilizar selección por torneo de dos individuos. Utilizamos además cruce uniforme con probabilidad 1.

La figura 4 muestra los resultados experimentales obtenidos con la función trampa utilizando 25 copias con *k=4*. La figura 5, utiliza también 25 copias con *k=3*. Fijémonos en ambas figuras, que aunque los experimentos que utilizan una sola ejecución y 128 individuos durante 64 generaciones, o bien 2 ejecuciones y 64 individuos de nuevo durante 64 generaciones son los que obtienen los mejores resultados durante las primeras etapas del proceso evolutivo, éstos resultados son superados por los experimentos que utilizan 1 ejecución de 256 individuos durante 32 generaciones, y también 2 ejecuciones de 128 individuos y de nuevo 32 generaciones (tanto en con *k=3* y *k=4*).

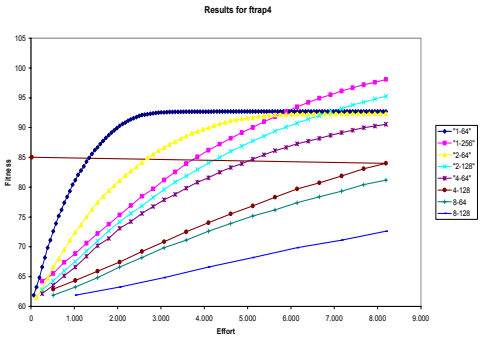

Figura 4. Experimento con función trampa f (4).

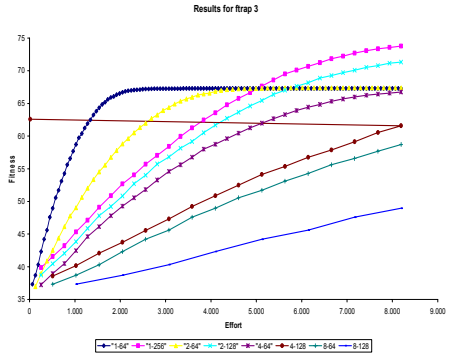

Figura 5. Experimento con función trampa f (3).

Estos resultados coinciden con los obtenidos en el problema *one-max*. Hemos visto que dado un número de ejecuciones elegido, es mejor utilizar más individuos y menos generaciones que lo contrario (salvo el caso de 8 ejecuciones). Debemos en general concluir, que los experimentos que utilizan dos ejecuciones con la mitad de individuos son preferibles cuando dispongamos de sistemas paralelos o distribuidos, porque las diferencias en tiempo de computación serán tan grandes, que permitirán obtener la misma calidad de fitness que el modelo de una

sola ejecución con todos los individuos en aproximadamente la mitad de tiempo.
Si analizamos más detenidamente algunas de estas figuras, podemos llegar de nuevo a conclusiones sobre esfuerzos de computación en AGs Paralelos. Basta que consideremos cada una de las ejecuciones de un determinado experimento como una población independiente. Si nos fijamos en la figura 4 de nuevo, vemos que aunque una población con 256 individuos obtiene mejores resultados que 4 poblaciones con 64 individuos, en el caso de utilizar una máquina secuencial, las cosas cambian cuando disponemos de 4 procesadores. La tabla 3 muestra un resumen de estas conclusiones. Por ejemplo, si dispusiéramos 8192 unidades de esfuerzo de computación la configuración de 1 población con 256 individuos sería la ganadora. Pero si solo disponemos de 2048 unidades de esfuerzo (considerado este esfuerzo por procesador), sería mejor utilizar todos los procesadores y llegar a individuos con mayor calidad.

Procesadores	1	4	1	1
Poblaciones	4	4	1	1
Individuos por población	64	64	256	256
Esfuerzo disponible	8192	2048	8192	2048
Fitness obtenido	90,54	90,54	98,08	75,35

Tabla 3. Si disponemos de 4 procesadores, los resultados son diferentes.

6. Conclusiones y trabajo futuro

Hemos presentado en este trabajo un análisis del esfuerzo de computación en los Algoritmos Genéticos, variando tres parámetros que influyen decisivamente en la convergencia y eficiencia. Experimentalmente hemos estudiado la relación entre el número de ejecuciones, individuos en la población, y número de generaciones, y hemos llegado a las siguientes conclusiones:

- Utilizar una población de individuos pequeña en cada ejecución no es una buena elección. Incluso si podemos evaluar un gran número de generaciones, los resultados suelen ser habitualmente peores.
- Los mejores resultados se obtienen utilizando una sola ejecución con el mayor número posible de individuos.
- Tal como se había concluido previamente analizando dos parámetros [7], podemos decir que es mejor ahorrar generaciones y utilizar el esfuerzo haciendo las poblaciones mayores.
- Aunque 1 población con 256 individuos obtiene mejores resultados en la función trampa que 4 poblaciones con 64 individuos cada una cuando se utiliza una máquina secuencial, las cosas cambian si disponemos de 4 procesadores y una implementación paralela.

7. Agradecimientos

Este trabajo ha sido financiado por los Proyectos del Ministerio Español de Ciencia y Tecnologías TIC 750/2002 y TIC2002-04498-C05-01.

8. Bibliografía.

[1] A. Hernandez-Aguirre, B.P. Buckles, A. Martinez-Alcantara, The probably approximately correct (PAC) population size of a genetic algorithm 12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'00) 2000 Vancouver, British Columbia, Canada.

[2] Arabas, Z. Michalewicz, and J. Mulawka. GAVaPS: a genetic algorithm with varying population size. In Proceedings of the First IEEE Conference on Evolutionary Computation, pages 73-78. IEEE Press, Piscataway, NJ, 1994.

[3] Back, A.E. Eiben, and N.A.L. van der Vaart. An empirical study on Gas "without parameters". Proceedings of the 6th Conference on Parallel Problem Solving from Nature, number 1917 in Lecture Notes in Computer Science, pages 315-324. Springer, Berlin, 2000.

[4] Jiang J, Cai, J., Nording G.P., and Li L., Parallel microgenetic algorithm design for photonic crystal and waveguide structures. Optics Letters 23, Vol. 28, 2381-2383. (2003).

[5] Cantu-Paz, E., Goldberg, D.E.: Modeling idealized bounding cases of parallel genetic algorithms. In Koza, J., et al., eds.: Proceedings of the Second Annual Genetic Programming Conference, Morgan Kaufmann (1997) 353–361

[6] F. Fernández, "Distributed Genetic Programming Models with application to Logic Synthesis on FPGAs", PhD Thesis. University of Extremadura. February 2001.

[7] Cantu-Paz, E., Goldberg, D. E.: Are multiple runs of genetic algorithms better than one? Proceedings Gecco 2003. Chicago. Pp. 801-812. Springer Verlag.

[8] Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. In Whitley, L.D., ed.: Foundations of Genetic Algorithms 2, Morgan Kaufmann (1993) 93–108.

[9] F. Fernández, M. Tomassini, J.M. Sánchez, "Experimental Study of Isolated Multipopulation Genetic Programming", in Proceedings IECON 2000. 2000 IEEE International Conference on Industrial Electronics Control and Instrumentation. pp. 2672- 2677. IEEE Press 2000.

[10] F. Fernandez, M. Tomassini, W. Punch and J.M. Sanchez, "Experimental study of multipopulation parallel genetic programming", Genetic Programming, proceedings of EuroGP2000, Springer-Verlag, Lecture Notes in Computer Science, Vol. 1802, 283- 293, 2000.

[11] F. Fernandez, E. Cantú-Paz, T. Manzano, I. López, "Saving Resources with Plagues in Genetic Algorithms", VIII Parallel Problem Solving from Nature Conference. LNCS 3242. pp. 272-281. Springer. 2004.

[12] Hidalgo J.I., Evolutionary Algorithms for solving the partitioning and placement problems in Multi-FPGA systems. Proceedings of 2000 Genetic and Evolutionary Computation Conference. Workshop Program (2000).

[13] Goldberg, D.E., Deb, K., Clark, J.H.: Genetic algorithms, noise, and the sizing of populations. Complex Systems 6 (1992) 333–362

[14] Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, MA (1989).

[15] Harik, G., Cant´u-Paz, E., Goldberg, D., Miller, B.L.: The gambler's ruin problem, genetic algorithms, and the sizing of populations. Evolutionary Computation 7 (1999) 231–253.

[16] J. Costa, R. Tavares, and A. Rosa. An experimental study on dynamic random variation of population size. In Proc. IEEE Systems, Man and Cybernetics Conf. volume 6, pages 607-612, Tokyo, 1999. IEEE Press.

[17] Luke, S.: When short runs beat long runs. In Spector, L. et al., eds.: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann (2001) 74–80

[18] M. Tomassini, L. Vanneschi, F. Fernández, G. Galeano, "Diversity in Multipopulation Genetic Programming", Genetic and Evolutionary Computation Conference, GECCO 2003. Chicago. USA. pp. 1812-1813. July 2003.

[19] Gao Y. Lower Bounds on the Population Size in Genetic Algorithms and Implicit Parallelism Revisited. Tech. Report TR02-20, Dept. of Computing Science, University of Alberta, 2002.

[20] Miller, B.L., Goldberg, D.E.: Genetic algorithms, selection schemes, and the varying effects of noise. Evolutionary Computation 4 (1996) 113–131

[21] Gao Y. Population size and Sampling complexity in Genetic Algorithms. Proceedings of the Bird of a Feather Workshops (Gecco 2003), Learning, Adaptation, and Approximation in Evolutionary Copmputation, pp. 178-181, AAAI, 11 July 2003.

[22] Eiben A.E., E. Marchiori V. A., Valk, "Evolutionary Algorithms on-the-fly Population Size Adjustment", VIII Parallel Problem Solving fron Nature Conference. LNCS 3242. Springer (2004).

[23] F. Fernández, M. Tomassini, L. Vanneschi, "An Empirical Study of Multipopulation Genetic Programming", . Genetic Programming and Evolvable Machines, Vol. 4. 2003. pp. 21-51. Kluwer Academic Publishers.