

# Distribución de Información en Algoritmos Evolutivos P2P

M.G. Arenas, P.A. Castillo, G. Romero, J.J. Merelo

*Resumen*— Este trabajo presenta las últimas investigaciones realizadas con la ayuda de JEO, *Java Evolutionary Objects* como parte integrante del proyecto DREAM, *Distributed Resources Evolutionary Algorithm Machine*, destacando las mejoras realizadas en los mecanismos de distribución de información estadística en algoritmos evolutivos basados en el modelo Isla. Al final del trabajo se demuestra como las mejoras realizadas disminuyen en gran medida el volumen de comunicaciones existente con el modelo anterior de distribución.

*Palabras clave*— Computación Evolutiva, EC, Computación Paralela, Sistemas P2P, Java Evolutionary Objects, JEO.

## I. INTRODUCTION

Hoy en día Internet está cambiando en muchos aspectos su utilización original de simple red de computadores para pasar a ser un sistema distribuido [14] de computación que no sólo se utiliza para transferir información de un lado a otro de la tierra sino que presta sus recursos para procesar datos como si fuera un gran laboratorio [34]. Prueba de esto es *Java Evolutionary Objects* (JEO) [2] [4] [3] con la que se pueden programar algoritmos de Computación Evolutiva (CE) que serán posteriormente ejecutados de forma distribuida utilizando los recursos que Internet ofrece.

Esta librería está incluida como parte de un proyecto mayor denominado *Distributed Resources Evolutionary Algorithms Machine*, (DREAM) [28] [1] cuyo principal objetivo es el desarrollo de un entorno completo para ejecutar experimentos de CE de forma distribuida, robusta y escalable. Este sistema de computación soporta aplicaciones que cumplen las siguientes características:

- Paralelismo, es decir, que el experimento pueda resolverse utilizando un conjunto de computadores.
- Asincronismo.
- Volúmenes de comunicación no muy grandes entre procesos.
- Robustez, es decir, que la solución global de la aplicación no dependa del éxito de un sólo proceso.

Estas características son precisamente las que cumplen los experimentos construidos con JEO

por lo que se prestan fácilmente a ser resueltos utilizando DREAM.

DREAM es un sistema organizado en cinco capas:

- DRM *Distributed Resource Machine* es la capa básica que independiza la arquitectura física que exista en la red de los niveles superiores. Esta capa ha sido ampliamente explicada en los trabajos presentados en [22][21].
- JEO es la capa inmediatamente superior a DRM y permite al usuario desarrollar experimentos de CE que serán distribuidos por DREAM utilizando el soporte proporcionado por DRM. Son posibles muchas otras librerías que hagan uso de DREAM.
- EASEA [12] es un lenguaje de especificación de experimentos de CE que toma como entrada las especificaciones introducidas en la capa superior, (GUIDE), y el conjunto de herramientas que proporciona JEO para así generar de forma automática un experimento. Esta capa se considera justo encima de JEO y debajo de la siguiente capa denominada GUIDE.
- GUIDE es un entorno gráfico que facilita la introducción de experimentos para los usuarios de DREAM. Esta capa es la capa de nivel superior y se sitúa justo encima de EASEA.
- La CONSOLA es la parte de DREAM que permite al usuario comenzar y controlar un experimento cualquiera. Se trata de una capa independiente que controla los experimentos que DRM está ejecutando.

Cada capa o parte de DREAM está orientado a un tipo de usuario determinado. Así la capa DRM es el punto de entrada a DREAM para usuarios expertos en Java y en aspectos de implementación a bajo nivel como pueden ser direcciones de red o conexión de sockets. JEO está orientada a programadores familiarizados con conceptos de CE y Programación Orientada a Objetos sin necesidad de preocuparse de aspectos más relacionados con el nivel físico. EASEA junto con GUIDE es el punto de entrada a DREAM para usuarios no programadores que comienzan a trabajar con CE. Por último *CONSOLE* da servicio a cualquier usuario anterior para controlar sus experimentos. En la figura 1 se puede apreciar la estructura general de todas las partes de

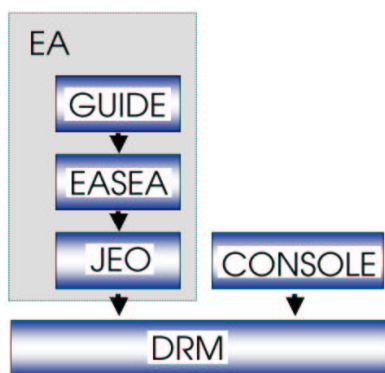


Fig. 1. Diagrama general de las capas de DREAM

## DREAM.

El propósito de este trabajo es presentar la forma de distribución de información que utilizan los experimentos desarrollados con JEO para mostrar así una nueva forma de computación distribuida aprovechando las ventajas que ofrece DRM como soporte básico para la distribución.

Este trabajo está organizado de la siguiente forma: La sección II realiza un estudio de las librerías de CE de las que se dispone actualmente. La sección III realiza un resumen de las principales características de JEO. La sección IV describe detalladamente el proceso de distribución pasiva de información. La sección V describe los experimentos realizados. Por último se incluyen conclusiones, sección VI, agradecimientos, sección VIII y el trabajo aún por realizar en la sección VII.

## II. ESTADO DEL ARTE

La fuente principal de inspiración para JEO es *Evolutionary Objects*, EO [19][18][24][9]. EO es una biblioteca de objetos C++ diseñada para evolucionar cualquier estructura de datos y su principal premisa es ser lo suficientemente flexible para permitir la evolución de cualquiera de estas estructuras. Sin embargo, EO está desarrollada en C++ y utiliza en muchas de sus clases técnicas de programación que hacen difícil su comprensión por lo que no es la herramienta más adecuada para principiantes o para investigadores no expertos en C++. Incluso en las últimas versiones presentadas de EO ya se presentan algunas características para la paralelización de aplicaciones aunque el hecho de estar desarrollada en C++ plantea problemas de exportación para redes de computadoras completamente heterogéneas como puede ser Internet.

Además de EO existen otras bibliotecas desarrolladas también en C++ como GALib [37] que utiliza PVM [16][35] (*Parallel Virtual Machi-*

*ne*) para paralelizar procesos. Sin embargo GALib presenta algunas características que no son modificables como la funcionalidad de los operadores que proporciona por lo que dependiendo de la necesidad del usuario puede ser poco flexible. Por último también hay que mencionar otras herramientas como EvolC [31], ECLib [23] y Open Beagle [15]. EvolC es una librería de funciones desarrollada en C pero no es comparable en extensión ni madurez a ninguna de las comentadas. ECLib es una herramienta realizada en C++ al igual que Open Beagle pero ninguna de las tres últimas opciones mencionadas ofrecen recursos para paralelización de procesos. El principal problema que plantean las bibliotecas de funciones o de objetos realizadas en C o en C++ son los problemas de portabilidad para la distribución de procesos por lo que el usuario de estas librerías debe tener en cuenta los recursos de los que dispone para elegir las como herramienta de desarrollo.

Además de C++ también Java es ampliamente utilizado como lenguaje de desarrollo. Algunas de las bibliotecas de objetos que utilizan Java son JDEAL [13], ECJ [27], JRGP [7], DGP [10][11], GPSYS [29], MAFRA [25] y GJGP [5]. De todas estas opciones, sólo JDEAL y ECJ son comparables con JEO puesto que el resto están diseñadas sólo para construir experimentos de Programación Genética, (PG) excepto MAFRA que es una herramienta especializada para Algoritmos Genéticos, (AG) Híbridos por lo que tampoco se trata de una herramienta de CE sino más bien una herramienta de AG que solo evoluciona individuos binarios.

JDEAL permite a los usuarios distribuir algunas tareas a través de redes de computadoras heterogéneas; además, posee un paquete estadístico que facilita la recolección de resultados pero el paradigma de computación distribuida que utiliza está basado en el modelo Maestro-Esclavo por lo que cuando la cantidad de procesos paralelos aumenta, la ganancia del modelo decrece presentando grandes problemas de escalabilidad. Otra característica a tener en cuenta de JDEAL es que está especialmente diseñada para Algoritmos Genéticos (AG) y Estrategias de Evolución (EE).

ECJ es una herramienta de CE desarrollada íntegramente en Java y su versión actual es la 8,0. Esta herramienta posee una amplia variedad de herramientas para la construcción de experimentos y tiene una arquitectura ampliable por el usuario. Sin embargo, al igual que JDEAL, el paradigma de computación que utiliza no es robusto puesto que los procesos están organizados

jerárquicamente y el éxito del experimento depende de un servidor que permite continuar la evolución. Los procesos hijos obedecen a un proceso servidor del cual dependen por lo que si la ejecución del servidor se detiene los procesos hijos también lo harán. Otra característica que puede condicionar esta herramienta es el hecho de que cada proceso que forma un experimento debe evolucionar la misma especie de individuos por lo que no es posible el desarrollo de experimentos de coevolución en los que se suelen presentar varias especies de individuos.

JEO, cómo ya ha sido expuesto en otros trabajos, ofrece una nueva alternativa a todas las herramientas presentadas. El paradigma de computación que utiliza es completamente P2P por lo que los experimentos son totalmente ampliables. Cada experimento está formado por un número variable de procesos independientes entre ellos y por lo tanto robustos, es decir, la pérdida de cualquiera de los procesos no condiciona el éxito o el fracaso total de la aplicación. Está completamente desarrollada en Java 1.3.1 y es totalmente compatible con Java 1.4 por lo que ofrece una independencia de la plataforma donde se ejecuten los procesos igual a la que ofrece Java por sí mismo, es decir, los experimentos desarrollados con JEO son totalmente portables. JEO esta diseñada para albergar cualquier paradigma de CE, ya sean Algoritmos Genéticos, Programación Genética, Programación Evolutiva o Estrategias de Evolución. Por último, otra característica que hace a JEO más completa que las herramientas existentes es el hecho que permite la coevolución de especies puesto que cada proceso es totalmente independiente del resto, cada proceso es responsable de aceptar o rechazar individuos que provienen de otras poblaciones externas por lo que sería posible evolucionar a la vez diferentes especies dentro de un mismo experimento o incluso dentro del mismo proceso.

### III. JEO

#### A. Diseño

JEO está diseñada siguiendo cuatro principios básicos:

- Orientación a Objetos.
- Independencia de Plataforma.
- Flexibilidad: JEO está desarrollada apoyándose en un conjunto de interfaces Java que establecen las mínimas restricciones que cada objeto perteneciente a la librería debe cumplir. A partir de ahí, un usuario puede añadir sus propios objetos implementando dicho interfaz o puede extender cualquiera de los objetos que ya ofrece JEO y que implementan dicho interfaz. Recordar que un

interfaz java no sólo está pensado para permitir la herencia múltiple en objetos sino que además establece unas pautas de comportamiento a cualquier objeto que los utiliza.

- Abstracción de la capa inferior: JEO oculta totalmente detalles del nivel físico de la red como direcciones de red o apertura de puertos de comunicaciones. La distribución de un experimento la realiza DRM una vez que los objetos que forman dicho experimento están creados por JEO. Un experimento desarrollado con JEO que va a ser paralelizado utilizando DRM se basa en el Modelo Isla [17] [26] [36]. Cada Isla que forma parte del experimento es un agente independiente y autónomo que intercambia información con los demás agentes o Islas y que estará ejecutándose en una de las computadoras que forman parte de la red que DRM maneja.

#### B. Arquitectura e Implementación

JEO está desarrollada en utilizando *jdk 1.3.1* y es totalmente compatible con *jdk 1.4*. JEO está organizada en paquetes que agrupan interfaces y clases relacionadas con el paquete donde se alojan. Además de esta distribución lógica, cada paquete hace referencia a un tipo de objeto característico que debe formar parte de un experimento como por ejemplo el paquete `jeo.genome`. Este paquete alberga todas las clases e interfaces relacionados con el genotipo de un individuo.

Además de los objetos ya clásicos como operadores o terminadores JEO aporta varios conceptos nuevos como los descritos a continuación:

- *InfoHabitant* representa a un individuo en CE tradicional pero JEO amplía este concepto para hacer de un individuo que decide qué hacer en cada momento de la evolución. Esta característica se consigue dotando a los individuos con un objeto *Brain* que puede ser tan sencillo como un conjunto de `if` anidados o tan complejo como una red neuronal. Este objeto es totalmente opcional pero en el caso de utilizarlo, permite al usuario de JEO realizar experimentos de comportamiento social [6] [32].
- *Environment* representa un entorno donde conviven un conjunto de *InfoHabitants* que pertenecen a la misma especie. En el caso más sencillo una Isla de nuestro experimento contendrá un sólo *Environment* y en experimentos donde se quiera imitar a la realidad más fielmente cada Isla puede contener varios *Environments* que interactúan entre ellos de la misma manera que las diferentes especies interactúan en un entorno real.
- *Assessor* es el encargado de lo que tradicionalmente se denomina Evaluación y Reemplazo por

lo que cada *Assessor* evalúa primero a cada individuo de cada especie que se aloja dentro de una Isla y posteriormente, utilizando toda esa información, organiza el reemplazo mediante un sistema de recompensas. Cada individuo o conjunto de individuos es evaluado y recibirá una recompensa que actualmente consiste en *pasar a la siguiente generación* o *morir al final de generación activa*. Con este sistema se pueden implementar los diferentes paradigmas de CE como el generacional que ya utilizo Holland [20] o el estacionario [30].

- *Checkpointing* tiene dos funciones principales. La primera es decidir si una Isla debe terminar o por el contrario debe seguir evolucionando. La segunda función es el mantenimiento actualizado de todas las estadísticas que el usuario desee extraer de la Isla. La potencia de este objeto reside en el hecho de que coordinando otros objetos mantiene un conjunto de estadísticas actualizadas de cada Isla que podrán ser enviadas a algún sitio a modo informativo o simplemente ser distribuido de forma pasiva a través de la red de computadoras que están evolucionando el experimento.

#### IV. COMO DISTRIBUIR INFORMACIÓN EN JEO

Cualquier experimento creado con JEO está basado en el Modelo Isla [17] [26] [36]. Cada isla es un proceso independiente que evoluciona una o varias especies de individuos aplicando un conjunto de operadores propio y con unas políticas de evaluación y reemplazo que pueden ser características para cada Isla. Cada Isla es distribuida utilizando DRM en una red heterogénea de computadoras cuya única característica en común a priori, es que están conectadas a Internet y por lo tanto pueden formar parte de DREAM. Una vez distribuidas cada Isla es un proceso independiente y con vida propia que puede comunicarse con las demás. Esto permite realizar movimientos de migración entre las diferentes Islas enviando individuos de una Isla a otra [8]. Este tipo de comunicaciones no ofrece ningún problema puesto que normalmente se da cada cierto intervalo de tiempo entre dos Islas cualesquiera del experimento, por lo que es una comunicación escalable si se utiliza adecuadamente. Para este propósito JEO utiliza los recursos que ofrece DRM para serializar objetos incluyendo los individuos a migrar en un mensaje dirigido a la Isla de destino.

Además de los movimientos de migración, cualquier experimento distribuido necesita enviar otro tipo de información referente a qué está ocurriendo en cada proceso. Por ejemplo, quizás sería importante conocer cual es el mejor

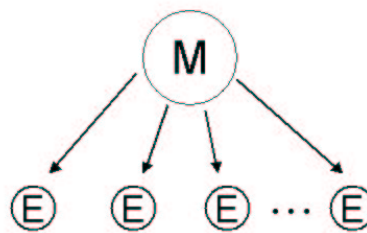


Fig. 2. Modelo de organización Maestro-Esclavo

individuo de una especie *E* determinada que aloja cada Isla o cual es el valor de evaluación medio de toda una población de individuos de una isla *I*.

En la primera versión de JEO cada Isla enviaba los datos estadísticos que el usuario especificara a una Isla principal denominada *root*. Este mecanismo limitaba en cierto modo el número de Islas que un experimento puede albergar puesto que cuando todas las islas de un experimento enviaban información a la isla *root* los procesos dejaban de ser independientes para pasar a tener una organización Maestro-Esclavo (Ver Figura 2 ). Este tipo de distribución de información se denominará *distribución activa*. Esta organización centraliza la recolección de información por lo que el sistema en cierto modo, deja de tener una arquitectura P2P y pierde robustez.

La implementación actual de JEO ofrece una solución a este problema utilizando de nuevo los recursos que DRM ofrece. Cada computadora perteneciente a DRM ejecuta al menos un proceso lo que vamos a denominar *Nodo* que está encargado de albergar los agentes que están ejecutándose en esa máquina. En [22] Jelasity et al. describe ampliamente el algoritmo que utilizado por DRM para asegurar la conectividad de los Nodos. Cada agente es independiente y autónomo al igual que las Islas creadas con JEO. Desde este punto de vista, cada Isla es también un agente y por lo tanto hereda todas sus características.

Los agentes en DRM se organizan en *Collectives*. Así, cada agente pertenece a un *Collective* único que en el ámbito de JEO significa que cada Isla pertenece a un sólo experimento identificado de forma única dentro del conjunto de *Nodos*.

Para mantener la conectividad de los Nodos, DRM incluye una pequeña base de datos incompleta de los Nodos del sistema en la que se almacena la aportación de cada Isla al sistema, que denominaremos *Contribución*.

Periódicamente cada Nodo *A* elige aleatoriamente otro Nodo *B* y ambos se intercambian la información de su pequeña base de datos si ha ha-

bido alguna modificación en ella desde él último intercambio. De este modo, la *Contribución* de cada isla va extendiéndose a través de los Nodos y por lo tanto llegará a ser conocida por todas las Islas del experimento. Denominaremos a este nuevo tipo de distribución *Distribución pasiva*.

JEO utiliza la distribución pasiva para distribuir información estadística de cada Isla al resto de Islas que forman un experimento. De este modo, aprovechando el algoritmo pasivo que utiliza DRM para mantener la conectividad, es posible distribuir información referente a qué está ocurriendo en cada Isla dentro de un experimento.

Este nuevo mecanismo de distribución no sólo lleva los resultados estadísticos a la isla *root* de cada experimento sino a todas las Islas que lo forman. El volumen de comunicaciones existentes permanece más o menos estable aumentando el tamaño de la base de datos de cada Nodo debido a la *Contribución* de cada Isla. El aumento de tamaño de la base de datos de cada Nodo provoca que el tamaño de los paquetes que se intercambian sea mayor. Para hacer frente a este inconveniente las *Contribuciones* son cadenas de texto que en ningún momento son comparables en tamaño a objetos reales paliando así el aumento de tamaño.

## V. EXPERIMENTOS

Para probar la notable disminución del volumen de comunicaciones entre las diferentes Islas que forman un experimento vamos a realizar un experimento sencillo que mide fiablemente el número de mensajes utilizados por la distribución activa y la pasiva. Para ello vamos a utilizar un problema de CE clásico como es la optimización de la función de la suma doble de Schwefel [33]. Se trata de una función continua unimodal no separable cuyo gradiente no está orientado a lo largo de sus ejes por lo que la búsqueda mínimo global se hace muy lenta.

La definición está representada en la ecuación V donde  $-65536 \leq x_i \leq 65536$  y cuyo valor mínimo se sitúa en  $X = (0, 0, 0, \dots, 0)$  para cualquier número de dimensiones. La representación para dos dimensiones se muestra en la figura 3.

$$f(x) = \sum_{i=1}^n \left( \sum_{j=1}^i x_j \right)^2 \quad (1)$$

Para el experimento se han utilizado individuos binarios que representan secuencialmente cada una de las coordenadas del vector  $X$ . Cada coordenada se ha representado por 18 bits hasta un total de 20 coordenadas. El valor fitness de cada

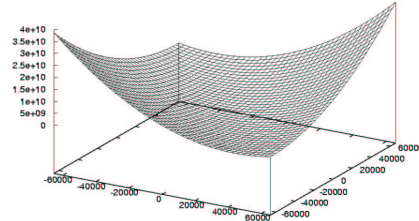


Fig. 3. Función suma doble de Schwefel

individuo es el valor de la función  $V$  para el vector de coordenadas que cada individuo representa.

El experimento construido con JEO crea un número variable de Islas que se distribuyen aleatoriamente entre un conjunto de Nodos. Cada Isla ejecuta un AG generacional elitista con una población de 100 individuos al que se le aplican tres operadores (Cruce en un punto 0,8 %, Mutación 0,1 % y Clonación 0,1 %). Cada isla emigra los dos mejores individuos con una probabilidad de 0,03. El número máximo de generaciones se ha fijado en 5000. El mecanismo de selección para la aplicación de operadores es por Torneo de dos individuos con una probabilidad de seleccionar el mejor de los dos del 0,8. La condición de terminación de cada Isla está compuesta por el número de generaciones, que nunca excederá a 5000 y por el valor del mejor individuo que será siempre mayor que 0.

Todos los resultados han sido obtenidos como media de los resultados obtenidos de 10 ejecuciones independientes. El conjunto de Nodos ha sido un número variable entre 3 y 5. Cada Nodo se ejecuta en una máquina diferente a las demás conectada a Internet.

La figura 4 muestra una ejecución media de este problema utilizando una isla y mostrando los resultados en escala logarítmica en el eje y para poder apreciar mejor el proceso de búsqueda del mínimo global de la función. En esta ejecución tipo se puede ver como el fitness decrece rápidamente al principio y es en una segunda etapa donde cuesta realmente alcanzar el mínimo global, que se alcanza en la generación 4989.

Las medidas se han realizado contando en número de mensajes que han llegado a cada Isla *root* hasta el momento en el que ésta encuentra la solución al problema. El conteo de mensajes se ha realizado teniendo en cuenta el tipo de mensaje de que se trata: mensajes estadísticos, (Stats), mensajes de migración (Migración) y el resto de mensajes (Otros) que son los utilizados por DRM.

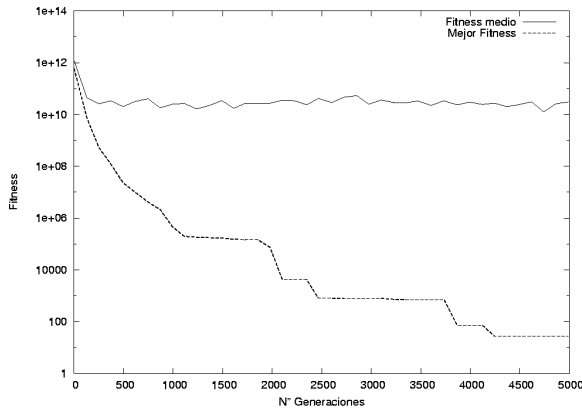


Fig. 4. Evolución tipo para una Isla.

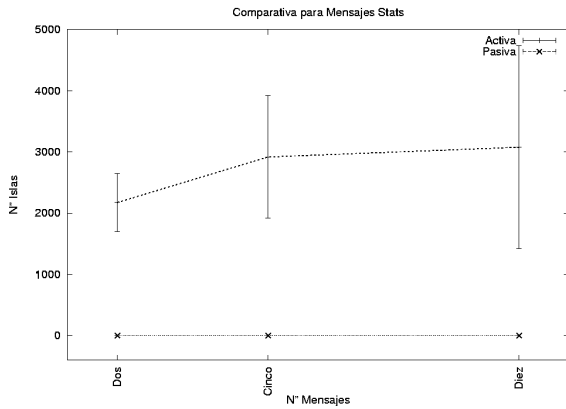


Fig. 5. Mensajes Stats para 2, 5 y 10 Islas.

La figuras 5, 6 y 7 representan respectivamente comparativas del número medio de mensajes recibidos en la isla *root* para los experimentos lanzados con dos, cinco y diez islas utilizando la distribución pasiva y activa.

Se puede apreciar que el número de mensajes *Stats* desaparece al utilizar la distribución pasiva porque la información se distribuye por otro medio. Si nos fijamos ahora en los mensajes *Otros* se aprecia que este número de mensajes aumenta para cualquier número de Islas. Esto se debe a que con la distribución activa, las actualizaciones en la base de datos de cada Nodo son más frecuentes puesto que las *Contribuciones* de cada Isla se actualizan cada generación. Esta actualización constante de la base de datos de cada Nodo hace que cuando dos Nodos comprueban si deben intercambiar sus datos la respuesta sea afirmativa en un número mayor de veces y por lo tanto se generen un número mayor de intercambios entre Nodos que provocan un número mayor de *Otros* mensajes.

Si comparamos ahora el número de mensajes de migración, se puede ver que no existe gran diferencia entre una distribución y otra ya que este

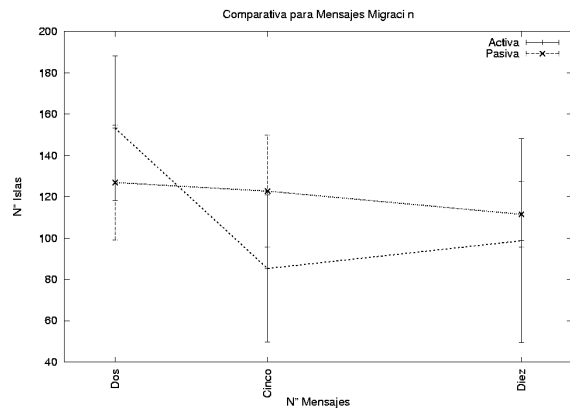


Fig. 6. Mensajes Migración para 2, 5 y 10 Islas.

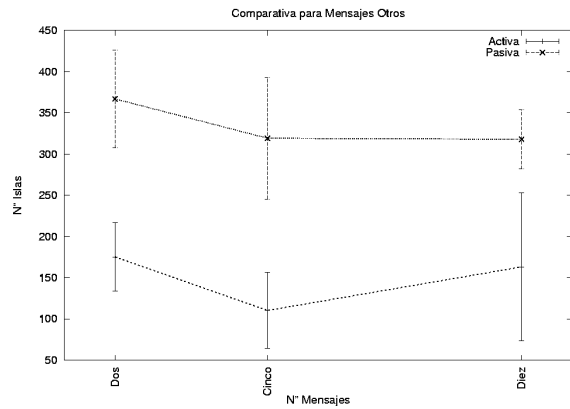


Fig. 7. Mensajes Migración para 2, 5 y 10 Islas.

valor depende del porcentaje de individuos emigrados y de la tasa de migración utilizada que se ha mantenido constante en todos los experimentos. Esto hace que a veces sea más alto para un tipo de distribución y otras veces para otra, dependiendo del factor de aleatoriedad de las migraciones. Si es sin embargo significativo el hecho de que el número de mensajes de migración sea mayor para el caso de dos islas que para el resto. Esto se debe a la política de migración utilizada. Cada isla selecciona aleatoriamente la isla de destino para los individuos a emigrar. En el caso de dos islas, las opciones de elección se reducen a una, porque no tiene ningún sentido que una isla se elija a sí misma para una migración. Conforme el número de opciones a elegir aumenta de una a cuatro y a nueve para los experimentos de cinco y diez islas respectivamente, el número de mensajes de migración recibidos por la isla *root* disminuyen. Esto no quiere decir que la frecuencia de migración haya variado sino que los mensajes de migración están más uniformemente repartidos entre todas las islas que forman el experimento.

En cualquier caso, el número de mensajes *Stats*

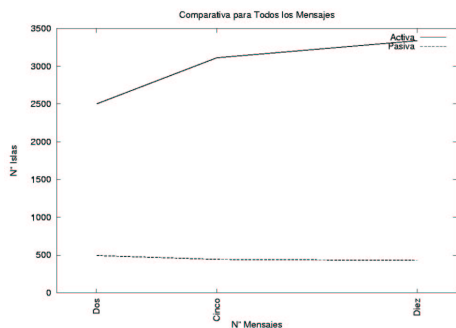


Fig. 8. Comparativa de todos los mensajes.

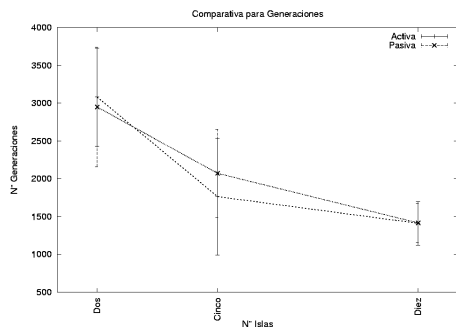


Fig. 9. Comparativa de mensajes totales.

que hemos eliminado con la distribución pasiva, es enormemente superior en número al número de mensajes que hemos podido ganar en otros apartados con la distribución activa, ya que la escala de los diferentes gráficos, 5, 6 y 7, son totalmente diferentes. Para clarificar resultados, hemos incluido la gráfica 8 donde se pueden ver las comparaciones de la suma de todos los tipos de mensajes recibidos para dos, cinco y diez Islas apreciándose cómo en todos los casos la distribución pasiva necesita menos mensajes y por lo tanto produce un volumen de comunicación menor.

Por último, anotar que los resultados obtenidos respecto al número de generaciones ( figura 9) necesarias para que la isla *root* alcance el óptimo global de la función decrece con el número de islas que componen el experimento. Esta tónica se mantiene tanto para la distribución activa como para la pasiva, lo que demuestra que el tipo de distribución utilizada no afecta a la resolución del problema para dos, cinco o diez islas sino que sólo influye en el volumen de comunicación que utilizan los diferentes procesos.

## VI. CONCLUSIONES

Este trabajo muestra como JEO utiliza un nuevo mecanismo de distribución de información aprovechando el algoritmo de conectividad de

Nodos que utiliza DRM disminuyendo enormemente el volumen de mensajes que cada conjunto de Islas enviaban entre sí para conocer información de otras Islas del mismo experimento.

El nuevo sistema, o distribución activa, distribuye toda la información de todas las Islas a todos los componentes mientras que con la distribución pasiva sólo la Isla denominada *root* conocía toda la información.

La desventaja de este método es que el tamaño de los mensajes que DRM utiliza para la conectividad de los Nodos aumenta puesto que lleva incluida la *contribución* de cada Isla. Sin embargo, esta información no es más que una cadena de texto con los resultados de las estadísticas de cada Isla por lo que ese aumento de tamaño no es en ningún momento comparable con la totalidad del mensaje original que incluye objetos completos como se describe en [21].

## VII. TRABAJO FUTURO

JEO se puede considerar una librería de CE completa por lo que en un futuro se realizarán ampliaciones para permitir la especificación de experimentos más específicos orientados a la investigación de las características emergentes en experimentos de comportamiento social.

## VIII. AGRADECIMIENTOS

This work is supported by *Distributed Resources Evolutionary Algorithm Machine* (DREAM IST-1999-12679) project. This work is funded as part of the European commission Information Society Technologies Programme (Future and Emerging Technologies). The authors have sole responsibility for this work: it does not represent the opinion of the European Community, and the European Community is not responsible for any use that may be made of the data appearing herein. Gracias a los proyectos CICYT TIC 1999-0550) e *INTAS* (INTAS-9730950) por su colaboración.

## REFERENCIAS

- [1] M. G. Arenas, P. Collet, A.E. Eiben, M. Jelasity, J.J. Merelo, B. Paechter, M. Preub, and M. Schoenauer. A framework for distributed evolutionary algorithms. In *Proceedings of the Seventh Conference on Parallel Problem Solving from Nature*, volume 2439 of *Lecture Notes in Computer Science*, pages 665–675, Granada, Spain, September 7-11 2002. Springer-Verlag.
- [2] Maribel García Arenas, Brad Dolin, Juan Julián Merelo, Pedro Angel Castillo, Ignacio Fernández De Viana, and Marc Schoenauer. JEO: Java Evolving Objects. In W. B. Langdon, E. Cantú-Paz, K. Matias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*.

- rence, page 991, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
- [3] M.G. Arenas, L. Foucart, J. J. Merelo, and P. A. Castillo. JEO: A Framework for Evolving Objects in Java. In Universidad Politécnica de Valencia, editor, *In XII Jornadas de Paralelismo*, pages 185–191, 46071 Valencia, September 2001. REPROVAL, S.L.
  - [4] M.G. Arenas, L. Foucart, M. Shoenauer, and J.J. Merelo. Computación evolutiva en Java: JEO. In Universidad de Mérida, editor, *Actas del I Congreso Español de Algoritmos Evolutivos y Bioinspirados*, pages 46–53, Merida, Badajoz, February 2002.
  - [5] Robert Baruch. Groovy Java genetic programming. <https://sourceforge.net/projects/jgprog>.
  - [6] Ana L. C. Bazzan, Joachim Wahle, and Franziska Klugl. Agents in traffic modelling - from reactive to social behaviour. In *KI - Kunstliche Intelligenz*, pages 303–306, 1999.
  - [7] Pietro Berkes and Samuele Pedroni. Jrgp. Available from <http://jrgp.sourceforge.net>.
  - [8] Erick Cantu-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms.
  - [9] J. G. Castellano, P.A. Castillo, J. J. Merelo, and G. Romero. Paralelización de evolving library usando MPI. In *XII Jornadas de Paralelismo. ISBN: 84-9705-043-6, Valencia*, pages 265–270, September 2001.
  - [10] Fuey Sian Chong. A Java based distributed approach to genetic programming on the internet. In *Proceedings of Evolutionary Computation and Parallel Processing*, 1:163–166, July 1999.
  - [11] Fuey Sian Chong. A Java based distributed approach to genetic programming on the internet. In *Proceedings of Genetic And Evolutionary Computation Conference, ISBN 1-55860-611-4, II*, July 1999.
  - [12] Pierre Collet, Marc Schoenauer, Evelyne Lutton, and Jean Louchet. EASEA : un langage de spécification pour les algorithmes évolutionnaires. Technical Report RR4218, INRIA, Domaine de Voluceau - Rocquencourt - B.P. 105 78153 Le Chesnay Cedex France, June 2001.
  - [13] Joao Costa, Nuno Lopes, and Pedro Silva. Jdeal, the Java distributed evolutionary algorithms library. Available from <http://laseeb.ist.utl.pt/sw/jdeal>.
  - [14] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, 2<sup>a</sup> edition, 1994.
  - [15] C. Gagn e and M. Parizeau. Open BEAGLE: A new c++ evolutionary computation framework. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, page Late breaking papers, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
  - [16] M. Feeley and J. S. Miller. A parallel virtual machine for efficient scheme compilation. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming, Nice*, pages 119–130, New York, NY, 1990. ACM.
  - [17] D. E. Golberg and E. Cantú-Paz. Modeling idealized bounding cases of parallel genetic algorithms. In Morgan Kaufmann, editor, *Proceedings of the Second Annual Conference of Genetic Programming*, pages 353–361, 1997.
  - [18] J. J. Merelo Guervós, M. G. Arenas, J. Carpio, P.A. Castillo, V. M. Rivas, G. Romero, and M. Schoenauer. Evolving objects. In *Proceedings JCIS 2000 (Joint Conference on Information Sciences)*, volume I, pages 1083–1086, 2000.
  - [19] J. J. Merelo Guervós, Maarten Keijzer, and Marc Schoenauer. EO evolutionary computation framework. Available from <http://eodev.sourceforge.net>.
  - [20] J. H. Holland. Properties of the bucket brigade algorithm. In *Proc. of the International Conference on Genetic Algorithms and Their Applications*, pages 1–7, Pittsburgh, PA, 1985.
  - [21] Mark Jelasity, Mike Preub, and Ben Paechter. A scalable and robust framework for distributed application. *2002 Congress on Evolutionary Computation*, May 2002.
  - [22] Mark Jelasity, Mike Preub, Maarten van Steen, and Ben Paechter. Maintaining connectivity in a scalable and robust distributed environment. In *2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002)*, May 2002.
  - [23] Dr. Kenneth De Jong. Ec lab ec++.
  - [24] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer. Evolving objects: a general purpose evolutionary computation library. In *Proceedings Evolution Artificielle 2001*, pages 231–244, 2001.
  - [25] Natalio Krasnogor and Jim Smith. MAFRA: A Java memetic algorithms framework. *Genetic And Evolutionary Computation Conference, First International Workshop On Memetic Algorithms - Workshop Proceedings. William Hart and Natalio Krasnogor and Jim Smith Editors. Las Vegas, Nevada, USA*, pages 125–130, August 2000.
  - [26] M. R. Leuze, C. B. Pettey, and J.J. Grefenstette. A parallel genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of the second International Conference on Genetic Algorithms*, pages 155–162, 1987.
  - [27] Sean Luke. A Java-based evolutionary computation and genetic programming research system. Available from <http://www.cs.umd.edu/projects/plus/ec/ecj>.
  - [28] Ben Paechter, Thomas Baech, Marc Schoenauer, Michèle Sebag, A. E. Eiben, J. J. Merelo Guervós, and T. C. Fogarty. Dream distributed resource evolutionary algorithm machine. In *Proceedings of the Congress on Evolutionary Computation 2000*, volume 2, pages 951–958, 2000. Available from <http://dr-ea-m.sourceforge.net>.
  - [29] Adil Qureshi. A Java genetic programming system. Available from <http://www.cs.ucl.ac.uk/staff/A.Qureshi/gpsys.html>.
  - [30] Khaled Rasheed and Brian D. Davison. Effect of global parallelism on the behavior of a steady state genetic algorithm for design optimization. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 534–541, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.
  - [31] Marc Schoenauer. Evolc. Available from <http://www.eark.polytechnique.fr/EvolC.html>.
  - [32] Ralph Schroeder. Social interaction in virtual environments: key issues, common themes, and a framework for research. *Springer CSCW Series*, pages 1–18, 2002.
  - [33] H.P. Schwefel. Evolution and optimum seeking, John Wiley and sons, 1995.
  - [34] SETI@home. The search for extraterrestrial intelligent.
  - [35] V. S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: practice and experience*, 2(4):315–339, December 1990.
  - [36] R. Tanese. Parallel genetic algorithms for hypercube. In J. J. Grefenstette, editor, *Proceedings of the second International Conference on Genetic Algorithms*, pages 177–184, 1987.
  - [37] M. Wall. Overview of matthew’s genetic algorithm library. Available from <http://lancet.mit.edu/galib-2.4>, 1995.