

Algoritmos evolutivos distribuidos sobre dispositivos Bluetooth

P. García-Sánchez, J.P. Sevilla, J.J. Merelo, F.A. Casado, P.A. Castillo, J.L.J. Laredo, y A.M. Mora¹

Resumen— En este trabajo se presenta un entorno basado en Java que permite el desarrollo fácil de aplicaciones de conectividad entre dispositivos Bluetooth. Actualmente es muy difícil programar aplicaciones para estos dispositivos, por lo que se hace necesario disponer de una API de alto nivel que facilite la creación de aplicaciones en las plataformas Java ME y J2SE, las más extendidas. Se muestra el desarrollo de un entorno utilizando una arquitectura en capas siguiendo un modelo cliente/servidor, basado en eventos y con comunicación asíncrona. Como ejemplo se muestra la creación de una aplicación que resuelve dos problemas de computación evolutiva conocidos (el problema del viajante de comercio y la función marea) usando un enfoque paralelo distribuido.

Palabras clave— Bluetooth, computación evolutiva paralela, algoritmos genéticos

I. INTRODUCCIÓN

ES un hecho innegable que la tecnología móvil está cada vez más presente en la sociedad. A esta presencia, cada vez más patente, se unen las nuevas formas de comunicación entre estos dispositivos, que ofrecen al usuario un gran abanico de posibilidades. Estos dispositivos tienen una potencia de cómputo comparable a los ordenadores de hace pocos años, potencia de cálculo que en general está desaprovechada y que con las herramientas adecuadas podría utilizarse para resolver problemas complejos.

Una de las tecnologías que más crecimiento está teniendo es Bluetooth [1], que permite la comunicación sin cables entre dispositivos móviles, aunque en distancias no superiores a 100 metros. El porcentaje de nuevos terminales que usan esta tecnología es cada vez mayor, y la distancia de comunicación está aumentando, pero sus posibilidades no están siendo aprovechadas en mucha medida, ya que sus usuarios se limitan a usarla en el intercambio de archivos con otros dispositivos, sin llegar a aprovechar sus capacidades “interactivas”.

La finalidad del entorno presentado en este trabajo es facilitar la creación de aplicaciones de comunicación de manera rápida y escalable. El diseño en capas facilita enormemente la estandarización y la reutilización a la hora de diseñar una aplicación [2]. Estas aplicaciones pueden ser de cualquier tipo, desde chats hasta videojuegos, pasando por el manejo de otros dispositivos a distancia (como un PC) y el uso de dispositivos móviles en programación distribuida, como el caso que vamos a presentar.

El resto del trabajo se estructura como sigue: primero se describe el estado del arte con algunos ejemplos de aplicaciones basadas en Java existentes

(sección II). La siguiente sección (III) explica las tecnologías utilizadas en el desarrollo de este trabajo. A continuación mostramos (sección V) el diseño de la arquitectura del entorno (llamada Ulfsark) y el desarrollo de una aplicación de computación distribuida y los experimentos y resultados obtenidos. Finalmente se exponen las conclusiones y las líneas de trabajo futuro.

II. ESTADO DEL ARTE

Actualmente existe una gran cantidad de aplicaciones comerciales que utilizan las posibilidades de comunicación Bluetooth en dispositivos móviles. Muchas son específicas de cada fabricante de dispositivos y utilizan lenguajes propietarios y poco accesibles. Otras aplicaciones están escritas en C++ y sólo son válidas en cierto tipo de dispositivos, como BuzzZone¹.

Además existen algunos proyectos realizados en Java, pero también son comerciales, siendo el máximo referente MobiLuck².

Simultáneamente al desarrollo de nuestro trabajo han surgido otros proyectos para comunicar dispositivos Bluetooth con código abierto basados en Java como Valhalla.³ Es el proyecto más avanzado de todos los encontrados, muy similar al mostrado en este trabajo, pero sin hacer una separación en capas, por lo que la creación de nuevas aplicaciones es más difícil, al estar orientado únicamente a la realización de un chat desde el principio de su diseño.

En lo relativo a comunicación distribuida utilizando dispositivos móviles cabe destacar el proyecto Boincoid⁴. Se ejecuta sobre dispositivos que utilizan el sistema operativo Android y permite donar el tiempo de inactividad del dispositivo para cálculo en proyectos científicos, siguiendo la filosofía del proyecto Boinc [3]. Sin embargo actualmente no se comercializan dispositivos que utilicen dicho sistema operativo.

III. TECNOLOGÍAS USADAS

A continuación describiremos en detalle los protocolos de comunicaciones y herramientas de programación utilizadas en el desarrollo de nuestro trabajo.

Bluetooth es la norma que define un estándar global de comunicación inalámbrica que posibilita la transmisión de voz y datos entre diferentes equipos mediante un enlace por radiofrecuencia [1], [4]. Los

¹Dpto. de Arquitectura y Tecnología de Computadores, Univ. Granada, e-mail: pgarcia@geneura.ugr.es

¹<http://www.buzzzone.net/eng/technologies.html>

²<http://www.mobiluck.com/>

³<http://www.valhallachat.com/>

⁴<http://boincoid.sourceforge.net/index.html>

principales objetivos que se persiguen con esta norma son facilitar las comunicaciones entre equipos móviles y fijos, eliminar cables y conectores entre estos y permitir la creación de pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales. El rango óptimo entre dos dispositivos es de 10 metros (opcionalmente 100) y la velocidad de transmisión es de 720Kb/s.

La pila de protocolos Bluetooth se compone de protocolos específicos (como L2CAP, RFCOMM o SDP[1]) y protocolos adaptados (como OBEX [5]).

La capa L2CAP es la capa núcleo de la pila por la que se transmiten todos los datos. Incluye segmentación de paquetes y reordenación de datos, así como multiplexación de protocolos. La capa RFCOMM[6], [7] proporciona emulación de múltiples puertos serie RS-232 entre dos dispositivos Bluetooth. Un dispositivo puede tener más de una sesión RFCOMM en dispositivos diferentes (una por cada dispositivo). Cada dispositivo sólo puede tener una conexión de este tipo con otro, pero sin embargo puede conectar con hasta siete dispositivos distintos a la vez.

La especificación proporciona además cuatro perfiles genéricos entre los que se incluyen el Perfil de Puerto Serie (SPP) y el Perfil de Descubrimiento de Servicios de Aplicación (SDAP) [1].

Cuando dos o más dispositivos Bluetooth dentro de un rango establecen una conexión, se forma un área de trabajo personal (Personal Area Network, PAN). Esta red puede ser *piconet* o *scatternet*: una red *piconet* tiene un solo maestro y hasta siete esclavos. El maestro es el dispositivo que inicia la conexión y el dispositivo que acepta la conexión automáticamente se convierte en esclavo. Los roles maestro/esclavo no están predefinidos. Si un octavo esclavo quiere conectarse a una red *piconet* el maestro no acepta nuevos esclavos hasta que uno de los existentes la abandone. Pero si alguno de los esclavos soporta multipunto entonces el nuevo dispositivo puede conectarse a ese esclavo creando así una *scatternet* (ver Figura 1).



Fig. 1. Configuración de varios dispositivos en una red scatternet

Hemos utilizado Java para el desarrollo de nuestra aplicación por su bajo coste de desarrollo y su independencia de la plataforma en la que se ejecutan las aplicaciones. Además es el único lenguaje que cuenta con una API Bluetooth estandarizada[8].

Las dos ediciones de Java utilizadas han sido J2SE [9], que contiene el conjunto básico de herramientas usadas para desarrollar Applets y aplicaciones de usuario final (GUIs, multimedia, redes de comunicación, etc.) y Java ME [10] (antiguamente conocida como J2ME) que es una colección de APIs para el desarrollo de software para dispositivos con pocos recursos, como PDAs, teléfonos móviles y similares.

Es necesario utilizar una interfaz estándar que permita desarrollar y probar las aplicaciones en una plataforma (como Win-32 o Linux) y desplegarla en otra (como un PDA o un teléfono móvil). JSR-82 [11] es una especificación que permite que dispositivos que utilizan Java puedan integrarse en un entorno Bluetooth. A pesar de que la especificación Bluetooth cubre muchas capas y perfiles, la API JSR-82 sólo cubre la transmisión de datos, no de voz, los protocolos L2CAP, RFCOMM y OBEX y los perfiles de puerto serie (SPP), de acceso genérico (GAP) y de intercambio de objetos (GOEP)[12]. En nuestra aplicación sólo utilizaremos el primero de ellos.

Esta especificación intenta servir las siguientes características:

- Registrar servicios
- Descubrir dispositivos y servicios
- Establecer conexiones RFCOMM, L2CAP y OBEX.
- Conducir esas actividades de forma segura.

IV. COMPUTACIÓN DISTRIBUIDA

La computación distribuida ofrece la posibilidad de aprovechar el procesamiento paralelo para conseguir una potencia de cómputo elevada a un coste mucho menor que otras arquitecturas multiprocesador al utilizar elementos hardware estándar, para los que existe un amplio mercado, con volúmenes de ventas significativos y donde es más fácil repercutir los costes de investigación y desarrollo necesarios para producir los elementos con mejores prestaciones. Dos ejemplos claros son las líneas de investigación centradas en clústers [13] y Grid [14] para procesamiento paralelo.

Esta ideas son las que hemos querido llevar a los dispositivos móviles. El avance que se está produciendo en estos aparatos en relación a sus prestaciones hace que dispongamos de grandes computadores si somos capaces de configurar los recursos de los que constan. Los elementos necesarios para la programación paralela distribuida son los dispositivos de cómputo y el entorno de comunicación serán, en este caso, los dispositivos móviles y Bluetooth respectivamente.

Por otro lado los métodos evolutivos, aunque son capaces de resolver problemas complejos, necesitan muchos recursos. En este sentido la computación distribuida es una técnica muy atractiva para mejorar

el rendimiento de algoritmos genéticos (AGs) [15]. Los algoritmos genéticos son métodos sistemáticos para la resolución de problemas de búsqueda y optimización que aplican a estos los mismos métodos de la evolución biológica: selección basada en la población, reproducción sexual y mutación. En un AG, tras parametrizar el problema en una serie de variables se codifican en un cromosoma. Todos los operadores utilizados por un algoritmo genético se aplicarán sobre estos cromosomas, o sobre poblaciones de ellos. Estos cromosomas compiten para ver cual constituye la mejor solución. El ambiente, constituido por los otros cromosomas solución ejercerá una presión selectiva sobre la población, de forma que sólo los mejor adaptados (aquellos que resuelvan mejor el problema) sobrevivan o leguen su material genético a las siguientes generaciones, igual que en la evolución de las especies. La diversidad genética se introduce mediante mutaciones y reproducción.

Desde el punto de vista de la implementación, los algoritmos evolutivos son implícitamente paralelos ya que procesan simultáneamente una población de soluciones potenciales para un problema [16]. Algunos de los procesos básicos en estos algoritmos como son la evaluación de las aptitudes de los individuos con respecto a cada uno de los objetivos que se están optimizando, la recombinación de la información genética, o la aplicación de mutaciones a algunas de las soluciones, se pueden realizar de forma simultánea si el algoritmo se ejecuta en una computadora con múltiples procesadores o bien en un clúster de ordenadores.

Se han propuesto diversas implementaciones de algoritmos evolutivos paralelos [17]:

- *Modelo de población global*: se utiliza una única población mientras que la evaluación de los individuos y la aplicación de los operadores evolutivos se hace en paralelo. Se crea una población global temporal que alternativamente se distribuye entre los procesadores y se devuelve al procesador maestro, y puede suponer un cuello de botella para la ejecución paralela. Los otros procesadores aplican los operadores de mutación cruce y evaluación de los individuos. Si se distribuyen equilibradamente los individuos se pueden obtener ganancias lineales pero, a no ser que la evaluación de las soluciones requiera un tiempo de cómputo elevado, los costes asociados a la distribución de las estructuras de datos entre los procesadores y a la comunicación y movimiento de resultados reducirán considerablemente la eficiencia de este tipo de procedimientos paralelos, que son más adecuados para computadores de memoria compartida.
- *Modelo de isla o de grano grueso*: los individuos se distribuyen entre diferentes subpoblaciones que se asignan a distintos procesadores. Estos aplican localmente los operadores evolutivos a su población e intercambian soluciones mediante migración, por lo tanto las subpoblaciones evolucionan de forma relativamente inde-

pendiente.

- *Modelo celular o de grano fino*: La población se divide en un número elevado de subpoblaciones con pocos individuos (frecuentemente un solo individuo), que se asignan cada una a un procesador. La selección y el cruce se realizan entre individuos que están en procesadores vecinos (conectados directamente entre sí).

V. DESARROLLO DE LA HERRAMIENTA ULFSARK

Tras explicar las tecnologías utilizadas en nuestro trabajo pasamos a explicar el entorno desarrollado, denominado Ulfark. Ulfark constituye un *framework* para el desarrollo de aplicaciones que utilicen tecnología Bluetooth en las comunicaciones. El principal objetivo es abstraer detalles del API especificado en el documento JSR-82, y ofrecer al programador las siguientes características:

- Interfaz sencilla
- Envío/recepción de datos asíncronos
- Flujo de datos delimitado en paquetes
- Programación orientada a eventos
- Modelo cliente-servidor

Tanto la documentación de Ulfark como el código fuente están disponibles en la dirección <http://ulfark.sourceforge.net>

A continuación describimos el formato de los paquetes, la arquitectura y la estructura en capas de la herramienta presentada.

A. Formato de los paquetes

Para facilitar la comunicación, el flujo de datos se segmenta en paquetes según el siguiente formato: los dos primeros bytes indican el tamaño de la sección de datos, el tercer byte es un código que representa información acerca del tipo de paquete y el resto de bytes del paquete constituyen los datos.

El rango de tamaño de un paquete va de 3 a 65538 bytes (incluida la cabecera), dicha delimitación está basada en el tamaño del paquete. Debido a que el protocolo L2CAP ofrece un canal libre de errores, el único error posible será la pérdida de conexión.

El segundo componente de la cabecera de un paquete, denominado *código*, permite distinguir el tipo de paquete, y permitiendo que cada aplicación establezca su propio protocolo basado en tipos concretos de paquetes. Por ejemplo, en la creación de un chat podrían usarse códigos para los paquetes de mensajes, de conexión, de mensajes privados, o en el caso de computación distribuida podría recibirse un paquete con la información referente a un individuo.

La lectura de datos se realiza de la siguiente forma:

1. Leer el paquete del flujo de entrada
2. Determinar el tipo de paquete (según el código).
3. Extraer los datos del paquete, que pueden tener una estructura interna determinada.
4. Realizar las acciones necesarias en función del tipo del paquete.

B. Arquitectura

Como hemos dicho anteriormente, uno de los objetivos de Ulfark es constituir una capa intermedia entre la API de Bluetooth y una aplicación cualquiera, así como automatizar la lectura en paralelo de los paquetes recibidos y la generación de eventos.

El modelo cliente-servidor permite que exista una entidad central creadora de servicios que administra a los clientes que se conecten a ella. Así un *cliente* puede encontrar los servicios de un *servidor* utilizando un *buscador* y conectarse a cada servicio, siendo el servidor el que se encargue de aceptar las peticiones y de distribuir los mensajes entre los clientes.

Una aplicación cliente que desea utilizar un servicio de un servidor primero tiene que encontrar el servicio que desea usar y a continuación solicitar al servidor la posibilidad de conexión. Al llegar este punto tanto el servidor como el cliente pueden enviar y recibir paquetes y actuar en consecuencia al tipo de paquete recibido.

C. Capas

El diseño del sistema se ha estructurado en tres capas. La primera, capa *ulfark*, define los paquetes de datos y un servidor y un cliente de manera abstracta. La segunda capa, *de aplicación*, es dependiente de la aplicación en sí y completa a la anterior, añadiendo las funcionalidades al servidor, al cliente y a los paquetes. Finalmente, la última capa es la capa de *presentación*, adaptada al dispositivo donde se ejecuta el programa.

En este trabajo se muestra la creación de una aplicación de cómputo distribuida usando la capa más baja como base. La Figura 2 muestra el funcionamiento de la aplicación en varios dispositivos, y los flujos de información entre las diferentes capas del sistema.

A continuación se detalla el funcionamiento de cada una de las capas diseñadas:

1. **Capa *ulfark*:** Esta capa define el servidor, el cliente y el buscador (de dispositivos y servicios dentro de cada uno). El servidor utiliza hebras distintas para controlar los clientes conectados y los servicios que oferta. El cliente utiliza una hebra de lectura en la que recibe los mensajes, y generan los eventos específicos.
2. **Capa *ulfark-evolution*:** Como ejemplo de aplicación se ha construido una aplicación de cómputo distribuida utilizando las características de la capa anterior. Así el buscador de servicios sólo encuentra dispositivos que ofrezcan servicios de computación distribuida, y el servidor sólo crea servicios específicos, además de la gestión de clientes y gestión de mensajes intercambiados.
3. **Capa de *presentación*:** Se presenta la información de las capas inferiores y depende del dispositivo que queramos usar. En nuestro caso hemos implementado el chat como aplicación

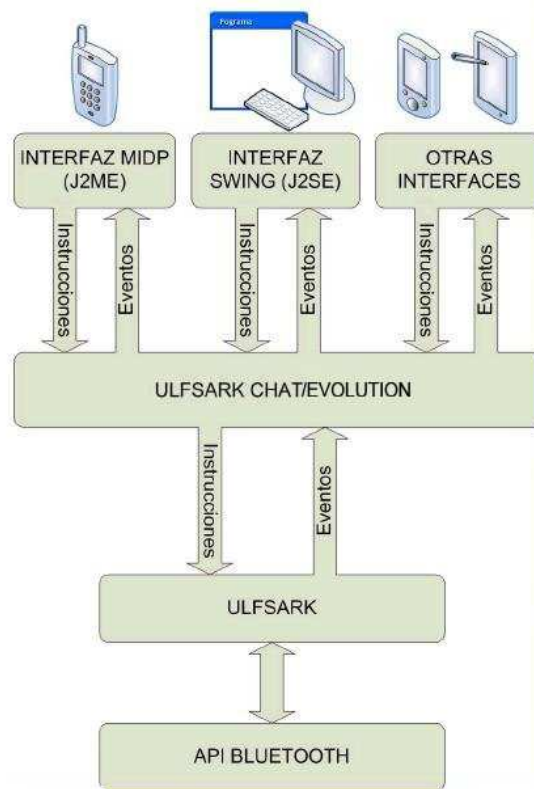


Fig. 2. Flujos de información e interacciones entre las capas de Ulfark



Fig. 3. Interfaz para dispositivos móviles tanto de un chat como de la aplicación mostrada en este trabajo utilizando Ulfark

para teléfonos móviles, usando la plataforma Java ME. Puede verse un ejemplo en la Figura 3.

D. Algoritmo Genético

Se ha decidido utilizar una arquitectura todos a todos en lugar de una versión con servidor central por la sobrecarga que ocasiona transmitir los mensajes al dispositivo encargado de gestionar al resto y el incremento del uso de memoria al tener que poner en cola los mensajes.

La forma en la que se utiliza Ulfark para computación distribuida es como sigue: cuando uno de los dispositivos móviles ejecuta la aplicación activa su servidor con un número de servicio (UUID) determinado, una vez está creado, el dispositivo se mantiene a la espera hasta que se le ordena que busque los ser-

vicios activos. Cuando esta búsqueda ha terminado crea tantos clientes como servicios ha encontrado y se conecta a todos los servidores que tengan el UUID específico. Por su parte el servidor del propio dispositivo recibe a todos los clientes que se quieran conectar guardando un registro de todos los conectados. Así se obtendrá la estructura de comunicación deseada y cada dispositivo estará a la espera de recibir los eventos que surjan a continuación.

Durante la ejecución del algoritmo cada dispositivo móvil enviará su mejor individuo cada cierto número de generaciones a otro de los dispositivos elegido aleatoriamente, siguiendo el esquema del modelo de isla comentado anteriormente. Este individuo se añadirá a la población actual del dispositivo que lo recibe en la siguiente generación. Cuando un dispositivo ha terminado la ejecución envía un mensaje de terminación al resto de dispositivos con el mejor de los individuos de su población.

VI. EXPERIMENTOS

Para probar la infraestructura creada se han desarrollado algoritmos genéticos para dos problemas conocidos:

- **Función Marea:** El problema consiste en encontrar los valores (x,y) que maximicen la función siguiente:

$$f(x,y) = \frac{\sin(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}} \quad (1)$$

Esta función es multimodal, es decir existen óptimos locales en los que un método analítico se estancaría, no alcanzando el óptimo global en la coordenada $(0,0)$ donde $f(x,y) = 1$. Los cromosomas serán en este caso un vector cuyos dos genes serán las dos coordenadas. La función cruce simplemente cruza estos dos genes y la mutación consistirá en sumar o restar aleatoriamente un número muy pequeño a una de las dos componentes.

- **Problema del viajante de comercio:** En este caso se trata de encontrar la ruta óptima, que empezando y terminando en una ciudad concreta, pase una sola vez por cada una de las ciudades y minimice la distancia recorrida por el viajante. En este caso los individuos son vectores que representan el orden de las ciudades y la función fitness a (minimizar) es la distancia recorrida por el viajante siguiendo dicho orden. La función de cruce intercambia un segmento del recorrido de las ciudades y completa el resto con las ciudades restantes. La mutación en este caso será el intercambio de dos ciudades dentro del recorrido.

El dispositivo móvil elegido es un teléfono Nokia 6288 con una velocidad del procesador virtual de Java de 15.7 Mhz y una memoria RAM de 2048 KB. Los parámetros para los dos algoritmos se muestran en la Tabla I. Cada experimento se ejecutó un total

de 15 veces y los resultados de estas ejecuciones se muestran en la Tabla II.

Puede verse que en la versión distribuida, aunque el número de individuos y el número de generaciones es la mitad que en las versiones secuenciales el número de generaciones en encontrar el mejor individuo es considerablemente inferior por utilizar el modelo de isla, en la que los espacios de búsqueda se mantienen relativamente independientes y permiten que las poblaciones no converjan a una misma solución. No obstante, en la versión distribuida la velocidad de transmisión Bluetooth hace que aumente el tiempo de ejecución del algoritmo.

VII. CONCLUSIONES Y TRABAJO FUTURO

La computación distribuida en dispositivos móviles no está siendo desarrollada tanto como sería deseable, teniendo en cuenta que los dispositivos actuales tienen la misma capacidad de procesamiento que los computadores de hace varios años. Además, el número de estos dispositivos es superior al de ordenadores personales y sus posibilidades de comunicación son más sencillas, ya que pueden crearse redes en cualquier sitio de manera directa utilizando tecnologías como Bluetooth, que aun estando muy extendida no termina de explotarse lo suficiente. Este trabajo es una base usable y sencilla para el desarrollo de aplicaciones de comunicación entre dispositivos que utilizan esta tecnología. El uso de Java hace que el número de dispositivos compatibles sea superior a cualquier otro lenguaje. Como desarrollos futuros usando la herramienta propuesta estamos estudiando las siguientes aplicaciones:

- Sistemas de votación
- Juegos multijugador
- Uso en docencia
- Chats en lugares de ocio
- Manejo de un PC desde un dispositivo móvil

Además actualmente se está trabajando en una red P2P para la creación de redes scatternet ampliables y que permitan la búsqueda de archivos en dispositivos móviles y su posterior descarga.

Se ha presentado una implementación de algoritmo genético distribuido utilizando para resolver dos problemas utilizando la herramienta Ulfark y se ha demostrado que se encuentran mejores soluciones y en menos generaciones en la versión distribuida respecto a la secuencial, sacrificando el tiempo de ejecución por la latencia producida por la comunicación Bluetooth.

AGRADECIMIENTOS

Trabajo financiado por los proyectos TIN2007-68083-C02-01, P06-TIC-02025 y OTRI-1515.

REFERENCIAS

- [1] Bluetooth SIG, "Bluetooth specification," <http://www.bluetooth.org/spec/>, 2004.
- [2] C. Larman, *Applying UML and Patterns*, Prentice-Hall, 1998.
- [3] D. P. Anderson, "Boinc: A system for public-resource computing and storage," 2004, pp. 4-10.

TABLA I

PARÁMETROS PARA LOS EXPERIMENTOS. POR CADA PROBLEMA (TSP Y MAREA) HAY DOS VERSIONES SECUENCIALES (SEC1 Y SEC2), UNA CON EL DOBLE DE GENERACIONES DE LA PRIMERA PERO CON LA POBLACIÓN DISMINUIDA EN LA MITAD. LA VERSIÓN DISTRIBUIDA (DIST) CUENTA CON LA MÍNIMA POBLACIÓN Y NUMERO DE GENERACIONES DE LAS ANTERIORES.

| Parámetro | TSPSec1 | TSPSec2 | TSPDist | MareaSec1 | MareaSec2 | MareaDist |
|--------------------------|---------|---------|---------|-----------|-----------|-----------|
| Tamaño del individuo | 10 | 10 | 10 | 2 | 2 | 2 |
| Tamaño de la población | 16 | 8 | 8 | 100 | 50 | 50 |
| Probabilidad de cruce | 70 | 70 | 70 | 70 | 70 | 70 |
| Probabilidad de mutación | 80 | 80 | 80 | 80 | 80 | 80 |
| Número de generaciones | 500 | 1000 | 500 | 200 | 400 | 200 |
| Selección torneo | 2 | 2 | 2 | 40 | 40 | 40 |

TABLA II

RESULTADOS DE LOS EXPERIMENTOS (MEDIA \pm DESVIACIÓN TÍPICA). PUEDE VERSE COMO EN LA VERSIÓN DISTRIBUIDA LA GENERACIÓN EN ENCONTRAR EL MEJOR INDIVIDUO Y SU FITNESS SON MEJORES QUE EN LAS VERSIONES SECUENCIALES, SACRIFICANDO PARA ELLO EL TIEMPO DE EJECUCIÓN, QUE AUMENTA DEBIDO A LA LATENCIA DE LA COMUNICACIÓN POR BLUETOOTH.

| Experimentos | Fitness | Generaciones | Tiempo del mejor (ms) | Tiempo total (ms) |
|--------------|-----------------------------|--------------------|------------------------|------------------------|
| TSPSec1 | 24.53 \pm 1.81 | 180.73 \pm 90.32 | 1368.33 \pm 731.44 | 3517.2 \pm 1171.07 |
| TSPSec2 | 25.2 \pm 1.66 | 163.6 \pm 148.92 | 1151.07 \pm 1074.76 | 6910.13 \pm 2576.32 |
| TSPDist | 24.4 \pm 1.55 | 77.73 \pm 57.03 | 6787.2 \pm 1656.88 | 10539.33 \pm 1559.68 |
| MareaSec1 | 0.9999997131 \pm 6.737e-7 | 162.2 \pm 31.68 | 10849.53 \pm 2649.23 | 13599.4 \pm 1395.95 |
| MareaSec2 | 0.9999998548 \pm 2.788e-7 | 336.67 \pm 57.07 | 9720.2 \pm 1571.13 | 11544.73 \pm 411.52 |
| MareaDist | 0.9999996953 \pm 4.491e-7 | 89.13 \pm 40.94 | 14479.53 \pm 3172.67 | 19986 \pm 3138.85 |

- [4] P. Wei, C. Chen, C. Chen, and J. Lee, "Support and optimization of Java RMI over a Bluetooth environment," *Concurrency Computation Practice and Experience*, vol. 17, no. 7-8, pp. 967-989, 2005.
- [5] IrDA, "OBEX specification," <http://irda.org/displaycommon.cfm?an=1\%subarticlenbr=7>.
- [6] Q. Wang, Z. Zhong, and Q. Gao, "Validation and implementation of Bluetooth RFCOMM protocol," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 4586, pp. 475-483, 2001.
- [7] D. Ye, Z. Jiang, H. Lin, and Q. Gao, "Testing of Bluetooth protocol stack using emulator," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 4586, pp. 256-265, 2001.
- [8] B. Hopkins and R. Antony, *Bluetooth For Java*, Apress!, 2004.
- [9] Sun Microsystems, "Java SE specification," <http://java.sun.com/javase/reference/index.js>.
- [10] Sun Microsystems, "Java ME specification," <http://java.sun.com/javame/reference/apis.jsp>.
- [11] Sun Microsystems, "JSR-82 specification," <http://jcp.org/en/jsr/detail?id=82/>, 2006.
- [12] P. Tremblett, "Java and Bluetooth," *Dr. Dobb's Journal*, vol. 30, no. 7, pp. 41-44, 2005.
- [13] R. Buyya, *High Performance Cluster Computing: Architectures and Systems*, Prentice-Hall, 1999.
- [14] I. Foster, "The Grid: A new infrastructure for 21st Century Science," *Physics Today*, vol. 55, pp. 42-47, 2002.
- [15] I. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 443-462, 2002.
- [16] J.J. Merelo, J.G. Castellano, P.A. Castillo, and G. Romero, "Algoritmos genéticos distribuidos usando SOAP," *Actas de las XII Jornadas de Paralelismo*, pp. 99-103, 2001.
- [17] L.D. Chambers, *Practical Handbook of Genetic Algorithms: Complex Coding Systems*, CRC-Press, 1998.
- [18] T. Song, S. Lu, K. Shen, X. Song, and Z. Ye, "The study and application of key techniques on short distance wireless networks," *Proceedings of the International Symposium on Consumer Electronics, ISCE*, pp. 428-431, 2005.