Prevención del bloat mediante una interpretación espacio-temporal de la Programación Genética Paralela.

Daniel Lanza[1], Francisco Fernández[2], Francisco Chávez[3], Gustavo Olague[4]

1. CERN. Organización Europea para la Investigación Nuclear.
daniel.lanza@cern.ch
2. Departamento de Arquitectura de Computadores, Universidad de Extremadura, C/. Santa Teresa de Jornet, 38, CP: 06800, Mérida, Spain. fcofdez@unex.es
3. Departamento de Ingeniería en Sistemas Informáticos y Telemáticos, Universidad de Extremadura,
C/. Santa Teresa de Jornet, 38, CP: 06800, Mérida, Spain. fcofdez@unex.es
4. CICESE
Carretera Ensenada-Tijuana 3918, Zona Playitas, 22860 Ensenada, B.C., México.
gustavo.olague@gmail.com

Abstract. La programación genética (PG) ha tenido éxito en la resolución de problemas difíciles del dominio del aprendizaje máquina. Pero a pesar de su éxito, todavía hay dificultades en la propia técnica cuya solución definitiva no se ha alcanzado para la PG, particularmente el del incremento continuo en tamaño y complejidad de las soluciones cuando el proceso evolutivo está actuando.
En este trabajo se propone una nueva alternativa para evitar este bien conocido fenómeno del *bloat*, basado en un principio que surge de forma natural en entornos de computación paralelo, cuando se evalúan de forma simultánea individuos de diferente tamaño y complejidad. Así, utilizando el tiempo de ejecución como medida del tamaño del individuo, se desarrolla un nuevo mecanismo natural que permite reducir el tamaño de los individuos sin perder calidad. Los experimentos desarrollados con un buen número de problemas de test muestran la viabilidad de la propuesta.

PALABRAS CLAVE: programación genética, control del bloat, tiempo de ejecución

1 Introducción

Uno de los problemas mejor conocidos y asociados al uso de cromosomas de tamaño variable en algoritmos evolutivos, y en particular en la Programación Genética (PG), es el incremento continuo de su tamaño a medida que se van generando nuevos individuos mediante procesos de selección basados en fitness y aplicación de operadores genéticos de cruce y mutación. Este fenómeno conocido como *bloat* provoca problemas diversos: a medida que se calculan nuevas

generaciones, cada vez se requiere más cantidad de memoria para albergar el mismo número de individuos, y a la vez se necesita más tiempo para evaluarlos, dado el incremento de tamaño; además, las soluciones generadas son difíciles de interpretar.

Aunque son varias las razones que provocan este problema, y varias las técnicas para hacerle frente, como se muestra más adelante, en este artículo proponemos un nuevo método que intenta neutralizar este problema enfocándolo desde una nueva perspectiva, y que surge de manera natural al utilizar modelos de computación paralelos y distribuidos.

La propuesta que se aporta intenta hacer una interpretación espacio-temporal del proceso evolutivo: mediante el uso de un mecanismo evolutivo de tipo estado-estacionario (steady-state) que se aplica mediante procesos paralelos de evaluación, selección y cruce. Así, una adecuada interpretación de la naturaleza paralela de los algoritmos evolutivos y su relación con el tamaño del individuo y tiempo de evaluación, nos permitirá desarrollar un mecanismo autónomo de control del bloat.

Los resultados preliminares obtenidos mediante simulaciones sobre un bien conocido conjunto de problemas de test con esta nueva metodología muestra un control del bloat exitoso.

El resto del artículo se organiza del siguiente modo: En la sección **??** se describe el problema del *bloat* y las alternativas disponibles; En la sección **??** mostramos los problemas utilizados y los experimentos diseñados; La sección **??** muestra los resultados obtenidos en los experimentos, y finalmente en la sección **??** se describen las conclusiones extraídas de los resultados obtenidos.

2 ¿Por qué ocurre el bloat?

El fenómeno del bloat ha sido ampliamente estudiado en la literatura de PG, tanto desde el punto de vista teórico como experimental.

Las primeras teorías sobre el bloat se concentraron en la existencia de intrones [**?**], regiones de código que no llevan a cabo ninguna operación. Un conocido tipo de intrón, denominado código inviable, invalida todo el subárbol/código contenido en el. Un ejemplo de este intrón es la multiplicación por 0 en operaciones aritmética, que implicaría que nada contenido en él subárbol al que afecta hará cambiar el valor resultado de la multiplicación, 0, aunque si puede cambiar el tamaño y complejidad del árbol.

Una teoría no basada en intrones [**?**], postula que el fitness es el causante del bloat. Sugiere que los árboles crecen ya que es más probable que un árbol grande proporcione una solución con buen fitness que un árbol pequeño. Por este motivo, al empezar el proceso con árboles pequeños, estos crecen conforme se lleva a cabo el proceso evolutivo.

Hay varias razones que hacen deseable aplicar algún tipo de control del bloat. En primer lugar, se espera que la solución final sea simple para que pueda ser fácilmente entendida. Además conviene realizar el mayor número de evaluaciones (explorar más soluciones) en un espacio de tiempo dado y aunque para algunos

problemas, el coste de la evaluación no está directamente relacionado con el tamaño del árbol, sin embargo, el consumo de memoria si será siempre un problema, al tratarse de un recurso limitado.

Dada las razones anteriores, la programación genética puede ser vista como una carrera contra el tiempo, donde el objetivo es conseguir los mejores resultados antes de que el bloat haga que sea difícil avanzar.

Entre los métodos recientes para controlar el bloat podemos citar métodos que directamente atacan al tamaño de los árboles, tal como limitar el tamaño máximo permitido [**?**,**?**], [**?**]; asignar calidad proporcional al tamaño, [**?**], [**?**,**?**,**?**]; técnicas multiobjetivo [**?**,**?**,**?**]; priorización por tamaño, [**?**], [**?**]; o métodos que indirectamente consiguen reducir el número total de nodos a evaluar manteniendo la calidad de las soluciones, tales como *prune-plant*, [**?**], o la *plaga* [**?**]. Algunas propuestas muy recientes se enfocan en versiones particulares de GP *neatgp*. En nuestro caso estamos más interesados en soluciones genéricas para GP estándar.

En este artículo estamos interesados en analizar nuevamente el modo que la PG trabaja, intentando descubrir mecanismos más naturales de control del tamaño. Como mostraremos a continuación nos hemos fijado particularmente en el paralelismo intrínseco de los algoritmos basados en poblaciones: en las poblaciones naturales todos los individuos *trabajan* de forma simultánea. En nuestros modelos basados en computador, esto es posible mediante el uso de arquitecturas paralelas y distribuidas -que permiten evaluar varios individuos a la vez- y el uso de modelos de estado estacionario, que evita esperas asociadas a puntos de sincronización, tal como la aplicación de operadores genéticos al final de cada generación. En los modelos de estado estacionario, los individuos pueden sufrir operaciones genéticas para crear descendientes en cualquier momento. Como veremos a continuación este modo de funcionamiento es el que inspira la nueva propuesta que presentamos. No obstante, la idea que desarrollamos puede igualmente considerarse en procesos generacionales, y es precisamente en este contexto en el que dearrollamos las pruebas preliminares mediante simulaciones que muestran los beneficios de la idea que se plantea.

3 Metodología

En el modelo que desarrollamos a continuación, tiene especial relevancia la relación entre el tamaño de los individuos y su tiempo de ejecución: frecuentemente los individuos grandes requieren mayor tiempo de ejecución, mientras que individuos pequeños necesitan poco tiempo para ser evaluados. Aunque esta regla no siempre se cumple, debido a uso de condiciones que hacen que ciertas partes del código no se ejecuten, o los bucles que aunque cortos en número de instrucciones pueden requerir largos tiempos de ejecución, por ejemplo, hay un alto grado de correlación entre tiempo de ejecución y tamaño de los individuos.

La idea que desarrollamos a continuación se basa en hacer una medición del tamaño de los individuos a posteriori: utilizando el tiempo de evaluación de los mismos. Lo habitual en las técnicas de control de bloat es utilizar una medida del tamaño -tal como el número de nodos del árbol- a priori, de modo que esa

medida es utilizada de algún modo en el proceso de evolución; por ejemplo, incluyendo esa medición en la función de fitness para penalizar individuos grandes. Lo que nosotros proponemos difiere de las técnicas tradicionales en dos puntos fundamentales: (i) la medición del tamaño es indirecta, en lugar de medir de algún modo espacio de memoria consumido utilizamos el tiempo de ejecución como medida del tamaño; (ii) la medición se realiza a posteriori, es decir, el dato se conoce una vez ha sido evaluado el individuo, por lo que no es posible utilizar la medida en la propia función de evaluación.

La ventaja de la medición a posteriori es que podemos realizarla a la vez que se calcula el valor de aptitud. Es más, el mismo proceso de cálculo de aptitud realiza simultáneamente el cálculo de tamaño. Por otro lado, cuando disponemos de una arquitectura paralela, podemos realizar todas las evaluaciones, y por tanto las medidas de tamaño, de forma simultánea. Más aún, dado que diferentes individuos van a culminar su evaluación en diferente momento, pues cada uno tiene un tamaño determinado, si aplicamos el modelo estacionario, permitiendo que los individuos puedan sufrir las operaciones genéticas a medida que están disponibles, de forma natural, los individuos se irán cruzando con aquellos que tienen tamaño similar: dado que cuando se lanza la generación, todos los individuos comienzan su evaluación a la vez, el orden de disponibilidad una vez evaluados dependerá de su tiempo de ejecución, y por tanto de su tamaño, de modo que irán estando disponibles para generar nuevos individuos en un orden que depende de su tamaño. Así, los individuos más pequeños llegarán primero, y entre ellos podrán cruzarse, mientras que los más grandes llegarán al final, y sólo entre ellos podrán realizar operaciones genéticas. Aunque en los modelos estacionarios la dinámica es más compleja, esta es la idea básica que se utiliza a continuación para analizar un nuevo mecanismo de control de bloat, que puede desarrollarse como mecanismo independientemente de la arquitectura de cómputo utilizada. Así, el modelo que mostramos incluye una serie de etapas necesarias en arquitecturas monoprocesador para realizar la simulación, etapas algunas de ellas que en entorno de cómputo paralelo se producen de modo automático, dado el natural modo en que los individuos terminan su ejecución en el orden asociado a su tamaño. No obstante, hemos decidido desarrollar el modelo genérico que puede ser utilizado en cualquier entorno, sea esta paralelo o no.

3.1 Individuos de parecido tamaño producen descendencia similar

Hemos comenzado nuestro estudio fijándos en el doble problema que causa el bloat: por un lado el consumo progresivo de memoria al ser los individuos cada vez de mayor tamaño, y por otro lado el aumento del tiempo de ejecución requerido, al incluir los individuos cada vez mayor número de operaciones. Ambos componentes, espacio de memoria y tiempo de ejecución están directamente relacionados, aunque no de modo lineal: algunos nodos de los individuos en programación genética nunca llegan a utilizarse.

Para comenzar nuestro estudio, vamos en primer lugar a fijarnos en la selección de individuos para realizar la reproducción: permitiremos sólo que los

individuos se crucen con otros de similar tamaño; así, cuando se crucen dos individuos pequeños, sus hijos deben ser de menor tamaño que cuando se crucen individuos grandes. Si encontráramos alguna relación particular entre la calidad de los descendientes asociada al tamaño de los padres, podríamos diseñar un método efectivo para el control del tamaño.

Con este objetivo se introduce una etapa del proceso evolutivo que se lleva a cabo antes de la fase de selección de individuos. En esta nueva etapa, todos los individuos de la población son ordenados en función de su tamaño con el fin de realizar agrupaciones de individuos de tamaño similar. El número de grupos es el mismo a lo largo de todo el proceso evolutivo y se decide antes de iniciar la ejecución.

Una vez que los grupos de individuos con tamaño similar han sido realizados, se llevan a cabo las fases de selección y cruce. En este caso, estas fases tienen una peculiaridad, ya que a diferencia del proceso evolutivo normal que toma toda la población en cuenta, se llevan a cabo procesos de selección y cruce por cada uno de los grupos. De este modo, la reproducción se lleva a cabo con individuos de tamaños similares, ya que el proceso de reproducción obtiene individuos del anterior proceso de selección el cual obtuvo individuos de un mismo grupo.

Se espera que la fase de reproducción genere en cada grupo, individuos con un tamaño medio similar al del grupo al que pertenece, aunque ciertamente el tamaño variará de unos individuos a otros, y en posteriores etapas podrán ser asignados a otros grupos de reproducción diferente.

Midiendo el tamaño a través del tiempo En numeroso métodos para el control del bloat que nos muestra la literatura, se ha medido el tamaño en función del número de nodos o profundidad que construyen el árbol de la solución del individuo. Esta técnica aporta un simple indicador a través del cual podemos extraer una idea del coste computacional de la solución. Aunque utilizada ampliamente, esta técnica no provee una aproximación muy acertada cuando las operaciones que se realizan son más complejas o costosas, o cuando los árboles PG incluyen nodos que no son utilizados. Aunque esta forma de medir es útil para calcular consumo de memoria, es bastente inexacta en el caso de medidas de tiempo de ejecución de los algoritmos. En su lugar, sería más conveniente realizar la medida del coste de los individuos mediante su tiempo de ejecución.

En nuestra propuesta, y aunque en un primer momento se puede realizar la medición del modo clásico, contando simplemente el número de nodos de cada árbol, pretendemos afinar el proceso midiendo el tiempo real de ejecución de cada individuo.

Haciendo uso de la nueva etapa de agrupación de individuos y de la técnica de tomar el tiempo de ejecución como el tamaño de los individuos, se ha realizado una serie de experimentos cuyos resultados se muestran a continuación.

Problemas de prueba A continuación se describen los resultados obtenidos haciendo uso de la metodología descrita anteriormente. Para obtener estos re-

sultados se han hecho experimentos utilizando algunos de los problemas más comunes en la programación genética [**?**]. Los problemas utilizados han sido:

- Ants, hormigas recolectando comida de la forma más eficiente.
- Lawnmower, recorrer una zona completa de la forma más eficiente (simulando cortar el césped).
- Multiplexer, contrucción de un multiplexor de 6 bits.
- Parity, encontrar la fórmula para el cálculo de la paridad.
- Regression, intenta resolver un problema de regresión simbólica.

Se ha utilizado la herramienta ECJ [**?**], con las configuraciones para cada problema que trae por defecto. El único cambio que se ha hecho en el algoritmo es la agrupación de individuos por tamaño: para cada uno de estos problemas se han realizado experimentos con diferente número de grupos. Se han realizado con 1, 2, 4, 8, 16, 32, 64, 128, 256 y 512 grupos, teniendo en cuenta que el tamaño total de la población es de 1024 individuos, hemos trabajado así desde un único grupo de 1024 individuoas, hasta 512 grupos con 2 individuos por grupo.

Cada experimento se ha ejecutado 30 veces, y se muestran a continuación los resultados promedios obtenidos en cada caso.

4 Resultados

La figura **??** nos muestra el resumen de resultados obtenidos para cada uno de los problemas con los que se ha trabajado. Incluye dos tipos de gráficas para cada problema, la referida a la calidad de la mejor solución encontrada en cada generación para cada experimento, y las referidas al tamaño medio de los individuos de la población. Ese valor medio multiplicado por el número total de individuos nos permitiría calcular el consumo total de memoria de la población a lo largo del tiempo.

Por otro lado, cada una de las gráficas incluye una línea de color asociada a cada uno de los experimentos realizados con distintas agrupaciones de individuos clasificados por tamaño: 1 sólo grupo equivale al experimento clásico, 2, 4, etc, corresponde al número de grupos establecidos para que los individuos se crucen entre sí y sólo con individuos de su grupo en cada generación.

4.1 Fitness

Analizamos en primer lugar la calidad de las soluciones que se encuentran dependiendo del número de grupos que se establece en la población.

La parte izquierda de la figura **??**, y de arriba abajo, muestra la evolución de la calidad de la mejor solución encontrada para los problemas ant, lawnmover, multiplexer, parity y regression. Salvo en el caso de Parity, en el resto de problemas podemos ver que la solución alcanzada cuando se utilizan pocos grupos es similar a la la calidad de solución en el experimento estándar (1 sólo grupo). Aunque en el problema *parity* el experimento clásico dio los mejores resultados,

en el problema *ant*, muchos de los experimentos con agrupación mejoraron notablemente el caso clásico. En resumen podríamos decir que la agrupación no ha perjudicado la calidad de la soluciones encontradas, al menos cuando se utiliza un número pequeño de grupos. Obviamente, cuando se utilizan muchos grupos, llegando al caso límite en el que sólo hay dos individuos por grupo, los resultados no son buenos.

4.2 Número de nodos

Por otro lado, si analizamos la parte derecha de la figura, en la que se incluyen las gráficas que muestran la evolución del tamaño medio de los individuos a lo largo de las generaciones, podemos observar en todos los problemas cómo el crecimiento del tamaño medio está inversamente relacionado con el número de grupos que utilicemos en el experimento. Así, el tamaño medio de los individuos es mucho menor cuando se utilizan muchos grupos; por contra, los individuos crecen más cuando se utiliza el modelo clásico en el que todos los individuos tienen la opción de cruzarse con cualquier otro de la población.

Así, podríamos resumir indicando que una correcta utilización de agrupación de individuos por tamaño en el proceso evolutivo conlleva una reducción significativa del tamaño de los individuos asegurando simultáneamente la calidad de las soluciones.

4.3 Interpretación espacio-temporal del proceso evolutivo

Aunque los resultados son preliminares, y la experimentación se ha llevado a cabo mediante procesos ejecutados en sistemas monoprocesador, la idea que ha inspirado el nuevo mecanismo de control nace directamente de los sistemas de cómputo paralelos, y será precisamente en estos dónde ésta encuentre todo su potencial: no será necesario realizar mediciones de tamaño, ordenación ni agrupación de individuos, bastará permitir que cada individuo se *relacione* con aquellos que han completado su evaluación en paralelo en una ventana temporal similar. Así, individuos que se evalúan en *tiempo* similar quedan automáticamente agrupados por *tamaño*.

Esperamos en próximos trabajos aportar resultados de este nuevo mecanismo paralelo natural que permite aprovechar mejor las relación espacio-temporal de los individuos que evolucionan mediante Programación Genética.

5 Conclusiones

En este artículo hemos presentado un nuevo mecanismo de control del tamaño de los individuos en programación genética basado en un proceso que surge de forma natural en arquitecturas paralelas: la evaluación simultánea de los individuos de la población produce una ordenación temporal de individuos asociada al tamaño de los mismos; los individuos más pequeños completan antes su evaluación, mientras que los más grandes en tamaño tardan más en evaluarse. Así,

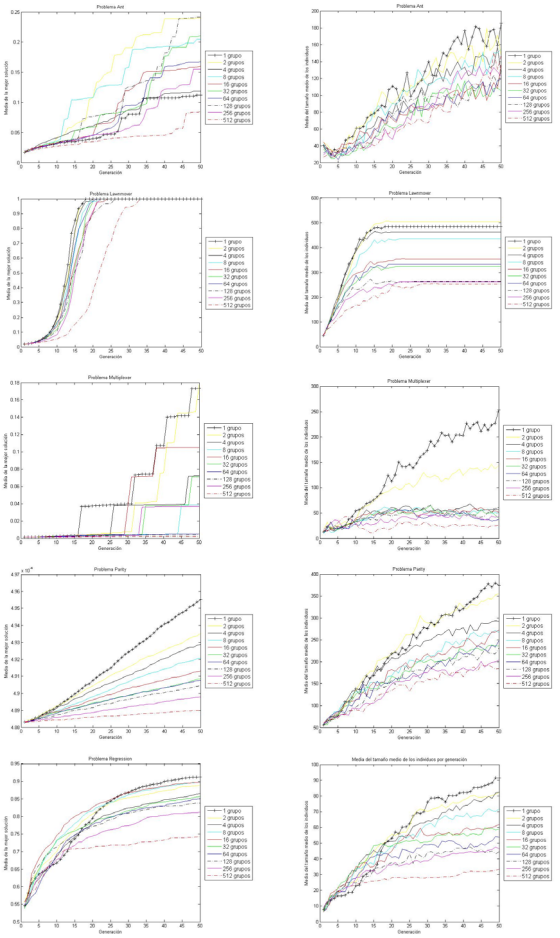

Fig. 1. De izquierda a derecha y de arriba a abajo se presenta el mejor fitness y el número medio de nodos por generación para el problema de Ant, Lawnmover, Multiplexer, Parity y Regression

aprovechar este mecanismo natural permite que los individuos se relacionen en los procesos de selección con aquellos con tamaño similar.

Mediante una serie de experimentos realizados mediante simulación del proceso de agrupación, y considerando diferentes niveles de agrupación, hemos observado cómo el número de grupos total con que se trabaja, y por ello el número de individuos en cada grupo, afecta directamente al tamaño de los individuos que se generan. Una correcta selección de número de grupos permite simultáneamente reducir el tamaño medio de los individuos de la población asegurando a la vez la calidad de las soluciones obtenidas, lo que en definitiva permite ahorrar tiempo y espacio (memoria) en la búsqueda de soluciones.

References

1. Trujillo, L., Muñoz, L., Galván-López, E., Silva, S. (2016). neat Genetic Programming: Controlling bloat naturally. Information Sciences, 333, 21-43.
2. Eva Alfaro-Cid, Juan Julián Merelo Guervós, F. Fernández de Vega, Anna Isabel Esparcia-Alcázar, Ken Sharman. Bloat Control Operators and Diversity in Genetic Programming: A Comparative Study. Evolutionary Computation 18(2): 305-332 (2010)
3. F. Fernandez, L Vanneschi, M Tomassini. The effect of plagues in genetic programming: A study of variable-size populations. Genetic Programming, 395-423
4. Tackett, W. A. (1994). Recombination, Selection, and the Genetic Construction of Computer Programs. PhD thesis, University of Southern California, Department of Electrical Engineering Systems.
5. Blickle, T. (1996). Theory of Evolutionary Algorithms and Application to System Synthesis. PhD thesis, Swiss Federal Institute of Technology, Zurich.
6. Poli, R. (2003). A simple but theoretically-motivated method to control bloat in genetic programming. In Genetic Programming, Proceedings of EuroGP'2003, pages 204–217. Springer.
7. Luke, S. (2000). Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat. PhD thesis, Department of Computer Science, University of Maryland, A. V. Williams Building, University of Maryland, College Park, MD 20742 USA.
8. Luke, S., Panait, L.: Lexicographic parsimony pressure. In: GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kauffman (2002) 829–836
9. Luke, S. (2003). Modification point depth and genome growth in genetic programming. Evolutionary Computation, 11(1):67–106.
10. Luke S. and Liviu Panait. 2004. Alternative Bloat Control Methods. Genetic and Evolutionary Computation – GECCO 2004 Volume 3103 of the series Lecture Notes in Computer Science pp 630-641.
11. Luke S. and Liviu Panait. 2006. A comparison of bloat control methods for genetic programming. Evol. Comput. 14, 3 (September 2006), 309-344.
12. Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA.
13. Cavaretta, M. J. and Chellapilla, K. (1999). Data mining using genetic programming: The implications of parsimony on generalization error. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, Proceedings of

the Congress on Evolutionary Computation, volume 2, pages 1330–1337, Mayflower Hotel, Washington D.C., USA. IEEE Press.
14. Belpaeme, T. (1999). Evolution of visual feature detectors. In Poli, R., Cagnoni, S., Voigt, H.-M., Fogarty, T., and Nordin, P., editors, Late Breaking Papers at EvoISAP'99: the First European Work- shop on Evolutionary Computation in Image Analysis and Signal Processing, pages 1–10, Goteborg, Sweden.
15. Nordin, P. and Banzhaf, W. (1995). Complexity compression and evolution. In Eshelman, L., editor, Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95), pages 310– 317, Pittsburgh, PA, USA. Morgan Kaufmann.
16. Bleuler, S., Brack, M., Thiele, L., and Zitzler, E. (2001). Multiobjective genetic programming: reducing bloat using SPEA2. In Proceedings of the 2001 Congress on Evolutionary Computation CEC2001, pages 536–543, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea. IEEE Press.
17. de Jong, E. D. and Pollack, J. B. (2003). Multi-objective methods for tree size control. Genetic Programming and Evolvable Machines, 4(3):211–233.
18. Ekart, A. and Nemeth, S. Z. (2001). Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. Genetic Programming and Evolvable Machines, 2(1):61–73.
19. Langdon, W. B. and Poli, R. (1997b). Fitness causes bloat. In Chawdhry, P. K., Roy, R., and Pant, R. K., editors, Soft Computing in Engineering Design and Manufacturing, pages 13–22. Springer- Verlag London.
20. ECJ, A Java-based Evolutionary Computation Research System. George Mason University's ECLab Evolutionary Computation Laboratory.