Implementación de algoritmos meméticos con capacidad de auto-generación sobre CouchBD

Manuel García García[a], Mariela Nogueira[a], Carlos Cotta[a], Antonio J. Fernández-Leiva[a]
Juan J. Merelo Guervós[b]

Resumen— **Los algoritmos meméticos constituyen un paradigma de optimización basado en la explotación sistemática del conocimiento acerca del problema que se desea resolver y de la combinación de ideas tomadas de diferentes metaheurísticas, tanto basadas en población como basadas en búsqueda local. Como la mayoría de los algoritmos evolutivos, los meméticos también han sido usados para resolver problemas de optimización en el campo de la Inteligencia Artificial, gracias a su capacidad de explorar espacios de búsqueda complejos en tiempos razonables. En este artículo se presenta una propuesta de implementación de algoritmos multimeméticos (esto es, algoritmos meméticos con capacidad de auto-generar las estrategias de búsqueda local) que emplea el sistema de Base de Datos CouchDB para manejar poblaciones persistentes y se hace un análisis del rendimiento que muestran estos algoritmos al resolver algunos problemas de optimización.**

Palabras clave— **Algoritmos evolutivos, Algoritmos meméticos, Persistencia**

I. Introducción

Los algoritmos evolutivos son procedimientos de optimización y búsqueda de soluciones basados en la evolución biológica, donde las posibles soluciones a un problema dado son consideradas como individuos de una población que irá evolucionando para hallar mejores soluciones a dicho problema. Estos algoritmos se consideran una rama de la inteligencia artificial usada principalmente en problemas con espacios de búsqueda extensos y no lineales, en donde otros métodos de búsqueda no son capaces de encontrar soluciones en un tiempo razonable.

Un ejemplo de algoritmos evolutivos son los algoritmos meméticos (MAs) [1] – véase [2], [3] para un tratamiento reciente y una revisión bibliográfica de estas técnicas. Concebidos originalmente como una estrategia de búsqueda en la que una colección de agentes alternaban periodos de cooperación y competición con fases de mejora local, los MAs pronto encontraron una forma natural de plasmarse como la hibridación de técnicas de búsqueda basadas en poblaciones con técnicas de búsqueda basadas en trayectorias, orquestando de esta manera la combinación sinérgica de las capacidades de búsqueda global y local de las técnicas combinadas. Uno de los conceptos más comúnmente mencionados en conexión con los MAs es el de evolución Lamarckiana [4] a través de la cual tanto la genética heredada como los rasgos adquiridos son transmitidos a la futura descendencia de la población. El empleo de memes [5] también ha sugerido en numerosas ocasiones una conexión con la evolución cultural.

En este artículo se hará uso de SofEA[1] [6], [7] el cual es un marco de trabajo que permite a un programador la implementación de algoritmos genéticos con poblaciones persistentes mediante el manejo de instancias en CouchDB [8]. Dichas poblaciones persistentes son útiles tanto desde el punto de vista del análisis forense de las trazas de ejecución como para abordar estrategias no convencionales de recuperación de soluciones anteriores durante el funcionamiento del algoritmo. Al ser una base de datos NoSQL, CouchDB "rompe" con las bases de datos tradicionales, las cuales poseen esquemas y están diseñadas para almacenar datos de forma estructurada. En CouchDB la información no se encuentra estructurada de la misma manera que en las bases de datos tradicionales, sino que se define usando la Notación de Objetos de JavaScript (JSON, por sus siglas en inglés). Un documento JSON [9] es una colección de pares clave:valor separados por comas (siendo los dos puntos el carácter separador de la clave y el valor). Las claves han de ser cadenas (y no repetirse), mientras que los valores pueden ser: números flotantes, cadenas Unicode, valores *booleanos*, valores nulos, y listas.

El objetivo principal de este proyecto es desarrollar el proyecto SofMA[2], que será una adaptación y ampliación de SofEA que permitirá implementar algoritmos meméticos con auto-generación usando CouchDB para garantizar la persistencia de las poblaciones. Otro objetivo es realizar un estudio del rendimiento y la escalabilidad del nuevo sistema, analizando en qué medida afectan los distintos parámetros que intervienen en un algoritmo memético. En función de estos objetivos, este artículo ha sido organizado como sigue. En la siguiente sección se muestra una breve descripción de SofEA, luego se presenta la propuesta de SofMA enfatizando en las mejoras que incorpora con respecto a SofEA, y finalmente se muestran los resultados obtenidos en varios experimentos que ayudaron a evaluar el rendimiento de la propuesta.

[a]Dpto. Lenguajes y Ciencias de la Computación (Universidad de Málaga) y [b] Dpto. Arquitectura y Tecnología de Computadores
E-mails: {mnogueira,ccottap,afdez}@lcc.uma.es, jmerelo@geneura.ugr.es

[1] `https://launchpad.net/sofea/`
[2] `https://github.com/ccottap/SofMA`

II. SofEA

SofEA es un cliente que permite la implementaión de algoritmos genéticos usando CouchDB como soporte para la persistencia de la población, pues los individuos no se eliminan cuando son reemplazados por la selección natural, sino que pasan a un estado especial y son almacenados en la base de datos. Las vistas hechas para CouchDB han sido desarrolladas en JavaScript pero el grueso del código de SofEA está hecho en Perl. Está compuesto básicamente por cuatro módulos que se conectan a una instancia de CouchDB para actuar sobre la población.

Cada uno de estos módulos se encarga de realizar una fase de las que componen los algoritmos genéticos:

- El módulo de inicialización crea la población inicial. En dicha población, los individuos o cromosomas son documentos JSON.
- El módulo de evaluación toma un número determinado de individuos de la población y calcula su *fitness*.
- El módulo de reproducción se encarga de seleccionar un número determinado de individuos y producir la nueva generación. Aquí se realiza el cruce y la mutación de individuos.
- El módulo de reemplazo se encarga de tomar un número determinado de individuos (los peores, según el fitness) y de "eliminarlos". Dado que uno de los objetivos es la persistencia de la población, los individuos nunca se llegan a eliminar de la base de datos, simplemente pasan a un estado especial para no ser usados en el futuro de la ejecución del algoritmo.

Cada uno de estos módulos toma una serie de parámetros de un archivo de configuración general que define el experimento que se va a realizar. Esta modularidad será útil más adelante para estudiar el comportamiento del sistema al modificar el número de estos clientes.

III. SofMA

SofMA se divide en dos componentes principales: SofMA-M que permite la implementación de algoritmos multimeméticos [10], y SofMA-C para desarrollar algoritmos meméticos coevolutivos [11]. En este trabajo nos concentraremos esencialmente en el primero de los modelos, esto es, algoritmos multimeméticos.

En un algoritmo memético, además de los individuos, hay un componente muy importante: los memes [12]. Mientras que los individuos o cromosomas representan soluciones a un problema binario dado, los memes codifican un procedimiento de aprendizaje individual (o búsqueda local). Los memes empleados en SofMA para realizar esta fase de aprendizaje individual son reglas. Dichas reglas están compuestas por un antecedente y un consecuente del mismo tamaño y son aplicadas a individuos. Por ejemplo para el caso de un individuo codificado como una cadena binaria, se podría tener por ejemplo la siguiente regla:

$$1001 \rightarrow 1111$$

Para el individuo 01001101 se podría aplicar la regla a partir del segundo bit, dando como resultado la cadena 01111101. Además del uno (1) y del cero (0), las reglas pueden contener un carácter especial: el punto (.), el cual es usado de comodín en en el antecedente, un punto significa que dicho bit de la cadena casará tanto con un uno como con un cero.

La arquitectura de SofMA la componen seis módulos (o clientes) que se conectan a una instancia de CouchDB para actuar sobre la población. El esquema de esta arquitectura se muestra en la Figura 1.

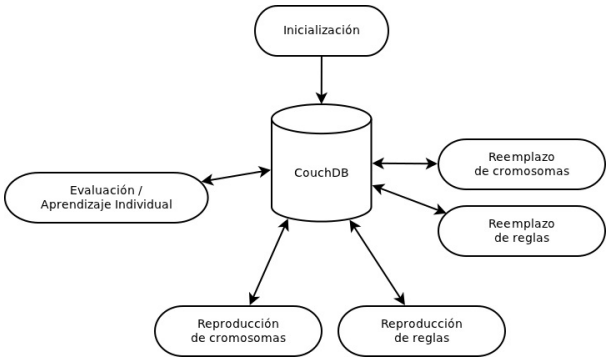

Fig. 1. Módulos que componen SofMA

En el caso de los algoritmos multimeméticos, los módulos para reproducción y reemplazo de reglas no son aplicables, ya que las reglas (memes) están empotradas en los individuos de la población. Así, en SofMA-M los individuos están compuestos por una parte genética y una parte memética. La parte genética representa una solución al problema, mientras que la parte memética codifica un meme o procedimiento de búsqueda local. Dicho meme (que acompaña siempre al individuo) es usado en la fase de aprendizaje individual para mejorar al propio individuo.

Los atributos de un individuo en SofMA-M son en su mayoría los mismos que se usan en SofEA y se explican a continuación (véase en la Figura 2 el ejemplo de un individuo codificado):

- _id: identificador único de un documento en CouchDB.
- _rev: versión del documento.
- rnd: número aleatorio de 0 a 1 que se le da a cada individuo al ser creado. Este número sirve para ordenar la población a la hora de tomar individuos para ser evaluados. Un número aleatorio (entre 0 y 1) es generado en cada ciclo del cliente de evaluación, el cual toma aquellos documentos cuyo atributo rnd sea mayor que el número generado.
- type: indica el tipo de individuo y como en SofMA-M solo hay cromosomas, este atributo vale siempre 0.
- state: indica el estado del individuo, 0 representa a un individuo vivo, mientras que 1 representa a un

```
{
    "_id": "00000000001010000000000000000000 0.11 —> .000 (4)",
    "_rev": "2-08bf53fa23996a8f134979a066825452",
    "rnd": 0.697582016535762,
    "type": 0,
    "state": 0,
    "gen": 172,
    "str": "00000000000000000000000000000000",
    "fitness": 32
    "rule": {
            "ant": "0.11",
            "con": ".000",
            "len": 4
    },
}
```


Fig. 2. Individuo codificado para SofMA-M

individuo “eliminado”.

- gen: representa la generación en la que se creó un individuo.
- str: cadena binaria que representa a un cromosoma como la posible solución al problema a resolver.
- fitness: el fitness de la cadena binaria.
- rule: contiene la regla asociada a un cromosoma, la cual posee los siguientes atributos: ’ant’ (antecedente de la regla), ’con’ (consecuente de la regla) y ’len’ (longitud de la regla).

Tal como se apuntaba anteriormente, SofMA-M está compuesto por cuatro módulos o clientes que se conectan a una instancia de CouchDB para actuar sobre la población, similar a como ocurre en SofEA. Cada uno de estos clientes se encarga de realizar una fase de las que componen los algoritmos. La principal diferencia con respecto a SofEA está en el “Módulo de evaluación” que ahora se convierte en “Módulo de evaluación y aprendizaje”. Este módulo funciona tomando un determinado número de individuos de la población que aún no tienen fitness y procede a calculárselos. De este conjunto de individuos (compuestos por cromosoma y regla), algunos serán sometidos a aprendizaje individual con la intención de mejorar su fitness.

El aprendizaje individual se realiza con un algoritmo de ascensión de colinas. Este algoritmo para un cromosoma dado, genera una lista de cromosomas vecinos y se queda con el mejor (en términos de fitness); ese mejor vecino será el nuevo cromosoma. Este proceso se repite un número determinado de veces, dando lugar a un camino ascendente en el paisaje de búsqueda. Tanto el nṁuero de vecinos generados en cada iteración como el nṁuero de iteraciones de búsqueda local son parametrizables para controlar el coste computacional de la fase de aprendizaje.

Para llevar a cabo la reproducción, el módulo homónimo hace una petición a la base de datos, pidiendo un determinado número de individuos (lo fijado previamente para la reproducción en el fichero de configuración) al azar. A partir de estos individuos se genera la nueva descendencia. Los individuos de la nueva generación son insertados en la base de datos. La producción de la nueva descendencia se realiza tomando individuos dos a dos de los seleccionados para la reproducción y cruzando sus cromosomas y sus reglas. Después de esto, los cromosomas y reglas sufren (o no) mutación. Existen (para los cromosomas y para las reglas) parámetros que determinan la probabilidad de cruce y mutación. Estos parámetros pueden ser modificados a priori desde la configuración del sistema. Para realizar el cruce en SofEA se seleccionan dos puntos aleatorios de la cadena a partir de los cuales esta se divide y se intercambian los genes. En cuanto a las reglas, su cruce siempre se hará intercambiando antecedente con consecuente. La mutación se realiza bit a bit y la probabilidad de que un bit mute viene determinada por el parámetro que se extrae de la configuración de SofMA. Después de la reproducción, el módulo de reemplazo se encarga de tomar una cantidad determinada de los peores individuos (aquellos con menor fitness) y los elimina. La cantidad de individuos que se toman puede ser configurada a priori mediante el fichero de configuración.

IV. Optimizaciones y mejoras con respecto a SofEA

A continuación se resumen las mejoras más significativas que se le han incluido a SofMA. Algunas de ellas eran necesarias debido a la mayor complejidad de SofMA respecto a SofEA y otras son mejoras que se hicieron sobre el proyecto original de SofEA.

En SofEA la cadena binaria a utilizar para resolver un problema venía determinada por su longitud. En SofMA, la cadena binaria a utilizar viene dada por dos parámetros, la cantidad de subcadenas que contiene y el tamaño de las subcadenas. Lo cual es muy conveniente para problemas en los que la propia definición demanda la división de la cadena en subcadenas.

Otro elemento es que en SofEA solo se realizaba la evaluación una vez por individuo. En SofMA sin embargo, dado que se realiza aprendizaje individual, los individuos son evaluados más de una vez. Cuando a un cromosoma se le aplica una regla, cambia parte del mismo, haciendo que sea necesario recalcular el fitness del individuo. Para una cadena de bits (presumiblemente de gran tamaño) realizar la evaluación del total de la cadena es costoso. Por ello, y gracias a lo comentado en el anterior punto, en SofMA se evalúan los individuos por partes, es decir, a cada subcadena del cromosoma se le aplica la función de fitness que se ha definido para el problema. Entonces cuando se calcula el fitness de un individuo, se obtiene (además del fitness total de la cadena) una lista con los sub-fitnesses de cada subcadena. De esta forma, al aplicar una regla al individuo solo habrá que recalcular los sub-fitnesses de las subcadenas afectadas por el cambio, ya que el resto de sub-fitnesses permanecerá igual.

Tanto en la configuración de SofEA como de SofMA, existen parámetros para determinar el tamaño de los paquetes de individuos a utilizar por los distintos módulos. Para hacer que estos paquetes tomados por los módulos no contengan ningún tipo de ordenación previa de los individuos, se utiliza su atributo rnd (número aleatorio de 0 a 1). En SofEA, si el número aleatorio generado es demasiado grande (cercano a 1), puede darse que el número de cromosomas que se obtengan (cuyo atributo rnd sea mayor que dicho número) sea menor que el número de cromosomas que se querían obtener en un principio. Además, existen individuos con un número aleatorio generado muy bajo (cercano a 0) y que por tanto tienen poca probabilidad de ser evaluados, y del mismo modo, aquellos individuos con un número aleatorio alto tendrán una mayor probabilidad de ser evaluados, especialmente cuando haya pocos individuos en la población. Estos problemas se han solucionado en SofMA haciendo que la lista de individuos a tomar sea "circular" y así en caso de que el número aleatorio generado sea tan alto que haga que se tomen menos individuos de los que se debiera, se tomará el número de individuos restantes del comienzo de la lista, que serían aquellos individuos con números aleatorios bajos.

V. Experimentos y resultados

En esta sección se exponen los diferentes experimentos que se han realizado sobre las propuestas implementadas y los resultados obtenidos. En el entorno experimental se ejecutaron tanto los clientes de SofMA como la base de datos CouchDB sobre un ordenador que tiene un procesador Intel Core 2 Duo P7350 con 3 Gb de memoria y se usó el sistema operativo Ubuntu 11.04. Para realizar un serio análisis de los resultados se empleará el Test de Wilcoxon (que al ser no-paramétrico no precisa considerar la distribución normal de los datos) con la intención de valorar la significancia estadísticas de las diferencias entre los conjuntos de datos recogidos.

La experimentación con SofMA-M estará dividida en dos fases. En ambas se tratará de hallar la configuración que aporte mayor rendimiento, primero variando el tamaño del paquete de individuos que toman los distintos módulos del sistema y en la segunda se variará el número de clientes de aprendizaje local y de reproducción. En todos los casos el tamaño de la población que evoluciona es de 128 individuos.

A. Experimentos para determinar la configuración idónea de paquetes

Para estos experimentos fueron usados los problemas Trampa y el Masivamente Multimodal (MMDP) los cuales son especialmente difíciles de afrontar para los EAs [13]. Estas funciones se definen en el apéndice.

La nomenclatura definida para denominar las diferentes configuraciones de los paquetes es: $lx - ry$, donde el parámetro 'x' representa el número de individuos (o tamaño del paquete) que toma el módulo de evaluación/aprendizaje individual (denotado por la letra 'l'), es decir, los individuos que pasarán a la fase de aprendizaje; e y indica la cantidad de individuos que irán a la fase de reproducción (para denotar al módulo de producción se usa la letra 'r'). Para estos parámetros existen tres posible valores: $128, 64$ y 32, lo que da lugar a nueve combinaciones del par $lx - ry$. A continuación se muestran los resultados obtenidos del análisis del número de evaluaciones y los tiempos de ejecución de los algoritmos para cada combinación $lx - yr$ en un total de 20 ejecuciones. En todas las figuras la posición de las cajas en el eje x está ordenada de menor a mayor respecto al valor de las medianas.

La Figura 3 muestra mediante diagramas de cajas el número de evaluaciones que necesitaron las diferentes configuraciones para resolver la función Trampa y la MMDP. Al contrastar estos resultados con el test de Wilcoxon se comprobó que para la función Trampa la configuración $l128-r128$ tiene diferencias estadísticamente significativas con la mayoría de las otras distribuciones, de hecho en los diagramas de cajas se ve que esta configuración es la peor de todas. Para el caso de la función MMDP no se detectaron diferencias estadísticamente significativas entres los conjuntos aunque sí hay que reconocer que existen ciertas similitudes con la función Trampa, pues aquí también la configuraciones $l128 - r128$ y $l64 - r128$ fueron las que más se distinguieron del

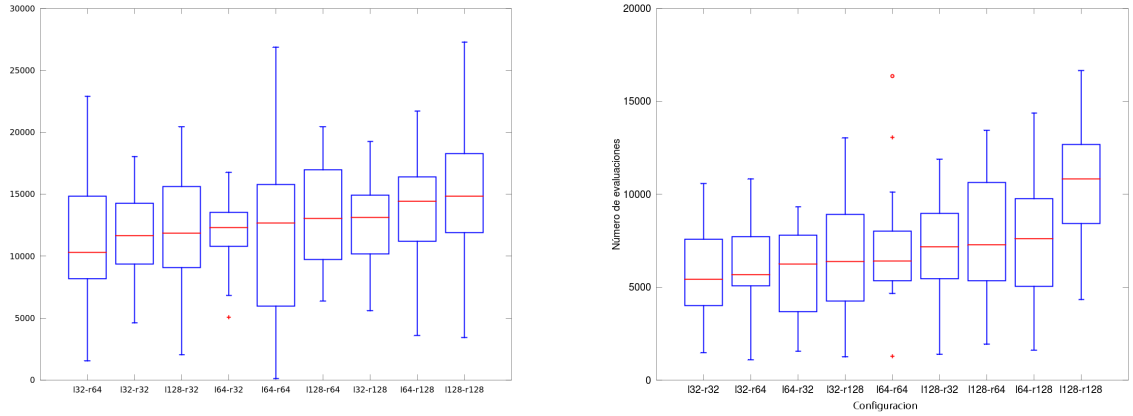

Fig. 3. (Izquierda) MMDP: cantidad de evaluaciones necesarias. (Derecha) Trampa: cantidad de evaluaciones necesarias

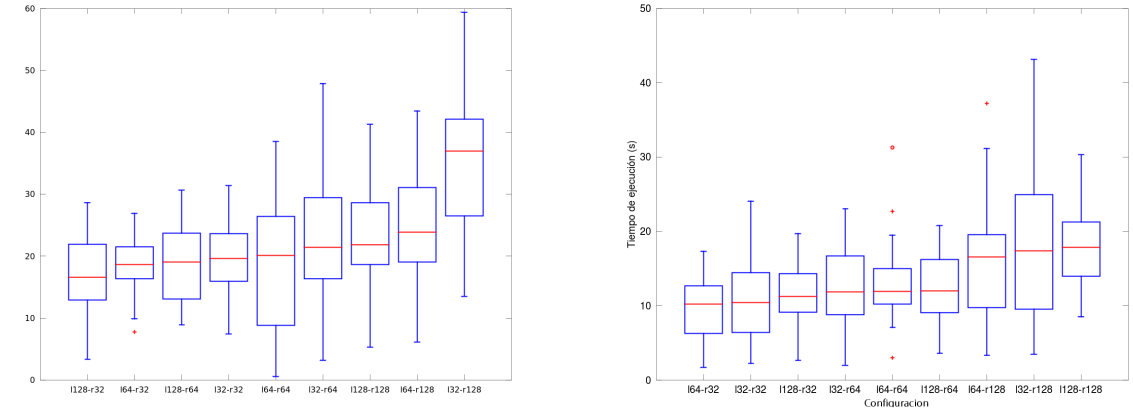

Fig. 4. (Izquierda) MMDP: tiempos de ejecución. (Derecha) Trampa: tiempos de ejecución

resto. La Figura 4 muestra el comportamiento del tiempo de ejecución para las mismas instancias de los problemas. Se puede notar que tanto en el diagrama de cajas para la función Trampa como en el de la función MMDP aquellos experimentos en los que el tamaño de paquete de aprendizaje individual l es mayor que el paquete de reproducción r requieren menos tiempo para encontrar la solución, de ahí que las tres mejores configuraciones para MMDP son $l128-r32$, $l64-r32$ y $l128-r64$, y dos de ellas ($l128-r64$ y $l64-r32$) también se encuentran entre las tres mejores configuraciones para la función Trampa.

Según Wilcoxon, para la función Trampa, la configuración $l128-r128$ es significativamente diferente (p-valor = 0,05, comparativas individuales) al resto de las configuraciones, exceptuando a $l64-r128$ y $l32-r128$. Otra de las diferencias significativas la aporta $l32-r128$ con el mayor tiempo de ejecución, y $l64-r32$ que resultó ser la mejor configuración de todas. Y en el caso de la función MMDP las configuraciones que mostraron diferencias estadísticamente significativas fueron $l32-r128$, que según el diagrama de cajas del tiempo de ejecución de la función MMDP es la peor de las configuraciones, y la segunda peor que es $l64-r128$ también se diferencia del resto de las configuraciones. Atendiendo a todo el análisis que se realizó variando el tamaño de paquetes de evaluación/aprendizaje individual y de reproducción, se comprobó que lo óptimo es que el paquete de aprendizaje sea mayor que el paquete de reproducción. Las configuraciones que se han observado que tienen los mejores tiempos de ejecución son $l64-r32$ y $l128-r32$.

B. Estudio de la configuración idónea de clientes

En esta etapa de experimentación se repetirán los mismos experimentos de la fase anterior pero esta vez el objetivo es determinar cuál es la configuración idónea del número de clientes para el aprendizaje individual y la reproducción de individuos. Estas configuraciones de clientes serán expresadas como un par $CLx-CRy$, donde x es el número de clientes de aprendizaje individual denotado por CL e y es el número de clientes de reproducción representado como CR. Se probarán los mismos problemas usados en la primera serie de experimentos, la función Trampa con 16 subcadenas de tamaño 4, y

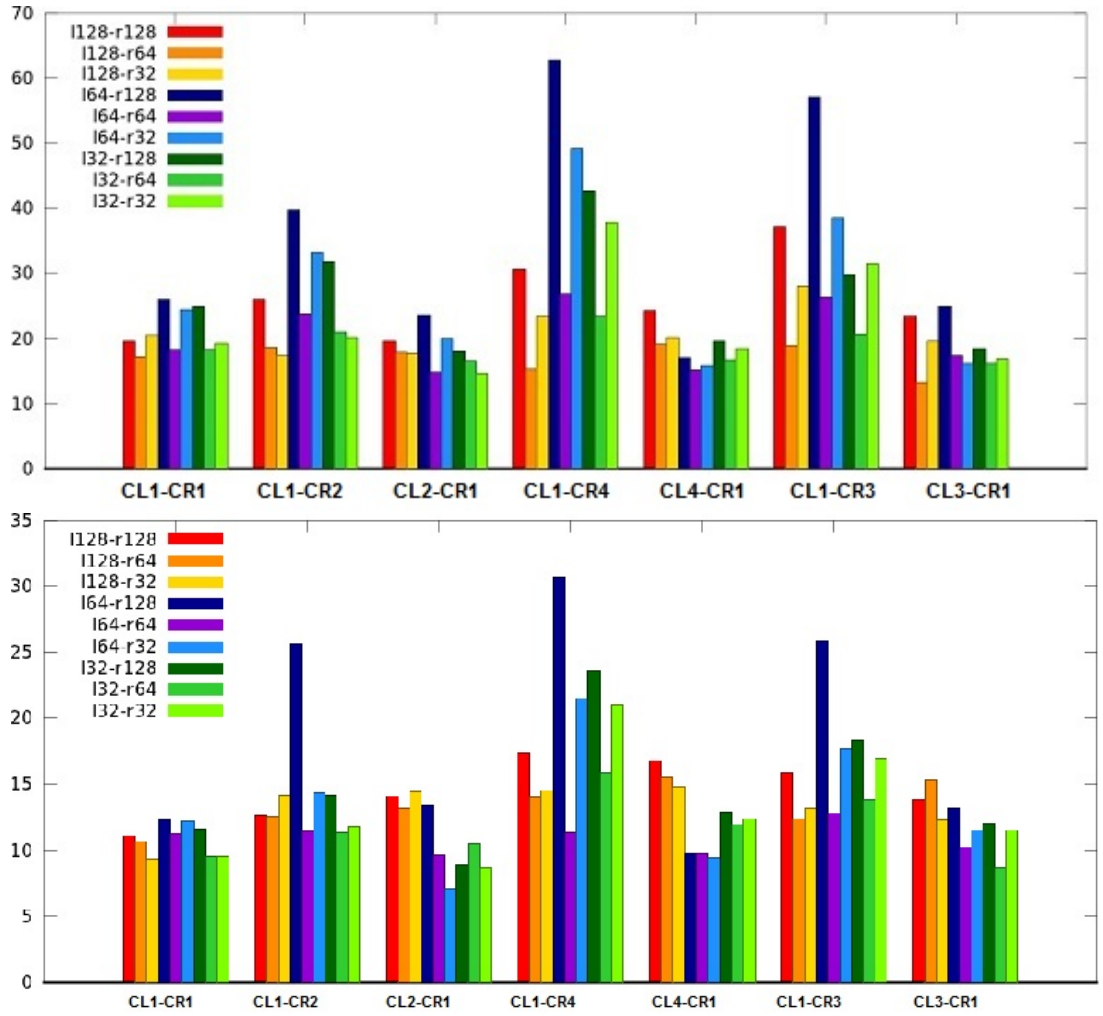

Fig. 5. (Arriba) Tiempos medianos de ejecución para MMDP. (Abajo) Tiempos medianos de ejecución para Trampa

MMDP con 10 subcadenas de tamaño 6; se experimentará con las mismas nueve configuraciones de paquetes anteriormente mostradas; y para el caso de las configuraciones de clientes, se probarán las que se muestran en la Tabla 1.

TABLA I
COMBINACIONES DE $CL-CR$ USADAS EN LOS EXPERIMENTOS

$CL1-CR1$	
$CL2-CR1$	$CL1-CR2$
$CL3-CR1$	$CL1-CR3$
$CL4-CR1$	$CL1-CR4$

En la Figura 5 se muestran gráficas de barras con los tiempos medianos de ejecución en función de la configuración de paquetes y de clientes, para analizar cuál es la configuración que ofrece los mejores resultados globales. Analizando los tiempos de ejecución, se puede ver que en ambos problemas ocurre lo siguiente:

- Si el número de clientes de aprendizaje individual es mayor que el número de clientes de reproducción, se consigue mejor rendimiento (para la función trampa el tiempo está alrededor de los 10 segundos mientras que para la función MMDP está entre 10 y 20 segundos. La configuración que se comporta mejor, en general, es $CL2-CR1$.
- Si el número de clientes de reproducción es mayor que el número de clientes de aprendizaje individual, se requiere más tiempo para encontrar la solución (llegando algunas a los 30 segundos para la función Trampa y a 60 segundos para MMDP). Por tanto, las mejores configuraciones de clientes son aquellas que tienen más clientes de aprendizaje individual que de reproducción, destacando para este experimento las configuraciones $CL2-CR1$.

VI. CONCLUSIONES

En este proyecto se ha realizado la implementación del proyecto SofMA para la creación de algoritmos meméticos con poblaciones persistentes en CouchDB. Este tipo de algoritmos son de gran utilidad para la resolución de problemas optimización (como algunos de los fueron empleados en los experimentos) que resultan muy costosos de resolver mediante algoritmos genéticos. Desde que se comenzó a extender el proyecto SofEA y durante todo el desa-

rrollo de SofMA, se han ampliado los parámetros existentes en la configuración del sistema, con el objetivo de hacer que SofMA sea lo más flexible posible y permita variar los escenarios experimentales sin necesidad de modificar el código. Por ejemplo, si quisiésemos que SofMA funcionase como algoritmo genético, bastaría con poner la probabilidad de aprendizaje individual a 0 (nunca se entraría en la fase de aprendizaje individual y simplemente se evaluarían los individuos). Obviamente, para algoritmos genéticos, los módulos de reproducción y de reemplazo de reglas no habría que ejecutarlos, ya que lo único que provocarían sería sobrecarga en la CPU y retrasaría el trabajo de los otros módulos. Esto es posible debido a que SofMA fue desarrollado respetando la modularidad original de SofEA.

Respecto a los resultados obtenidos, se ha observado que para ambas versiones del proyecto es conveniente emplear un mayor número de clientes de evaluación que de reproducción. Si se tienen demasiados clientes de reproducción, estos producirían más cromosomas de los que los clientes de aprendizaje pueden manejar. Por esto, lo más idóneo es que haya más clientes de aprendizaje que de reproducción. Y relacionado con el número de clientes se identificó la importancia que tiene el tamaño de los paquetes de aprendizaje individual y de reproducción para lograr un rendimiento adecuado, comprobándose que un tamaño de paquete de reproducción mayor que el de aprendizaje empeora el rendimiento de SofMA.

Las perspectivas de trabajo futuro en esta misma línea de investigación, están enfocadas a implementar mejoras sobre SofMA para aumentar su flexibilidad en la solución de diversos problemas de optimización, una de estas mejoras podría ser permitir la definición y resolución de problemas no binarios, es decir, problemas que no se puedan representar con una cadena binaria, como por ejemplo el Problema del Viajante de Comercio y otros aún más complejos

Agradecimientos

Este trabajo está parcialmente financiado por la Junta de Andalucía dentro del proyecto P10-TIC-6083 (DNEMESIS[3]), por el MICINN dentro del proyecto TIN2011-28627-C04 (ANYSELF[4]), y por la Universidad de Málaga. Campus de Excelencia Internacional Andalucía Tech.

Referencias

[1] Pablo Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms," Tech. Rep. Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

[2] Ferrante Neri and Carlos Cotta, "Memetic algorithms and memetic computing optimization: A literature review," *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, 2012.

[3] Ferrante Neri, Carlos Cotta, and Pablo Moscato, *Handbook of Memetic Algorithms*, vol. 379 of *Studies in Computational Intelligence*, Springer-Verlag, Berlin Heidelberg, 2012.

[4] Andrés Galera, "Lamarck y la conservación adaptativa de la vida," *Asclepio*, vol. 61, no. 2, pp. 129–140, 2009.

[5] R. Dawkins, *The Selfish Gene*, Clarendon Press, Oxford, 1976.

[6] Juan J. Merelo-Guervos, Antonio M. Mora, Carlos M. Fernandes, and Anna I. Esparcia-Alcázar, "Designing and testing a pool-based evolutionary algorithm," *Natural Computing*, vol. 12, no. 2, pp. 149–162, 2013.

[7] Juan J Merelo-Guervos, Antonio Mora, J Albert Cruz, Anna I. Esparcia-Alcazar, and Carlos Cotta, "Scaling in distributed evolutionary algorithms with persistent population," in *Evolutionary Computation (CEC), 2012 IEEE Congress on*. IEEE, 2012, pp. 1–8.

[8] J Chris Anderson, Jan Lehnardt, and Noah Slater, *CouchDB: the definitive guide*, O'Reilly Media, Inc., 2010.

[9] Douglas Crockford, "Json: The fat-free alternative to xml," in *15th International World Wide Web Conference*, 2006.

[10] Natalio Krasnogor, BP Blackburne, Edmund K Burke, and Jonathan D Hirst, "Multimeme algorithms for protein structure prediction," in *Parallel Problem Solving from Nature-PPSN VII*, pp. 769–778. Springer, 2002.

[11] Jim E Smith, "Coevolving memetic algorithms: a review and progress report," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 37, no. 1, pp. 6–17, 2007.

[12] Richard Dawkins, *The selfish gene*, Oxford university press, 2006.

[13] David E. Goldberg, Kalyanmoy Deb, and Jeff Horn, "Massively multimodality, deception and genetic algorithms," in *International Conference on Parallel Problem Solving from Nature II (PPSN-II)*, R. Männer and B. Manderick, Eds., pp. 37–46. North-Holland, 1992.

[14] Kalyanmoy Deb and David E. Goldberg, "Analyzing deception in trap functions.," in *Second Workshop on Foundations of Genetic Algorithms*, L. Darrell Whitley, Ed., Vail, Colorado, USA, 1993, pp. 93–108, Morgan Kaufmann.

[3] `http://dnemesis.lcc.uma.es/wordpress/`

[4] `http://anyself.wordpress.com/`

Apéndice

I. Problemas empleados

Hemos considerado la función completamente engañosa de Deb [14] (Trampa), y el problema engañoso masivamente multimodal de Goldberg et al. [13] (MMDP). La función Trampa de 4 bits se define de manera que tenga un único óptimo global en una posición aislada, y un óptimo local hacia el que dirige el gradiente de la función. Matemáticamente:

$$f(b) = \begin{cases} 0{,}6 - 0{,}2 \cdot u(b) & \text{if } u(b) < 4 \\ 1 & \text{if } u(b) = 4 \end{cases} \quad (1)$$

donde $u(b) = u(b_1 \cdots b_4) = \sum_j b_j$. Análogamente, MMDP es una función engañosa y bipolar con dos óptimos globales a distancia máxima el uno del otro, con un atractor engañoso en el punto medio entre ellos. Más concretamente:

$$f(b) = \begin{cases} 1 & u(b) \in \{0, 6\} \\ 0 & u(b) \in \{1, 5\} \\ 0{,}360384 & u(b) \in \{2, 4\} \\ 0{,}640576 & u(b) = 3 \end{cases} \quad (2)$$

En ambos casos se construye un problema de orden superior concatenando k bloques y considerando el fitness total como la suma del fitness de cada bloque. En la experimentación consideramos $k = 16$ (Trampa) y $k = 10$ (MMDP).