Algoritmos evolutivos en Perl

Juan J. Merelo Guervós

Historial de revisiones
Revisión 1.1 26 Septiembre 2002
Versión revisada para el congreso
Revisión 1.0 30 Julio 2002
Submitted.
Revisión 0 26 de julio de 2002

En este artículo se describe el módulo Algorithm::Evolutionary, un módulo en Perl para algoritmos evolutivos, diseñado con el objeto de que fuera fácilmente integrable con XML. Esto ha forzado a una serie de decisiones de diseño, que se detallan en el artículo.

Tabla de contenidos

1. Introducción

En una aplicación Open Source, aparte del hecho de que exista en sí, se debe, desde nuestro punto de vista, probar tres cosas: su **necesidad**, su **calidad** y su **aplicabilidad**.

En el mundo Open Source, no se trata simplemente de escribir una aplicación porque sí, sino de probar que realmente es un producto para el cual existe un hueco; si no, se cae en el error de crear sin ton ni son, sin que se produzca realmente un beneficio a la comunidad. En la sección Sección 1 presentaremos el panorama de las bibliotecas de algoritmos evolutivos, mostrando que, hasta el momento, no se había hecho una biblioteca versátil para algoritmos evolutivos en Perl

Por otro lado, habrá que probar que realmente lo diseñado corresponde a la necesidad existente, es decir, que cubre el nicho que se le ha sido asignado, y que lo hace con un mínimo de dignidad. También a qué decisiones de diseño han

llevado estas especificaciones, y qué ventajas ha tenido el usar las herramientas que se han usado. Esto se describirá en la sección Sección 2

Por último, ninguna herramienta, por muy Open Source que sea, sirve de nada si no ha sido usada en aplicaciones reales; en la sección Sección 3 describimos una serie de aplicaciones que han usado el módulo `Algorithm::Evolutionary`.

Finalmente, termSección 4inaremos con una sección () en la que expondremos las líneas futuras de trabajo en esta biblioteca.

2. Estado del arte

Los algoritmos evolutivos son métodos de optimización estocásticos basados en someter a un grupo de estructuras de datos que representan soluciones a un problema determinado a una serie de operadores inspirados en la teoría de la evolución de Darwin: mutaciones ciegas, recombinación, y selección de los más adecuados. Un algoritmo evolutivo procedería de la forma siguiente:

1. Inicializar una población de `n` individuos aleatoriamente.
2. Evaluar los individuos, asignándoles un *fitness*, que representa su adecuación como soluciones del problema.
3. Escoger una parte de la población, que servirán como progenitores de la siguiente generación.
4. Someter a los progenitores a operadores de variación unarios (individuo por individuo) o n-arios (que usan varios individuos, produciendo a su vez uno o varios individuos).
5. Insertar en la población estos nuevos individuos, sustituyendo, siguiendo algún criterio, una parte de la población.
6. Vuelta al segundo paso.

Actualmente, se considera que todos los algoritmos evolutivos forman parte de una misma familia, pero hay diferentes tribus que difieren en cosas como las estructuras de datos que usan para representación, operadores de variación, y método de selección, tal como se ve en la tabla siguiente. En realidad, es difícil encontrar un algoritmo de los anteriores en estado puro, aunque cada cual designa su algoritmo con la tribu a la que más se parece:

Tabla 1. Tribus de algoritmos evolutivos

Algoritmo	Representación del problema	Operadores de variación	Métodos de selección
Algoritmos genéticos (Goldberg)	Cadena binaria	Mutación y crossover	Selección de rueda de ruleta (a veces con elitismo)
Estrategias de Evolución (Rechenberg/Schwefel)	Vector de reales + desviaciones estándar	Mutación gaussiana y crossover aritmético (diferentes tipos)	Diferentes tipos de selección: lambda,mu; lambda+mu...
Programación evolutiva (Fogel)	Números reales	Mutación	Diversos tipos de selección
Programación genética (Koza)	Expresiones-S de LISP representadas habitualmente como árboles	Crossover, algo de mutación	Diversos tipos de selección

Hasta julio de 2002, había pocos módulos Perl publicados suficientemente flexible para implementar la gama completa de algoritmos evolutivos; la mayor parte de los esfuerzos han sido destinados a implementar programación genética en Perl. El primer trabajo publicado sobre el tema es aparentemente el de Baldi y otros [baldi00], aunque el código en sí no fue publicado, y no podemos aventurar ninguna hipótesis sobre sus características. En todo caso, posteriormente se han publicado una serie de artículos que mencionan una implementación de Perl de un algoritmo evolutivo: Kunken [glotbot01] describe una aplicación en la que hace evolucionar palabras que parecen escritas en inglés, a base de tratar de generarlas con la misma frecuencia que en inglés. Esa misma aplicación es también mencionada por Zlatanov en [zlatanov01], en donde implementa un sistema de programación genética para realizar la misma tarea.

Quizás el primer artículo que describe programación genética en Perl fue escrito por Murray y Williams [murray99-ga], que, a pesar de su título, describen un sistema de programación genética, similar al mencionado en un artículo del sitio Perl Monks ([gumpu01]). Otro módulo publicado en 2001 [ag01] implementa un algoritmo genético canónico, sin muchas posibilidades de ampliación ni de adaptación.

Ninguno de los sistemas mencionados anteriormente usan plenamenta las capacidades del Perl para diseñar una biblioteca orientada a objetos, adaptable y fácil de ampliar, tal como se ha pretendido que sea `Algorithm::Evolutionary`. El producto que más se acerca ha aparecido durante el año 2002, y se denomina myBeasties [myBeasties]. Este módulo implementa un mecanismo general de evolución de genotipos, y de asignación de fenotipos a genotipos, así como un método para serialización usando ficheros de texto; los algoritmos (tipos de cromosomas y operadores que se le van a aplicar) se pueden describir mediante un fichero de texto. Aparentemente, trabaja simplemente con cromosomas (que llama blobs) lineales, y, además, los operadores están asociados al cromosoma, no son independientes. Aunque el módulo es bastante general, nosotros lo hemos encontrado un tanto complicado; habrá que ver cómo evoluciona en el futuro.

Por otro lado, teniendo en cuenta que `Algorithm::Evolutionary` se ha concebido desde el principio para trabajar con XML como formato de serialización, habrá que prestar atención a cómo se ha usado XML dentro de

los algoritmos evolutivos: pues bien, sucede que el reino del XML aplicado a algoritmos genéticos se ha mantenido tradicionalmente aparte del anterior, aunque ha habido unos cuantos acercamientos al tema. El más notable es el lenguaje EAML, descrito por Veenhuis y otros [EAML]. Este lenguaje, especificado como un grupo de etiquetas en XML con un significado concreto y determinado, especifica un algoritmo evolutivo, a partir del cual se puede generar código en C++. Este lenguaje trata de ser una descripción exhaustiva de un algoritmo evolutivo, pero no lo consigue, porque, por ejemplo, los operadores de variación son etiquetas, en vez de ser atributos; los atributos pueden tener cualquier valor, pero sólo se pueden usar las etiquetas especificadas en el diccionario; de esta forma, la introducción de un nuevo operador significaría la modificación del diccionario. Tiene otros errores de diseño, pero, en todo caso, es un paso en la dirección correcta. El dialecto usado en esta implementación en Perl/XML de un algoritmo genético trata de evitar estos errores.

En los primeros meses del año 2002 han surgido otros muchos esfuerzos por combinar XML y algoritmos evolutivos. Por ejemplo, se ha propuesto un servicio web basado en algoritmos evolutivos [eaWeb], al cual se proporciona una descripciónd de las funciones a optimizar, y los parámetros del algoritmo evolutivo usando XML, y devuelve la función optimizada. En el wiki XMLMen [XmlMen] también se exploran los diferentes paradigmas de especificación de algoritmos evolutivos usando XML (inclusive una versión previa del usado en `Algorithm::Evolutionary`), usando finalmente EAML [EAML] para hacer diferentes experimentos dentro de un sistema denominado Sutherland. Otros sistemas basados en algoritmos evolutivos prometen, para versiones sucesivas, especificaciones XML de los algoritmos o bien serialización XML. También eaLib [eaLib] incluye un método de serialización XML, pero se trata de un método automático de serialización de objetos en Java, no uno específico que sea legible para alguien entendido en algoritmos evolutivos.

Importante

En resumen, `Algorithm::Evolutionary` pretende llevar un paso más allá el estado del arte, creando una biblioteca bien integrada con XML, flexible, que admita cualquier paradigma de computación evolutiva, que sepa aprovechar las características del lenguaje, y, finalmente, con una licencia libre. Hay que hacer notar que, en nuestra biblioteca, XML no es un procedimiento para dar los parámetros a un programa, sino una forma alternativa de representar las estructuras de datos del programa.

3. Descripción de la biblioteca

Dado que en los algoritmos evolutivos existen dos tipos de objetos, cromosomas (o simplemente objetos), es decir, estructuras de datos que irán cambiando para ser una solución lo más buena posible del problema, y operadores, que actúan sobre grupos de objetos, la jerarquía de clases de `Algorithm::Evolutionary` reflejará este hecho, tanto en el diseño de la biblioteca, como de las etiquetas XML que se usarán para representarla. En su actuado actual, esta biblioteca supone una mejora sobre su versión anterior, OPEAL [opeal-aeb02], aunque las reglas básicas de diseño son las mismas.

Empecemos con lo más simple: un individuo. Un individuo, en computación evolutiva, es lo que evoluciona, es decir, la estructura de datos que, en en contexto de una población, va sufriendo cambios hasta que, cuando se cumple alguna de las condiciones de terminación, se extrae la mejor. Un individuo se puede definir de la siguiente forma:

```
my $indi = new Algorithm::Evolutionary::Individual::BitString 10;
```

lo cual crearía una cadena de bits de longitud 10. Un individuo de la misma clase se puede definir de la forma siguiente:

```
<indi type='BitString'>
    <atom>0</atom><atom>1</atom><atom>0</atom><atom>1</atom><atom>1</atom>
</indi>
```

que luego se puede deserializar de la forma siguiente:

```
my $indibis =  Algorithm::Evolutionary::Individual::Base->fromXML( $xml  )
```

si es que hemos asignado la cadena anterior a la variable `$xml`; `$indibis` será un inviduo de la clase adecuada, deducida a partir del atributo `type` del elemento XML. Generalmente, los individuos se crean en grupos, y para ello lo más razonable es usar un objeto `Creator`, que se instancia de la forma siguiente:

```
my $creator =  new Algorithm::Evolutionary::Op::Creator( 20, 'BitString', { length => '10' });
```

. Un `Creator` recibe como parámetros el número de elementos que va a crear, el nombre de la clase, y los parámetros adicionales que necesita el individuo en forma de hash. A partir de este `Creator`, se crea la población completa de individuos de la forma siguiente:

```
my @pop;
$creator->apply( \@pop )
```

Los operadores se aplican a individuos y/o devuelven individuos (como el `Creator` mencionado anteriormente). Su uso es similar al caso de los individuos: se pueden instanciar de forma independiente, o bien a partir de su descripción en XML. Sin embargo, los operadores son estructuras de datos más complejas, que en algún caso incluye código (para evaluar a los individuos), y otros operadores. Un operador complejo se representaría en XML de la forma siguiente:

```
        <op name='Easy'  type='unary'>
        <param name='selrate' value='0.4' />
        <param name='maxgen' value='1' />
        <code type='eval' language='perl'>
       <src><![CDATA[ my $chrom = shift;
                my $str = $chrom->Chrom();
                my $fitness = 0;
                my $blockSize = 4;
```

```
             for ( my $i = 0; $i < length( $str ) / $blockSize; $i++ ) {
            my $block = 1;
            for ( my $j = 0; $j < $blockSize; $j++ ) {
              $block &= substr( $str, $i*$blockSize+$j, 1 );
            }
            ( $fitness += $blockSize ) if $block;
             }
             return $fitness;
        ]]></src>
      </code>
      <op name='Bitflip' type='unary' rate='1'>
          <param name='probability' value='0.5' />
          <param name='howMany' value='1' />
      </op>
      <op name='Crossover' type='binary' rate='5'>
        <param name='numPoints' value='2' />
      </op>
    </op>
```

1. Declaración de los parámetros para el operador
2. Declaración del código de la función `eval`, se indica que está escrito en Perl y se incluye el código, que se convertirá en una subrutina en Perl
3. Declaración de un operador de mutación, que será aplicado a los individuos dentro del algoritmo. El cómo se aplique el operador dependerá del objeto.
4. Declaración de un operador de Crossover

Los individuos se combinan en una población, a la cual se aplicarán una serie de operadores secuencialmente, para dar lugar a la población final. El operador de generación de la población y el de evolución de la población se combinan en un `Experiment` o experimento, de la forma siguiente:

```
my $onemax = sub {
  my $indi = shift;
  my $total = 0;
  my $len = $indi->length();
  my $i;
  while ($i < $len ) {
 $total += substr($indi->{'_str'}, $i, 1);
 $i++;
  }
  return $total;
};
```

```
my $ez = new Algorithm::Evolutionary::Op::Easy $onemax;
my $popSize = 20;
my $indiType = 'BitString';
my $indiSize = 64;

my $e = new Algorithm::Evolutionary::Experiment $popSize, $indiType, $indiSize, $ez;
```

; en este listado se crea un operador de evolución de la población (`Easy`, un operador fácil de usar que corresponde al algoritmo genético simple), y, posteriormente, con otros parámetros como el tipo y número de individuos que van a componer la población y su tamaño, se crea un objeto experimento. Ese objeto experimento se puede usar posteriormente para llevar a cabo la evolución de la forma siguiente:

```
my $popRef = $ez->go();
```
; que devolverá la población al final del experimento.

`Algorithm::Evolutionary` incluye individuos de tipo vector, cadena, cadena binaria y árbol (para programación genética), operadores para mutación y entrecruzamiento para cada uno de ellos, diferentes tipos de métodos de selección y de terminación, y algunas clases de utilidad (por ejemplo, una clase para hacer tiradas de "rueda de ruleta").

La extensión de `Algorithm::Evolutionary` es relativamente simple: habría que crear un individuo en alguna de las dos jerarquías, con la única restricción que se pueda serializar usando XML; en el caso de los operadores, hay que definir a qué tipo de individuo se aplica, y definirle el método `apply`. Cualquier otro método externo que se necesite, fuera de los operadores y estructuras de datos necesarios para la evolución, no tienen porqué seguir ningún tipo de convención.

Las ventajas que nos ha ofrecido Perl en todo esto han sido diferentes: aparte de que, evidentemente, un desarrolla mejor en el lenguaje que mejor conoce, la "orientación a objetos" de Perl es bastante heterodoxa, pero muy flexible: se puede determinar el tipo de un objeto en tiempo de ejecución, e instanciarlo sabiendo únicamente el nombre; en lenguajes como Java se puede hacer también, pero en C++ sería poco menos que imposible, salvo que se compare el nombre con una lista de nombres preexistente, y se modifique el código para cada nuevo tipo que se añada. Por otra parte, Perl permite serialización de código incluido dentro de un objeto: el código de las funciones de evaluación se almacena dentro de los objetos como un puntero a código; y para serializarlo sólo hay que llamar a una función del módulo `B::Deparse` para obtener las fuentes: Perl permite decompilar en tiempo de ejecución. Esto, que sepamos es difícil, o probablemente imposible, en lenguajes no interpretados. En cuanto a la eficiencia, es difícil comparar lenguajes diferentes, puesto que la programación de un benchmark no va a ser nunca exactamente igual, pero sí podemos decir que la velocidad en un PC normal es bastante aceptable.

Por otro lado, el hecho de preparar esta biblioteca para su inclusión en CPAN [http://www.cpan.org], el sitio web de módulos de Perl "oficiales", obliga a una disciplina que consiste en que el empaquetamiento e instalación del módulo debe ser siempre igual; se debe consultar con la comunidad para ver si existe algún otro módulo que cumpla la misma necesidad, y, además, se aconseja que se diseñen una serie de tests que se ejecutan a la hora de la instalación. El diseño de estos tests ha obligado a ser sistemático en el desarrollo del módulo, y también a evitar ciertos errores tras su ejecución sistemática. Con los tests se asegura que, al menos en lo fundamental, la biblioteca funciona.

Importante

En resumen, el diseñar un módulo alrededor de su serialización en XML ha hecho que se corresponda la jerarquía de clases (y los espacios de nombres) con las etiquetas en XML, y que cada nuevo objeto necesite tener una función para serializar/deserializar desde XML. A la vez, el trabajar con un lenguaje sin tipificación fuerte e interpretado, como el Perl, permite bastantes ventajas a la hora de esta serialización, permitiendo incluso hacerlo con el propio código, sin que signifique un handicap importante sobre la velocidad de ejecución.

4. Aplicaciones

Este módulo ha sido usado en una serie de aplicaciones, que describimos a continuación:

- Una de las primera aplicaciones usó el módulo `SOAP::Lite` [soap-aeb02] para crear un algoritmo evolutivo paralelo usando SOAP, en este caso, usar Perl permitió usar esa biblioteca, que, dentro de las implementaciones de SOAP, es de las más fáciles de usar.

- También se usó, en conjunción con los módulos de Perl para acceso a BDs, para la asignación de revisores a trabajos para un congreso, el PPSN2002 [http://ppsn2002.ugr.es]

- Una aplicación a la cual se puede acceder online es la generación de palabras [http://geneura.ugr.es/Mason/EvolveWords.html] , de la cual se ha hablado en Barrapunto [http://barrapunto.com/article.pl?sid=01/09/05/179212]. En este caso, el código en Perl se usa en combinación con la biblioteca `HTML::Mason` para incluir el algoritmo evolutivo dentro de la misma página.

- Unos alumnos de proyecto de fin de carrera la han usado para generación de equipos para jugar a la liga fantástica de Yahoo [http://es.fantasysports.yahoo.com/es_fb], donde han probado diferentes estrategias para generación de equipos: optimización en cada jornada, u optimización de las reglas de cambio de jugadores usando información procedente de varias jornadas; ésta última ha sido la que ha tenido más éxito.

Importante

Aunque evidentemente, todavía no se trata de un producto maduro, la biblioteca tiene una calidad de producción adecuada para trabajar en una serie de proyectos que necesiten algoritmos evolutivos.

5. Conclusiones, discusión y trabajo futuro

En este trabajo se ha presentado una visión general de la biblioteca `Algorithm::Evolutionary`, una biblioteca de algoritmos evolutivos en Perl fuertemente integrada con XML, que además proporciona un dialecto de XML para intercambio de información entre diferentes bibliotecas de algoritmos evolutivos. Se ha mostrado cuál es la estructura general de la biblioteca y sus aplicaciones.

Una de las principales dificultades a las que nos hemos enfrentado es precisamente la serialización usando XML. El módulo que se usó en un principio, `XML::Simple`, distaba mucho de ser simple, y dependiendo del número de etiquetas, se comportaba de forma diferente. Actualmente se está considerando la posibilidad de usar algún otro módulo.

Pero todavía queda mucho trabajo por hacer. Por ejemplo, haría falta un interfaz gráfico, para que se pudieran seleccionar las opciones del algoritmo evolutivo de forma gráfica. La parte de programación genética está todavía en cuadro, y falta por desarrollar. Algunos módulos del OPEAL antiguo, especialmente los de programación paralela usando SOAP, no han sido portados a la nueva biblioteca. Evidentemente, hará falta también subirla a CPAN. Y, finalmente, harían falta más aplicaciones y demos online para demostrar la utilidad del módulo.

Sugerencia

La biblioteca está disponible en SourceForge [http://opeal.sourceforge.net], de dónde se puede bajar la última versión (la 0.5 en julio 2002), participar en los foros, solicitar ayuda, y, por supuesto, ofrecerla. Por el momento, no se puede decir que sea extraordinariamente popular, porque ha tenido aproximadamente unas 100 descargas, aunque ha sido generalmente bien acogida entre los usuarios de la lista de Perl e Inteligencia artificial.

Quiero agradecer a los organizadores de HispaLinux el haberme obligado a aprender el DocBook este, lo cual me ha abierto un mundo entero de posibilidades.

Referencias

[] *Genetic Algorithms + Data Structures = Evolution programs*. Ficha en Amazon [http://www.amazon.com/exec/obidos/ASIN/3540606
Zbigniew Michalewicz. Springer-Verlag. 1996. 3rd (revised).

[baldi00] M Baldi , F Corno , M Rebaudengo , M Sonza Reorda , y G Squillero.. *GA-Based Verification of Network Protocols Performance. Telecommunications Optimization: Heuristic and Adaptive Computation Techniques by David Corne (Editor) - George Smith - Martin J. Oates*. 2000. Wiley.

[myBeasties] James Hugman. *Project: myBeasties*. An Object Ecosystem: Document Manager. Disponible en http://sourceforge.net/docman/?group_id=26958 [http://sourceforge.net/docman/?group_id=26958].

[glotbot01] Joshua Kunken.. *The application of genetic algorithms in english vocabulary generation.*. In Proceedings of the Twelfth Midwest Artificial Intelligence and Cognitive Science Conference 2001. Miami University Press 2001. Available also from http://www.ocf.berkeley.edu/~jkunken/glot-bot [http://www.ocf.berkeley.edu/~jkunken/glot-bot]. 2001.

[zlatanov01] Teodor Zlatanov.. *Cultured perl : Genetic algorithms applied with perl create your own darwinian breeding grounds. Available from http://www-106.ibm.com/developerworks/linux/library/l-genperl/ [http://www-106.ibm.com/developerworks/linux/library/l-genperl]* . August 2001..

[murray99-ga] Brad Murray y Ken Williams.. *Genetic algorithms with perl.*. The Perl Journal, 5(1) Also available from http://mathforum.org/~ken/genetic/article.html [http://mathforum.org/~ken/genetic/article.html] . Fall 1999..

[gumpu01] gumpu. *Genetic programming or breeding perls, Available from PerlMonks [http://perlmonks.org/index.pl?node_id=31147]* September 2001.

[ag01] Michael K. Neylon. *Algorithm::geneticAvailable from PerlMonks [http://perlmonks.org/index.pl?node_id=81678]*. May 2001..

[EAML] Christian Veenhuis , Katrin Franke , y and Mario Köppen.. *A semantic model for evolutionary computation.*. En *Proceedings IIZUKA*, 2000..

[eaWeb] Vladimir Filipovic. *Evolutionary Algorithm Web Service Available from its website [http://www.matf.bg.ac.yu/~vladaf/EaWeb/index_e.html]*. Junio 2002.

[XmlMen] David Gjerdingen. *XmlMen Project Report Disponible en el wiki de su curso [http://tyvex.mrs.umn.edu/twiki/view/CSci4553/XmlMenProjectReport]*. Mayo 2002.

[eaLib] Andreas Rummler. *eaLib Disponible en la página web del autor [http://www.inf-technik.tu-ilmenau.de/~rummler/eng/ealib.html]*. 2002.

[opeal-aeb02] Juan J. Merelo-Guervós . *OPEAL, una librería de algoritmos evolutivos en Perl* . Publicado en las Actas del Primer Congreso Español sobre Algoritmos Evolutivos y Bioinspirados, AEB'02, Mérida, Febrero 2002. 54-59. 84-607-3913-9.

[soap-aeb02] Juan J. Merelo-Guervós , Pedro Ángel Castillo Valdivieso, y Javier García Castellano. *Algoritmos evolutivos P2P usando SOAP* . Publicado en las Actas del Primer Congreso Español sobre Algoritmos Evolutivos y Bioinspirados, AEB'02, Mérida, Febrero 2002. 31-37. 84-607-3913-9.