Procesamiento del Lenguaje Natural, núm. 35 (2005), pp. 187-194 recibido 29-04-2005; aceptado 01-06-2005

Uso flexible de soluciones evolutivas para tareas de Generación de Lenguaje Natural

Raquel Hervás **Pablo Gervás**
Facultad de Informática,
Universidad Complutense de Madrid
28040 Madrid
raquelhb@fdi.ucm.es, pgervas@sip.ucm.es

Resumen: En este artículo se discute la necesidad, a la hora de construir aplicaciones de generación de lenguaje natural, de disponer de la posibilidad de ensamblar módulos que utilicen técnicas distintas para cada una de las tareas específicas de la generación. Se plantea una arquitectura genérica que proporciona esa posibilidad y se describe su aplicación a un problema concreto, para el que se presentan ejemplos que utilizan distintas técnicas (una algorítmica y otra evolutiva) para resolver las tareas concretas de generación de expresiones de referencia y de agregación de un tipo concreto.
Palabras clave: generación de lenguaje natural, técnicas evolutivas, ingeniería del software

Abstract: This paper discusses the necessity, when implementing natural language generation applications, of disposing the possibility of assembling modules that use different techniques for each of the specific tasks of the generation process. We propose a generic architecture that provides this possibility and its applicability to a specific problem is described. We present examples of use of different techniques (an algorithmic one and an evolutive one) to deal with the specific tasks of Referring Expression Generation and Aggregation of a particular type.
Keywords: natural language generation, evolutionary techniques, software engineering

1. *Introducción*

La generación de lenguaje natural (GLN) se subdivide en varias tareas concretas (Reiter y Dale, 2000), y cada una de ellas opera a un nivel distinto de representación lingüística (discurso, semántica, léxico, sintaxis...). La GLN se puede aplicar en dominios donde los objetivos de comunicación y las características de los textos a generar son muy distintos, desde la transcripción a lenguaje natural de contenidos numéricos (Goldberg, Driedger, y Kittredge, 1994), a la generación de textos literarios (Callaway y Lester, 2001).

Cada tipo de aplicación de la GLN puede necesitar una organización del sistema en módulos distinta (DeSmedt, Horacek, y Zock, 1995). Incluso dada una organización concreta (o *arquitectura*) del sistema, puede ocurrir que distintos tipos de aplicación requieran soluciones diferentes a la hora de acometer cada una de las tareas específicas involucradas en el proceso de generación. Para una misma tarea, en procesos en los que sea necesaria una respuesta corta y rápida (por ejemplo, en casos de comunicación interactiva entre el usuario y el sistema en tiempo real) puede interesar utilizar soluciones simples basadas en heurísticas, que den respuesta rápidamente aunque la calidad alcanzada no sea muy alta, mientras que en casos en que se pretenda generar textos largos de calidad sin restricciones sobre el tiempo invertido en hacerlo puede ser más útil recurrir a técnicas basadas en conocimiento que ponderen exhaustivamente más posibilidades.

El desarrollador que se plantea añadir un módulo de generación de lenguaje natural a una aplicación existente no contempla la posibilidad de dedicar largo tiempo al desarrollo a medida de su solución. Para conseguir que el uso de este tipo de aplicaciones se extienda, se hace necesario el proporcionar algún tipo de esqueleto que facilite esa labor, posiblemente que incluya distintas opciones ya preparadas de módulos capaces de resolver tareas concretas, utilizando distintas técnicas, y que resulten fácilmente ensamblables para obtener una solución operativa. Un es-

ISSN: 1135-5948 © 2005 Sociedad Española para el Procesamiento del Lenguaje Natural

queleto de este tipo es el que se plantea en este artículo.

2. Generación de Lenguaje Natural y Algoritmos Evolutivos

Para el trabajo que nos ocupa vamos a centrarnos en dos de las tareas de la GLN: la Generación de Expresiones de Referencia y la Agregación.

Según Reiter y Dale (1992), una expresión de referencia debe comunicar suficiente información para ser capaz de identificar unívocamente al referente en el contexto del discurso actual, pero siempre evitando modificadores redundantes o innecesarios. Reiter y Dale proponen un algoritmo de generación de frases nominales para identificar objetos en el foco de atención del lector u oyente.

Kibble y Power (2000) proponen un sistema para planificar textos coherentes y escoger las expresiones de referencia. Afirman que la planificación textual y sintáctica debe estar parcialmente dirigida por el objetivo de mantener la continuidad referencial, incrementando las oportunidades de usos no ambiguos de los pronombres.

Palomar et al. (2001) también presentan un algoritmo para identificar frases nominales precedentes de pronombres personales, demostrativos, reflexivos y omitidos en textos en español. Definen una lista de restricciones y preferencias para los distintos tipos de expresiones pronominales, y destacan la importancia de cada tipo de conocimiento (léxico, morfológico, sintáctico y estadístico) en la resolución de expresiones anafóricas.

La agregación (Reape y Mellish, 1999) se encarga de decidir cómo se debe compactar la representación de la información en un texto dado. Esta tarea opera a muchos niveles lingüísticos, pero aquí sólo vamos a considerar su aplicación sobre conceptos y sus correspondientes atributos. La agregación es siempre deseable, pero hay que tener cuidado de no obtener textos fuertemente adjetivados cuando la información a mostrar es muy densa en cuanto a atributos. Una buena solución será encontrar el equilibrio entre concisión y un estilo aceptable.

Los Algoritmos Evolutivos (AEs) engloban un amplio conjunto de técnicas de resolución de problemas complejos inspiradas en los mecanismos de la evolución y selección natural (Holland, 1992), las cuales mantienen una población de individuos (soluciones potenciales del problema) que evolucionan de acuerdo a reglas de selección y operadores genéticos como son el cruce y la mutación (Figura 1). La función de aptitud es una métrica que permite evaluar cuantitativamente cada solución candidata, de forma que las expectativas son que la aptitud media de la población se incrementará en cada generación y, por tanto, repitiendo el proceso de evolución cientos o miles de veces, pueden descubrirse soluciones muy buenas para el problema (Goldberg, 1989). De esta forma, los AEs combinan la búsqueda aleatoria, dada por las transformaciones de la población, con una búsqueda dirigida, dada por la selección.

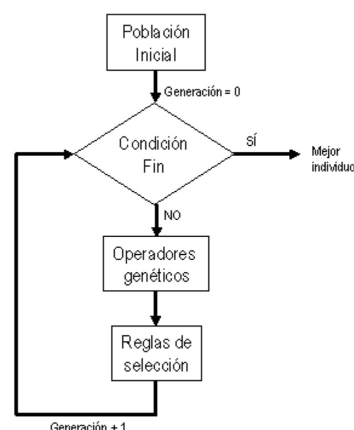

Figura 1: Esquema básico de un algoritmo evolutivo

Las técnicas evolutivas han demostrado estar particularmente bien preparadas para la generación de poesía. Los trabajos de Manurung (2003) y Levy (2001) proponen diferentes modelos computacionales para composición de poesía basada en aproximaciones evolutivas. En ambos casos, la principal dificultad radica en la elección de la función de adaptación para guiar el proceso. Aunque Levy sólo utiliza un modelo simple relacionado con la información silábica, su descripción general de la arquitectura en términos de una población de borradores de poemas que evolucionan, dando prioridad a los borradores que son mejor evaluados, es una idea a tener en cuenta. Levy usa una red neuronal, entrenada con ejemplos de versos válidos, para evaluar los poemas. Por otro lado, el trabajo de Manurung abarca la tarea completa, y presenta un conjunto de evaluadores que clasifican las soluciones candidatas de acuer-

do a ciertas heurísticas.

3. Uso flexible de módulos GLN

Hay muchas formas de organizar un sistema de Generación de Lenguaje Natural (GLN), y las ventajas y desventajas de cada una de ellas son todavía campo de discusión (DeSmedt, Horacek, y Zock, 1995; Reiter, 1994). Podemos encontrar muchas arquitecturas con grandes diferencias en cuanto a la división en módulos y la topología de conexiones de estos módulos entre sí. En (DeSmedt, Horacek, y Zock, 1995) varias de estas arquitecturas se discuten en detalle, y cada una de ellas demuestra tener sus méritos y desventajas. El desarrollador de una aplicación de GLN debe considerar un amplio rango de soluciones arquitecturales, ya que cada una de ellas será relevante para aspectos particulares de su problema.

3.1. Una arquitectura genérica para GLN

La arquitectura cFROGS (García, Hervás, y Gervás, 2004) pretende proporcionar a un posible diseñador de aplicaciones GLN la infraestructura necesaria para facilitar su labor al máximo, por medio de los esquemas y estructuras arquitectónicas genéricas usados comúnmente en este tipo de sistemas.

Teniendo en cuenta la división en módulos, podemos encontrar desde una arquitectura integrada (Kantrowitz y Bates, 1992), donde el sistema está formado por un único módulo, hasta una arquitectura con módulos separados para cada una de las tareas de la GLN (Cahill et al., 2001). Considerando el flujo de control, por un lado podemos encontrar una arquitectura de pipeline (Reiter, 1994), donde los módulos son completamente independientes, y por otro una arquitectura de pizarra (Calder et al., 1999), donde los módulos colocan información en un espacio de almacenamiento común sin preocuparse de qué otros módulos van a acceder a ella. Finalmente, también hay que considerar que las aplicaciones GLN tienen el problema adicional de estar basadas en teorías lingüísticas específicas, y que estas teorías imponen restricciones adicionales a las arquitecturas generales desde el punto de vista de las estructuras de datos.

La biblioteca cFROGS parte de la idea de que para diseñar la arquitectura de cualquier sistema de GLN es necesario definir tres aspectos fundamentales:

- Conjunto de etapas o módulos que componen el sistema.
- Flujo de control que gobierna la arquitectura, encargado de decidir cómo se conectan los módulos y cómo se transfieren los datos de unos a otros.
- Estructuras de datos específicas que se intercambian los módulos.

Estos tres ejes no son completamente ortogonales, y existen dependencias entre ellos.

3.2. Una instanciación de cFROGS: ProtoPropp

Un ejemplo de instanciación de cFROGS es el módulo de generación de texto de ProtoPropp (Gervás et al., 2004). Éste es un sistema para la generación automática de historias asentado sobre una ontología que incluye las características de los conceptos y las relaciones existentes entre ellos. La estructura de datos que genera es un plan de trama, en el que se describe un esqueleto del argumento en términos de elementos de la ontología que maneja el sistema (personajes y sus atributos, escenarios del cuento, eventos que ocurren...). A partir de ahí, el subsistema GLN tiene que obtener la representación textual de la historia.

3.2.1. Estructuras de datos

Las clases abstractas proporcionadas por la librería nos permiten crear las estructuras de datos necesarias para nuestro sistema, siempre de forma que todos los módulos puedan trabajar sobre ellas según sea necesario. En nuestro caso es necesario definir una estructura de datos genérica para el texto que se está creando. Dentro de esta estructura o borrador se va almacenando toda la información recopilada durante el funcionamiento del sistema, desde el plan de trama inicial hasta el texto final, pasando por las representaciones intermedias que van generando las distintas etapas. Cualquier módulo puede acceder a toda la información que se ha ido acumulando durante el tratamiento del cuento en cuestión.

3.2.2. Conjunto de módulos

En esta ocasión el sistema GLN está compuesto por cinco módulos, que son los siguientes:

- **ContentDeterminationStory.** Decide qué hechos de la trama original van a incluirse en el texto final. Basándonos en el "registro histórico"de la información ya transmitida, decide qué información es redundante y la elimina.
- **DiscoursePlanningStory.** Organiza la información de manera que exista cierta coherencia: se presentan los personajes y se describen los escenarios. Genera la estructura retórica del cuento.
- **ReferringExpressionStory.** Elimina redundancias léxicas sin introducir ambigüedad, de modo que podamos referirnos a conceptos aparecidos previamente utilizando distintos términos, pronombres, etc., pero manteniendo siempre la claridad del mensaje. Las expresiones de referencia a utilizar para cada concepto se determinan usando una heurística muy simple: la primera vez que un concepto aparece en un párrafo el generador le asigna su nombre completo, y en el resto de los casos usa un pronombre.
- **LexicalizationStory.** Selecciona las palabras concretas que hacen referencia a los componentes del mensaje, utilizando términos léxicos para los conceptos estáticos, como personajes y escenarios, y plantillas para los que involucran verbos, proporcionando la estructura de las oraciones en las que van a aparecer.
- **SurfaceRealizationStory.** Se ocupa de completar las plantillas de la etapa anterior con los términos seleccionados, aplicando las reglas de puntuación y capitalización del idioma inglés.

3.2.3. El flujo de control

El flujo de control en forma de pipeline de ProtoPropp se puede ver en la Figura 2. A partir del plan de trama de la historia generada, el módulo de generación lleva a cabo las tareas ya mencionadas de Determinación de Contenido, Planificación del Discurso, Generación de Expresiones de Referencia, Lexicalización y Realización Superficial, cada una de ellas localizada en un módulo independiente.

3.3. EvoProtoPropp

Para demostrar la flexibilidad de la biblioteca cFROGS nos planteamos la idea de sustituir algunos de los módulos simples de ProtoPropp por otros que utilicen técnicas más elaboradas. En la mayor parte de las tareas parece difícil encontrar una heurística adecuada para todos los posibles casos, así que decidimos intentar soluciones basadas en Algoritmos Evolutivos.

Figura 2: Flujo de control del módulo de generación de ProtoPropp

Como primera aproximación al uso de técnicas evolutivas para la Generación de Lenguaje Natural decidimos seleccionar sólo algunas de las características del texto para probar estas ideas. Dada la complejidad de los cambios que son posibles en un texto a todos los niveles (sintaxis, semántica, estructura del discurso, pragmática, ...), es impracticable intentar ocuparse de todos ellos a la vez. Elegimos los problemas de la Generación de Expresiones de Referencia y la Agregación, que son adecuados para ser resueltos utilizando algoritmos evolutivos. El problema del método utilizado inicialmente para estas tareas es que dos apariciones del mismo concepto pueden estar muy separadas entre sí dentro del mismo párrafo, de forma que el lector quedará confundido debido a la distribución de las referencias. Para evitarlo hemos implementado un módulo evolutivo simple para las tareas mencionadas.

La entrada del módulo será el plan de trama o argumento de la historia que se quiere pasar a texto. La población del algoritmo evolutivo estará formada por borradores del texto final, donde las apariciones de los conceptos son consideradas los genes. La población inicial de textos se genera aleatoriamente, usando para cada concepto su nombre com-

pleto o el pronombre correspondiente. Cuando se usa su nombre completo, un subconjunto de los atributos que pertenecen a ese concepto es seleccionado. Estos atributos aparecerán junto al nombre del concepto, en nuestro caso delante del mismo como es usual en inglés.

El sistema trabaja sobre esta población durante el número de generaciones indicado por el usuario. En cada generación se usan tres operadores genéticos: cruce, mutación y agregación.

El operador de cruce selecciona dos textos y los cruza por un punto aleatorio de su estructura, de forma que cada uno de los textos hijos resultantes tendrá parte de cada uno de los padres.

El operador de mutación selecciona aleatoriamente ciertos genes y los muta, cambiándolos a pronombres o nombres completos con atributos según corresponda. Si el gen contiene un pronombre – como en *"she lived in a castle"* -, cambiará al correspondiente nombre completo, siempre con un subconjunto de sus posibles atributos – por ejemplo *"the princess lived in a castle"* o *"the pretty princess lived in a castle"* -. Si la referencia es un nombre completo – como en *"the pretty princess"* –, cambiará a un pronombre – en este caso *"she"* –, o cambiará el conjunto de atributos que le acompañan – por ejemplo *"the pretty blonde princess"* -. Una de estas dos opciones se escoge aleatoriamente.

El tercer operador se encarga de la agregación entre conceptos y sus atributos. Los atributos en el texto pueden aparecer acompañando al concepto correspondiente – como en *"The pretty princess"* – o en frases de tipo 'X is Y' – como en *"The princess is pretty"* –. El operador de agregación provocará ciertos cambios en la estructura del texto, ya que algunas frases del mismo deberán ser eliminadas si la información sobre atributos que proporcionan forma parte de una frase anterior. Además, este operador sólo actúa sobre los genes que contienen una referencia explícita a un concepto, no sobre conceptos mencionados con pronombres.

Al final de cada generación cada uno de los textos es evaluado, y una selección de los mismos pasa a la siguiente generación, de forma que los que tengan mayor valor de aptitud tendrán más posibilidades de ser escogidos. La clave del correcto funcionamiento del algoritmo genético radica en la elección de las funciones de evaluación que asignan a cada individuo de la población un valor de aptitud concreto. Para definir estas funciones se analizaron las características de los textos generados por seres humanos, siempre desde el punto de vista de las expresiones de referencia, y se encontraron cinco características de este tipo de textos que podían ser utilizadas para realizar la evaluación de la población. Estas funciones de evaluación pueden verse en el Cuadro 1.

A continuación se muestra cada una de las tres decisiones de diseño a tomar en la implementación de un sistema GLN.

3.3.1. Estructuras de datos

La población sobre la que trabaja el algoritmo es una población de borradores de texto. Cada uno de ellos tiene un conjunto de genes que representan las apariciones de conceptos en el texto, y sobre estos genes trabajan los operadores genéticos y las funciones de evaluación.

La librería nos proporciona clases abstractas para las estructuras de datos. En este caso ha sido necesario instanciar estas clases abstractas de dos formas. La primera para crear la clase que contiene cada uno de los textos que forman la población, y la segunda para formar la población de individuos a partir de estos textos.

Gracias a las facilidades proporcionadas por el framework hemos podido comprobar que no supone ningún problema pasar a trabajar con un conjunto de textos en lugar de con uno sólo.

3.3.2. Conjunto de módulos

La biblioteca cFROGS también proporciona clases abstractas para la implementación de los módulos de nuestro sistema. Todos los módulos tendrán la misma interfaz, recibiendo como entrada los datos abstractos también definidos por el framework. Así no hay problemas a la hora de conectar los módulos entre sí, ya que todos proporcionan siempre el mismo tipo de salidas y reciben el mismo tipo de entradas.

Tomando como referencia el flujo de control de un algoritmo evolutivo, representado en la Figura 1, se han identificado los siguientes submódulos para el módulo de Generación de Expresiones de Referencia y Agregación:

- **InitializerStory.** Su entrada es la estructura del discurso para texto que han

Correct Referent	$err_1 = \sum$ referencias pronominales de un concepto no referido en los dos genes previos
Redundant Attributes	$err_2 = \sum$ frases tipo "<adj> X is <adj>"
Reference Repetition	$err_3 = \sum$ uso repetitivo del conjunto de atributos $att(gen_i)$ para referir el concepto en gen_i
Coherence	$err_4 = \sum_{i=1}^{N}(att(gen_i) - I)$ con I el conjunto de atributos ya usados para el concepto en gen_i
Overlooked Information	$err_5 = \sum$ subconjunto de atributos del concepto i en la ontología no mencionados en el texto

Cuadro 1: Definición de las funciones de aptitud

generado las etapas anteriores de GLN. Este esquema del texto se copia tantas veces como sea necesario hasta formar una población con el número de individuos especificado por el usuario. Una vez creada la población es necesario inicializarla desde el punto de vista de las expresiones de referencia: para cada aparición de un concepto se escogerá aleatoriamente una referencia con el nombre completo acompañado de atributos, o sólo con el pronombre correspondiente.

- **ReferringExpressionStory.** Aplica aleatoriamente sobre la población los tres operadores genéticos: cruce, mutación y agregación. Cada uno de ellos escoge, a partir de ciertas probabilidades asociadas, algunos de los individuos de la población para mutar o agregar algunos de sus genes, o bien para cruzarlos en el caso del operador de cruce. Algunas de las referencias cambiarán aleatoriamente, y habrá que evaluar si este cambio es favorable o no.
- **EvaluatorStory.** Se encarga de la evaluación de los individuos de la población, calculando el valor de aptitud para cada uno de los borradores de texto usando las funciones de evaluación definidas previamente. A partir de la población evaluada, es necesario seleccionar qué individuos pasan a la siguiente generación escogiendo aleatoriamente de la población actual tantos individuos como sea necesario para crear una nueva población, pero siempre de tal forma que los borradores con mayor valor de aptitud tengan más posibilidades de ser escogidos.
- **ResultStory.** Una vez que se ha ejecutado el número de generaciones especificadas por el usuario, se encarga de extraer de la población el mejor individuo, es decir, el borrador de texto con mayor valor de aptitud. Sobre este borrador se podrían ejecutar las etapas restantes del flujo de control para dar lugar al texto final.

3.3.3. El flujo de control

A partir de los submódulos del apartado anterior, es necesario definir el flujo de ejecución de los mismos para que formen el algoritmo evolutivo que se está buscando. La biblioteca de clases cFROGS facilita la definición de flujos de control atípicos, como el caso que nos ocupa.

El flujo de control del módulo evolutivo se encarga de pasar la salida de etapas anteriores de GLN al módulo inicializador *InitializerStory* para formar la población inicial. A partir de ella, los módulos *ReferringExpressionStory* y *EvaluatorStory* se ejecutan en un bucle el número de generaciones definidas por el usuario del sistema. Una vez acabado dicho número de iteraciones, el módulo *ResultStory* devuelve como salida el mejor borrador conseguido por el algoritmo evolutivo. Esta salida podría ser utilizada como salida final del sistema, o como entrada en sucesivos módulos de Lexicalización y Realización Superficial para formar el texto final.

4. Discusión

La arquitectura de cFROGS ha permitido la sustitución del módulo del sistema ProtoPropp que realizaba la tarea de generación de expresiones de referencia utilizando técnicas algorítmicas básicas por otro módulo similar que aplica técnicas evolutivas. Estas técnicas han demostrado su potencial para optimizar los resultados conseguidos en esas tareas concretas. El resultado positivo de este experimento abre las puertas a plantear la aplicación de soluciones similares a otras tareas de GLN, como puedan ser la generación de contenidos – en la que la capacidad de las técnicas evolutivas de encontrar soluciones sorprendentes puede ser una aportación interesante para determinadas aplicaciones –, la planificación del discurso – que involucra la construcción de árboles complejos en los que deben ser optimizadas las referencias anafóri-

cas de unas partes del discurso a otras –, o la lexicalización – en la que el uso de técnicas evolutivas presenta la ventaja de que permitiría aplicar criterios definidos globalmente, como funciones de evaluación sobre el total de un texto a la hora de validar o modificar decisiones locales acerca de elecciones léxicas concretas para conceptos particulares.

4.1. Mejoras aportadas por la solución evolutiva

Los primeros experimentos realizados en la optimización de las tareas de Generación de Expresiones de Referencia y Agregación en ProtoPropp utilizando técnicas evolutivas son prometedores (Hervás y Gervás, 2005). En la Figura 3 se puede ver la evolución del valor de aptitud para los cinco cuentos con los que hemos trabajado. La gráfica representa la evolución de la aptitud según el tamaño de la población, tomando siempre 50 generaciones para el algoritmo. Los valores de aptitud suben cuanto mayor es la población, aunque se ha comprobado que con los mismos tamaños de población la aptitud también aumenta a mayor número de generaciones. También es importante comentar que los cuentos que presentan mayor pendiente de subida son aquellos con menor número de genes, y que por ello obtienen el valor máximo 1 con menor tamaño de población.

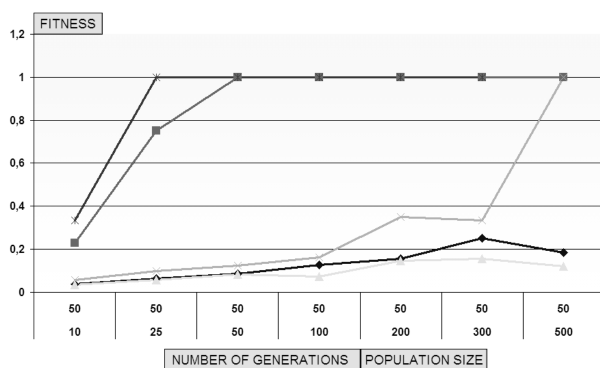

Figura 3: Evolución de los valores de aptitud con 50 generaciones

Como ejemplo de las mejoras obtenidas, el texto resultante para un cuento utilizando ProtoPropp sería el siguiente:

A princess lived in a castle. She loved a knight. She was pretty. She was blonde. It had towers. It was strong.

En este texto se pueden apreciar algunos de los problemas de la heurística utilizada para la generación de expresiones de referencia. El mayor de ellos es el uso de los pronombres, que confunde al lector al hacer referencias con pronombres de conceptos muy alejados de su primera mención.

Sin embargo en EvoProtoPropp, dentro de un cuento que ha alcanzado la aptitud máxima con 50 generaciones en una población de 500 individuos, el texto final es:

A pretty princess lived in a strong castle. She loved a brave knight. The princess was blonde. The castle had towers.

La mayor ventaja con la que nos encontramos es que los algoritmos evolutivos no necesitan reglas específicas para construir una solución, sino simplemente medidas de la calidad de dicha solución para el problema que se pretende resolver.

4.2. Valoración de la flexibilidad arquitectónica

La permutabilidad de módulos dentro de un flujo de control fijo estudiada aquí es solamente una de las posibilidades que ofrece una arquitectura como cFROGS. Vista la utilidad de la aplicación de técnicas evolutivas, sería interesante contemplar la posibilidad de aplicar una solución de este tipo a la generación como un todo. La arquitectura de cFROGS facilita este tipo de cambio concentrando las decisiones arquitectónicas afectadas en la implementación de un flujo de control evolutivo, que sustituiría a la secuencia de ejecución de módulos que enlaza las tareas clásicas de GLN.

Se está estudiando la viabilidad de utilizar la arquitectura de cFROGS para construir una solución basada en agentes que cooperen entre sí, cada uno encargado de realizar una tarea específica sobre el material lingüístico disponible (pronominalización, selección de referencia determinada o indeterminada, enriquecimiento mediante adjetivos,...). Esta opción permitiría que cada decisión pudiese reconsiderarse a la vista de decisiones tomadas por otros agentes sobre partes del material que afecten a sus datos de entrada. Por ejemplo, si la eliminación de un adjetivo en algún punto anterior de texto hace incorrecta una referencia anafórica posterior.

Bibliografía

Cahill, L., R. Evans, C. Mellish, D. Paiva, M. Reape, y D. Scott. 2001. The RAGS reference manual. Informe Técnico ITRI-01-07, Information Technology Research Institute, University of Brighton.

Calder, J., R. Evans, C. Mellish, y M. Reape. 1999. Free choice and templates: how to get both at the same time. En *May I speak freely? Between templates and free choice in natural language generation*, páginas 19–24, Saarbrücken.

Callaway, C. y J. Lester. 2001. Narrative prose generation. En *Proceedings of the 17th IJCAI*, páginas 1241–1248, Seattle, WA.

DeSmedt, K., H. Horacek, y M. Zock. 1995. Architectures for natural language generation: Problems and perspectives. En G. Ardoni y M. Zock, editores, *Trends in natural language generation: an artificial intelligence perspective*, LNAI 1036. Springer Verlag, páginas 17–46.

García, C., R. Hervás, y P. Gervás. 2004. Una arquitectura software para el desarrollo de aplicaciones de generación de lenguaje natural. *Procesamiento de Lenguaje Natural*, 33:111–118.

Gervás, P., B. Díaz-Agudo, F. Peinado, y R. Hervás. 2004. Story plot generation based on CBR. En A. Macintosh R. Ellis, y T. Allen, editores, *12th Conference on Applications and Innovations in Intelligent Systems*, Cambridge, UK. Springer, WICS series.

Goldberg, D.E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Goldberg, E., N. Driedger, y R.I. Kittredge. 1994. Using natural-language processing to produce weather forecasts. *IEEE Expert: Intelligent Systems and Their Applications*, 9(2):45–53.

Hervás, R. y P. Gervás. 2005. Applying genetic algorithms to referring expression generation and aggregation. En A. Quesada-Arencibia R. Moreno-Díaz, y J.C. Rodríguez, editores, *10th EuroCast*, Las Palmas de Gran Canaria, Spain.

Holland, J.H. 1992. *Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, Massachusetts, Second Edition.

Kantrowitz, M. y J. Bates. 1992. Integrated natural language generation systems. En R. Dale E. Hovy D. Rösner, y O. Stock, editores, *Aspects of Automated Natural Language Generation*. Springer Verlag, Berlin, páginas 13–28.

Kibble, R. y R. Power. 2000. An integrated framework for text planning and pronominalization. En *Proc. of the International Conference on Natural Language Generation (INLG)*, Israel.

Levy, R. P. 2001. A computational model of poetic creativity with neural network as measure of adaptive fitness. En *Proccedings of the ICCBR-01 Workshop on Creative Systems*.

Manurung, H.M. 2003. *An evolutionary algorithm approach to poetry generation*. Ph.D. tesis, School of Informatics, University of Edinburgh.

Palomar, M., A. Ferrández, L. Moreno, P. Martínez-Barco, J. Peral, M. Saiz-Noeda, y R. Muñoz. 2001. An algorithm for anaphora resolution in spanish text. *Computational Linguistics*, 27(4):545–567.

Reape, M. y C. Mellish. 1999. Just what is aggregation anyway? En *Proceedings of the 7th EWNLG*, Toulouse, France.

Reiter, E. 1994. Has a consensus NL generation architecture appeared, and is it psychologically plausible? En D. McDonald y M. Meteer, editores, *Proceedings of the 7th. IWNLG '94*, páginas 163–170, Kennebunkport, Maine.

Reiter, E. y R. Dale. 1992. A fast algorithm for the generation of referring expressions. En *Proceedings of the 14th conference on Computational linguistics*, Nantes, France.

Reiter, E. y R. Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press.