XIII Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (XIII MAEB)

MAEB 11:

COMPUTACIÓN EVOLUTIVA:

FUNDAMENTOS Y MODELOS II

Un estudio sobre la influencia de la función objetivo en evolución gramatical para regresión simbólica

J. Manuel Colmenar
Universidad Rey Juan Carlos
C/. Tulipán s/n, 28933, Móstoles, Spain
Email: josemanuel.colmenar@urjc.es

J. Ignacio Hidalgo
Universidad Complutense de Madrid
C/. Prof. José García Santesmases 9, 28040, Madrid, Spain
Email: hidalgo@dacya.ucm.es

***Resumen*—La evolución gramatical es una variante de la programación genética, que permite, entre otras características, introducir conocimiento del problema en el proceso de búsqueda. Una de las aplicaciones más importantes de esta técnica, al igual que de la programación genética tradicional, es la regresión simbólica. La regresión simbólica consiste en obtener expresiones matemáticas que modelen un conjunto de datos. Como todo algoritmo evolutivo, la búsqueda tiene que estar guiada por una función objetivo. Esta función objetivo, en el caso de la regresión simbólica, debe medir la proximidad de la función matemática reportada por el algoritmo a los datos objeto del problema. Por lo tanto, lo más habitual es utilizar el error cuadrático medio, el valor de R^2, o alguna otra variante de la diferencia entre los puntos. En este artículo investigamos la influencia de la utilización del error cuadrático medio y de R^2 en el proceso de búsqueda y en la calidad de las soluciones obtenidas con evolución gramatical en problemas de regresión simbólica.**

I. Introducción

Uno de los paradigmas más importantes en cuanto a las aplicaciones de la inteligencia artificial es la computación evolutiva y dentro de ella es de fundamental importancia la programación genética (PG, o GP, de sus siglas en inglés: *Genetic Programming*) [6]. La PG es una metodología que permite obtener de una forma automática programas que representen soluciones a problemas o que realicen una determinada función. Utiliza una representación de las soluciones en forma de árboles, lo que hace que en ocasiones el tamaño de memoria necesario para representar las soluciones de la población sea excesivo. Por ello se han desarrollado alternativas que funcionen bajo los mismos principios pero que permitan reducir tanto el espacio ocupado en memoria, como el tiempo efectivo de búsqueda de soluciones. La evolución gramatical, también conocida como gramáticas evolutivas (de una mala traducción del inglés, *Grammatical Evolution*) [7], es una variante de la programación genética en la que se utilizan una serie de reglas recogidas en una gramática para construir los árboles que representan una solución. Esto permite introducir conocimiento del problema en el proceso de búsqueda y controlar el tamaño de los árboles que representan a las soluciones.

Una de las aplicaciones más relevantes de esta técnica, al igual que de la programación genética tradicional, es la regresión simbólica. La regresión simbólica consiste en obtener expresiones matemáticas que representen un conjunto de datos. Sus aplicaciones en la actualidad son innumerables ya que es un técnica fundamental en el análisis de datos y modelado de sistemas a partir de registros de información [3, 4]. Como todo algoritmo evolutivo, la búsqueda tiene que estar guiada por una función objetivo. Esta función objetivo, en el caso de la regresión simbólica, debe medir la proximidad de la función matemática reportada por el algoritmo a los datos objeto del problema. Por lo tanto, lo más habitual es utilizar el error cuadrático medio, el valor de R^2, o alguna otra variante de la diferencia entre los puntos. Sin embargo, aunque el objetivo buscado sea el mismo, al utilización de una u otra función puede tener consecuencias importantes en el resultado final del proceso de búsqueda.

En este artículo se analiza la influencia de la utilización del error cuadrático medio y de R^2 tanto en el proceso de búsqueda com en la calidad de las soluciones obtenidas con evolución gramatical en problemas de regresión simbólica. Como se verá en la discusión del trabajo, a partir de las expresiones de las funciones objetivo, intuitivamente se puede pensar que cuando se utiliza el valor de R^2 los resultados de la búsqueda deben reportar expresiones que tengan una alta correlación con los datos, pese a que no estemos obteniendo una función que represente exactamente todos los datos. Sin embargo, al utilizar el error cuadrático medio (*RMSE*, del inglés *Root Mean Square Error*) se esperaría que las buenas soluciones representen los datos de manera más fiel. Para realizar el estudio se han utilizado varios problemas clásicos de

regresión simbólica, y se han comparado los resultados utilizando gramáticas iguales con funciones objetivo diferentes. Los resultados muestran la diferencia entre los modelos y soluciones obtenidas así como la influencia de la función objetivo en estos resultados.

El resto del artículo está organizado como sigue. En la sección II se describe brevemente la técnica de la evolución gramatical. En la sección III se revisa el diferente enfoque de las funciones objetivo bajo estudio. La exposición de los resultados y su análisis se realiza en la sección IV y la sección V resume las principales conclusiones de este trabajo.

II. Evolución Gramatical

La evolución gramatical es un paradigma evolutivo que deriva de la programación genética. En resumen, la aportación de la evolución gramatical consiste en modificar la representación de los individuos utilizando cromosomas en lugar de árboles. Este cambio permite que se puedan utilizar operadores clásicos de algoritmos genéticos como el cruce, mutación o selección, cuya implementación es más sencilla y eficiente al ejecutar sobre cromosomas que sobre árboles. Además, para realizar la decodificación del cromosoma se utiliza una gramática, que incluye reglas de producción que permiten generar las expresiones que corresponden al fenotipo del individuo.

Pese a ser una técnica reciente, durante los últimos años se ha utilizado de forma extensa en problemas de modelado. Para mayor información sobre evolución gramatical se puede recurrir a los trabajos mencionados en la introducción y también acudir a [2] ó [5] para ver el detalle del proceso de decodificación.

Como ejemplo de gramática, la Figura 1 muestra la que se ha utilizado en los experimentos de este artículo. En particular, esta gramática corresponde al conjunto de datos de referencia Vladislavleva-F8 porque considera dos variables de entrada, como se indica en la última regla. Para el resto de las pruebas de referencia, esta regla será la única modificada, adaptada al número de variables de entrada.

III. Funciones objetivo en problemas de regresión simbólica

A continuación, se planteará la formulación de la funciones objetivo bajo estudio en este artículo. Siendo O el vector de valores observados (aquellos provinientes de la optimización) y E el vector de valores esperados (formado por la variable objetivo), y $\overline{O}$ y $\overline{E}$ los valores promedio de cada uno de ellos, las expresiones para las funciones objetivo son las siguientes:

```
# Rule to create the main structure of the
    produced expression
<expr> ::=  (<expr> + <expr>) / (<expr> + <
    expr>)
       | <expr> +  <expr> |  <expr> - <expr
           >
       | <expr> * <expr> |  <expr> / <expr>
       | <c> | <c>.<c> | <c><c>.<c><c> | <
           var> | <var>
       | <expr> <op> <expr> |  <c>^<var>
       | exp(abs(<c> <op> <var>)) | log(abs
           (<c> <op> <var>))

# Digits
<c>  ::= 0|1|2|3|4|5|6|7|8|9

# Arithmetic operands
<op> ::= +|-|*|/

# Input variables
<var> ::= X1 | X2
```

Figura 1: Gramática diseñada para abordar los experimentos. El ejemplo corresponde a Vladislavleva-F8, que trata 2 variables de entrada.

$$R^2 = \left(\frac{\sum_i ((E_i - \overline{E})(O_i - \overline{O}))}{\sqrt{\sum_i (E_i - \overline{E})^2 \sum_i (O_i - \overline{O})^2}} \right)^2$$

$$RMSE = \sqrt{\frac{1}{N} \sum_i (O_i - E_i)^2}$$

R^2 no es sólo una medida de la bondad de ajuste, sino también una medida de hasta qué punto el modelo (el conjunto de variables independientes que se seleccionó) explica el comportamiento (o la varianza) de su variable dependiente. Por lo tanto, si un modelo tiene una R^2 de 0,6, esto explica el 60 % del comportamiento de su variable dependiente. Ahora, si se se usa el R^2 Ajustado que esencialmente penaliza el R^2 por el número de variables que usa el modelo, se puede obtener una idea bastante buena de cuándo hay que dejar de agregar variables al modelo (y eventualmente simplemente obtener un modelo que se ajuste). Si el R^2 Ajustado de los modelos obtenidos con un conjunto de variables es 0,8, y al añadir una variable sólo aumenta en 0,01, lo más indicado es que no sea necesario añadir esa variable. Realmente, si el algoritmo de evolución gramatical está bien diseñado, esto no debe suceder ya que de inicio se habrán seleccionado las variables que intervienen y el propio algoritmo proporcionará un modelo que implícitamente indica qué variables son las más importantes, ya que son las que aparecen en el modelo.

En el caso de RMSE, éste devuelve una estimación de la desviación estándar de los residuales[1] [1]. Por

[1]La diferencia entre los valores reales y los estimados es el residual.

tanto, a mayor RMSE, mayor desviación estimada en los errores del modelo.

En principio, por tanto, R^2 mide el ajuste de un modelo sobre un conjunto concreto de datos, mientras que RMSE mide la dispersión de los errores. A priori, podría parecer que estas medidas indican ambas la calidad de un modelo, por lo que podrían ser intercambiables. Sin embargo, como se verá en la sección de resultados, valores altos de una de las métricas no implican valores altos de la otra.

IV. Resultados experimentales

Los experimentos de este artículo se han realizado sobre un conjunto de tres problemas de regresión estándar, que se denominarán *benchmarks* a partir de este punto, haciendo uso de la expresión comunmente utilizada en inglés. Estos problemas se han seleccionado como parte de los problemas recomendados para PG en [10], y son los siguientes: *Tower*, *Vladislavleva-F8* y *Spatial co-evolution*. A continuación se detallan algunas de las características de estos problemas:

- En el conjunto de datos *Spatial co-evolution* [8], la variable objetivo se define de la siguiente manera:

$$F(x, y) = \frac{1}{1 + x^{-4}} + \frac{1}{1 + y^{-4}}$$

  En el conjunto de entrenamiento original existen 676 registros en los cuales los valores para las variables x e y se muestrean desde -5 a +5 en intervalos de 0.4. En el conjunto de test hay 1000 registros en los que las variables x e y se muestrean en el intervalo $[-5, \ldots, +5]$ aleatoriamente. En conjunto, forman un total de 1676 datos, y se han utilizado como serie de regresión al completo.
- El conjunto de datos *Tower* [9] proviene de un problema industrial sobre medidas de cromatografía de gases en una torre de destilación. El conjunto de datos contiene 5000 registros y 25 variables de entrada potenciales. La variable de respuesta es la concentración de propileno en la parte superior de la torre de destilación. Los datos de *Tower* se pueden descargar en http://www.symbolicregression.com/?q=towerProblem.
- El conjunto de datos *Vladislavleva-F8*, denotado como *VF8* a partir de este momento, define las variables de salida como funciones de x_1 y x_2, de la siguiente manera:

$$F_8(x_1, x_2) = \frac{(x_1 - 3)^4 + (x_2 - 3)^3 - (x_2 - 3)}{(x_2 - 2)^4 + 10}$$

Para todos los *benchmarks* se han estudiado las dos funciones objetivo, R^2 y $RMSE$, con dos tipos de inicialización para la primera población del algoritmo: inicialización aleatoria (*random*) e inicialización *sensible*, que controla el tamaño de los fenotipos de los individuos de la generación inicial evitando que sean excesivamente grandes. De cada experimento se han realizado diez ejecuciones diferentes utilizando los mismos parámetros de entrada para el algoritmo de GE, que se muestran en la Tabla I. Los experimentos se ejecutaron en un ordenador equipado con un procesador Intel Core i7 a 2.9 GHz con 16 GB de RAM sobre sistema operativo Mac OSX 10.13.5.

Parámetro	**Valor**
Población	300 individuos
Generaciones	1700
Selección	Torneo (2 indiv.)
Prob. Cruce	80 %
Prob. Mutación	2 %
Función objetivo	R^2 ó $RMSE$
Inicialización	*random*, *sensible*
Longitude del Cromosoma	256
Número máximo de wraps	5

Tabla I: Valores de los parámetros utilizados en los experimentos.

El primer aspecto a analizar es la evolución de las ejecuciones a través del análisis del mejor individuo. Para ello, se tomó el valor de la función objetivo cada 100 generaciones, en cada una de las ejecuciones. Dado que se lanzaron 10 ejecuciones para cada experimento, los datos que se muestran se corresponden con el promedio de las 10 ejecuciones.

En primer lugar, las figuras 2 y 3 muestran la evolución para el caso de la inicialización con una población aleatoria (*random*), utilizando como función objetivo R^2 y $RMSE$ respectivamente. Dado que el algoritmo trata de minimizar, el valor que se presenta en el caso de R^2 es en realidad $1 - R^2$, y así será para el resto de experimentos. Como se puede apreciar, el problema más difícil de resolver es *Tower*. Por otro lado, la evolución tiene una tendencia hacia abajo menos pronunciada a partir de la generación 500 aproximadamente en ambas gráficas.

Las figuras 4 y 5 muestran la convergencia para el caso de la inicialización *sensible*, utilizando como función objetivo $1 - R^2$ y $RMSE$ respectivamente. De nuevo, el comportamiento es similar en ambos casos, siendo *Tower* el problema más difícil. En el caso de la comparativa entre los diferentes tipos de inicialización, se puede ver que entre las figuras 2 y 4, que utilizan la misma función objetivo, $1 - R^2$, y diferente inicialización, la inicialización *sensible* alcanza mejores valores promedio a lo largo de toda la ejecución del algoritmo en los tres *benchmarks*.

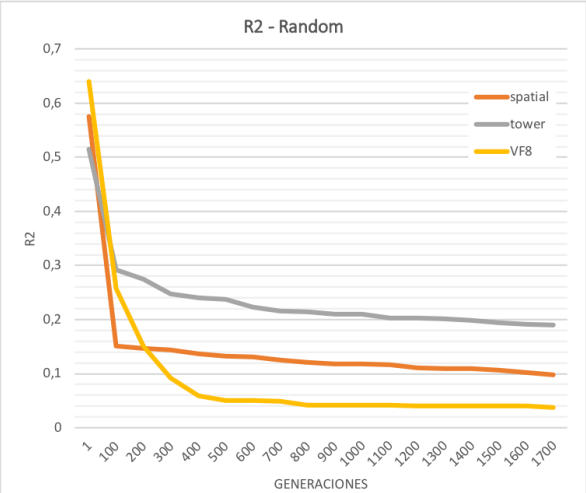

Figura 2: Valores promedio de la función objetivo: $1 - R^2$. Inicialización *random*.

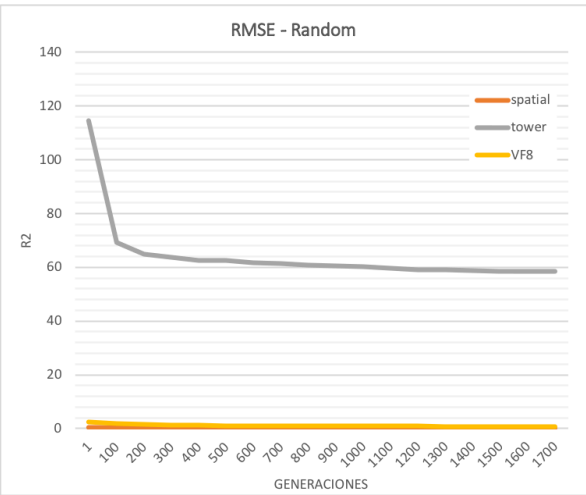

Figura 3: Valores promedio de la función objetivo: $RMSE$. Inicialización *random*.

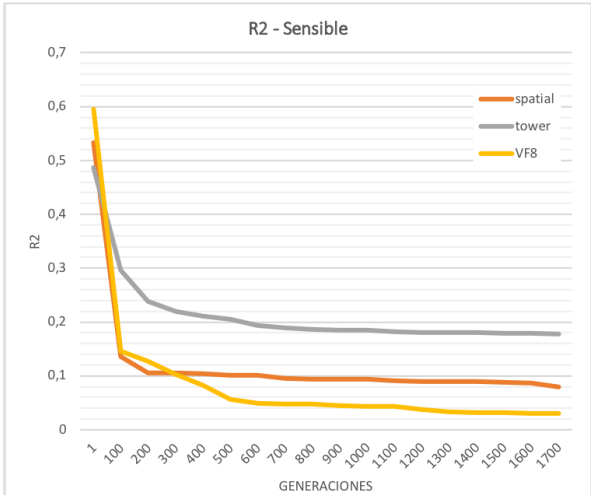

Figura 4: Valores promedio de la función objetivo: $1 - R^2$. Inicialización *sensible*.

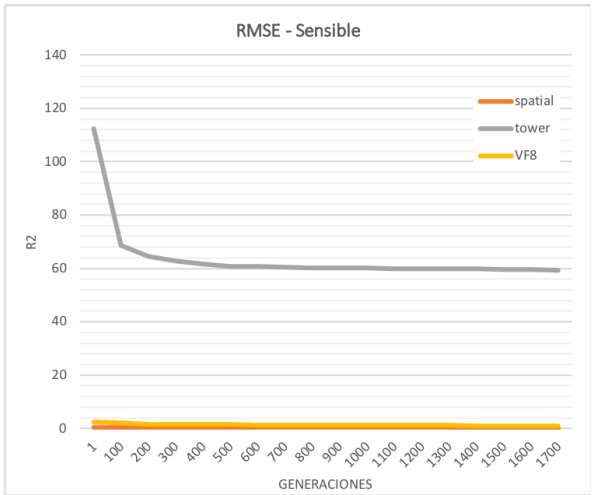

Figura 5: Valores promedio de la función objetivo: $RMSE$. Inicialización *sensible*.

En el caso de la optimización con $RMSE$, la inicialización *sensible* no aporta valores mejores en la evolución. A pesar de que la escala no permite ver las diferencias gráficamente en las figuras 3 y 5, sí que se aprecia que al principio los valores promedio son mejores, pero según avanza la evolución, los experimentos con inicialización aleatoria obtienen mejores valores promedio.

Independientemente de las funciones objetivo utilizadas, se han calculado los valores de R^2 y $RMSE$ para todas las soluciones obtenidas en los experimentos. La figuras 6 y 7 muestran, respectivamente, los valores promedio de R^2 y $RMSE$ para cada uno de los *benchmarks* en los diferentes casos. Se denota el experimento con optimización a través de R^2 e inicialización aleatoria como *R2-Random*; optimización a través de R^2 e inicialización *sensible* como *R2-Sensible* y, de manera análoga, para $RMSE$ y las dos inicializaciones estudiadas como *RMSE-Random* y *RMSE-Sensible*.

En la Figura 6 se puede apreciar cómo los valores de *R2-Random* y *R2-Sensible* son mayores que en la optimización con $RMSE$. En el caso, por ejemplo, de *Tower*, la diferencia es muy significativa, llegando a estar por encima del 30 %. Los valores concretos de R^2 se muestran en la Tabla III. Se puede apreciar también que la inicialización *sensible* mejora los resultados hasta en un 2 % para los experimentos donde la función objetivo es $1 - R^2$.

La Figura 7 muestra los valores de $RMSE$ para las mismas soluciones que se han utilizado en la

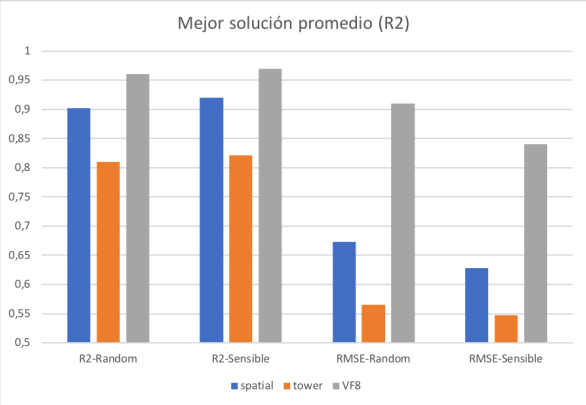

Figura 6: Valores promedio de R^2 para los diferentes experimentos. Mejor cuanto más alto.

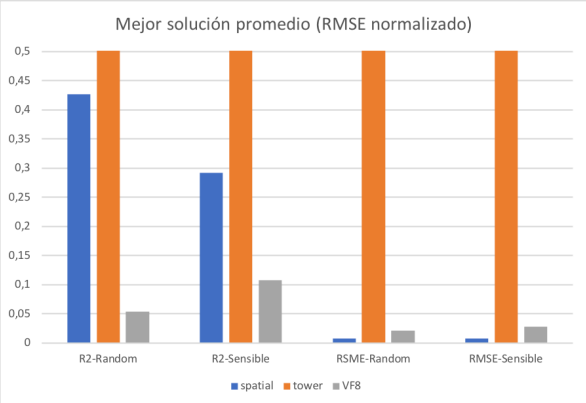

Figura 7: Valores promedio de $RMSE$ normalizados para los diferentes experimentos. Mejor cuanto más bajo.

	spatial	**tower**	**VF8**
R2-Random	0,901794689	0,80986102	0,960807366
R2-Sensible	0,920002539	0,821656958	0,969546579
RMSE-Random	0,672605164	0,56536177	0,910126819
RMSE-Sensible	0,627583903	0,547273778	0,840721671

Tabla II: Valores promedio de R^2.

Figura 6. En este caso, dado que los valores presentan diferentes órdenes de magnitud, se han normalizado respecto a la décima parte del valor máximo. Los valores concretos de $RMSE$ se presentan en la Tabla III. Como se puede apreciar, los valores de $RMSE$ son mucho mejores en el caso de los algoritmos que utilizan esta métrica como función objetivo, llegando a obtener un orden de magnitud de diferencia. Resulta significativo señalar que la inicialización *sensible* no mejora los resultados en este caso. Al contrario, se llega a peores resultados en los tres problemas bajo estudio.

Estos resultados son coherentes con lo que se espera de algoritmo de optimización, que consigue la mejora

	spatial	**tower**	**VF8**
R2-Random	15,58366595	348,2669206	1,942964282
R2-Sensible	10,67370612	365,2646472	3,94804835
RMSE-Random	0,272769052	58,39444643	0,787039727
RMSE-Sensible	0,298058917	59,40281873	1,035522363

Tabla III: Valores promedio de $RMSE$.

en los valores de la función objetivo. Sin embargo, también demuestran que las funciones objetivo bajo estudio no son equivalentes.

Por un lado, la función objetivo R^2 muestra la correlación entre dos series de valores. Una elevada correlación indica que la tendencia de las dos series es similar, independientemente del valor concreto del error entre el valor observado y el valor esperado. Un modelo con elevado valor de R^2 puede conseguir buenas predicciones sobre valores no conocidos.

Por otro lado, la función objetivo $RMSE$ muestra el error o diferencia entre una serie de predicciones y los datos esperados correspondientes. Al minimizar $RMSE$ se reduce el error, por lo que la serie de predicciones se acerca a la serie esperada. La consecuencia es que se obtienen modelos cuya precisión es alta, pero cuya capacidad de predicción puede ser mala, debido al efecto de sobreajuste, también llamado *overfitting*.

Con objeto de evaluar la calidad de las soluciones obtenidas a través de una métrica diferente, se propone en este artículo la utilización de la métrica conocida como similitud del coseno. Esta medida interpreta ambas series de datos, observada y esperada, como vectores O y E respectivamente. De esta manera, se calcula el coseno del ángulo que forman ambos vectores utilizando la siguiente expresión:

$$Coseno(\theta) = \frac{\sum_i O_i E_i}{\sqrt{\sum_i O_i^2}\sqrt{\sum_i E_i^2}}$$

Si el valor del coseno es cercano a 1, entonces el ángulo que forman los vectores es cercano a 0º, por lo que se tratará de series similares. Si el valor es cercano a -1, entonces el ángulo es cercano a 180º, lo que significa que las series tienen tendencia similar, pero en sentido opuesto. En caso de que el valor del coseno esté cerca de 0, se trata de series perpendiculares, lo que se considera como la menor similitud posible. Sobre las soluciones obtenidas en los experimentos se ha calculado la similitud de coseno. La Tabla IV muestra los valores obtenidos para los experimentos donde la inicialización es aleatoria. Para cada *benchmark* se muestran los valores de las optimizaciones donde la función objetivo es $1 - R^2$ (R2) y también los valores donde la función

de coste ha sido $RMSE$. Los valores presentados corresponden a las 10 ejecuciones realizadas. Como se aprecia en la tabla, los valores para $RMSE$ son más altos y más consistentes, tal y como indican los valores promedio que se muestra en la última fila. En el caso de R^2, existen valores altos, cercanos al 1, pero también existen valores negativos y alguno cercano al $0,5$.

	spatial		tower		VF8	
Run	R2	RSME	R2	RSME	R2	RSME
1	0,9393	0,9931	0,9514	0,9881	0,9255	0,9758
2	0,9931	0,9666	0,96	0,9857	0,9865	0,9732
3	0,9807	0,9877	0,9541	0,9856	0,9782	0,9534
4	-0,7232	0,9962	0,9846	0,9886	0,9634	0,9736
5	0,9643	0,9897	0,9754	0,9838	0,8801	0,9434
6	0,5044	0,9698	-0,9682	0,9844	0,6949	0,9871
7	0,9608	0,9902	0,9655	0,9874	0,905	0,961
8	0,946	0,9723	-0,9826	0,9871	-0,9157	0,9535
9	0,9622	0,9905	0,9784	0,9874	0,9879	0,9487
10	0,9588	0,9914	0,9721	0,985	0,9824	0,9265
Avg.	0,7486	0,9847	0,5791	0,9863	0,7388	0,9596

Tabla IV: Similitud de coseno para los experimentos con inicialización *random*. La última fila muestra el valor promedio.

La Tabla V muestra las mismas estadísticas, pero para los experimentos donde la inicialización es *sensible*. De nuevo, la tendencia es la misma, aunque los resultados para los experimentos donde la función de coste es $1 - R^2$ son peores en este caso. Es destacable el conjunto de resultados de *spatial*, que obtuvo el mejor promedio de R^2 para la optimización *R2-Sensible* (ver Tabla II), mientras que los valores de similitud de coseno de ese mismo experimento son los peores, con dos valores por debajo de $0,29$.

	spatial		tower		VF8	
Run	R2	RSME	R2	RSME	R2	RSME
1	0,9966	0,9886	0,9391	0,9866	0,8819	0,9733
2	0,9705	0,9911	-0,965	0,9862	-0,9872	0,9673
3	0,9708	0,9914	0,9793	0,9844	-0,9889	0,9456
4	0,2871	0,982	-0,9777	0,987	0,9486	0,8021
5	0,9883	0,9743	0,9576	0,9863	0,6819	0,9566
6	0,9519	0,9762	0,9868	0,9856	0,9682	0,845
7	0,9112	0,9897	0,9611	0,9853	0,9178	0,8424
8	0,9634	0,9725	0,9687	0,9863	0,9484	0,953
9	0,957	0,9739	-0,9472	0,9863	0,9791	0,9485
10	0,1608	0,991	0,9707	0,9844	0,9199	0,9714
Avg.	0,8158	0,9831	0,3873	0,9858	0,527	0,9205

Tabla V: Similitud de coseno para los experimentos con inicialización *sensible*.

V. Conclusiones y trabajo futuro

A la vista de los experimentos realizados en este artículo, se puede afirmar que R^2 y $RMSE$ no son funciones que lleven a soluciones equivalentes en todos los casos. Muy al contrario, pueden obtener resultados realmente diversos. Sin embargo, este es un estudio muy preliminar que se deberá extender en el futuro.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad de España a través de los proyectos con referencias TIN2014-54806-R y TIN2015-65460-C2.

Referencias

[1] D. L. J. Alexander, A. Tropsha, and David A. Winkler. Beware of r^2: Simple, unambiguous assessment of the prediction accuracy of qsar and qspr models. *Journal of chemical information and modeling*, 55.7:1316–1322, 2015.

[2] J. M. Colmenar and J. I.. Hidalgo. Análisis de algoritmos de evolución gramatical en problemas de regresión simbólica. In *XII Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2017)*, pages 901–910, Barcelona, Spain, 2017. Universitat Pompeu Fabra.

[3] Ian Dempsey, Michael O'Neill, and Anthony Brabazon. Live trading with grammatical evolution. In *GECCO 2004 Workshop Proceedings*, Seattle, Washington, USA, 26-30 June 2004.

[4] J. Ignacio Hidalgo, J. Manuel Colmenar, José L. Risco-Martin, Alfredo Cuesta-Infante, Esther Maqueda, Marta Botella, and José Antonio Rubio. Modeling glycemia in humans by means of grammatical evolution. *Applied Soft Computing*, (20):40–53, 2014.

[5] José Ignacio Hidalgo, J. Manuel Colmenar, José Luis Risco-Martín, Alfredo Cuesta-Infante, Esther Maqueda, Marta Botella, and José Antonio Rubio. Modeling glycemia in humans by means of grammatical evolution. *Appl. Soft Comput.*, 20:40–53, 2014.

[6] J. R. Koza. *Genetic Programming*. The MIT Press, Cambridge, Massachusetts, 1992.

[7] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Trans. Evolutionary Computation*, 5(4):349–358, 2001.

[8] Ludo Pagie and Paulien Hogeweg. Evolutionary consequences of coevolving targets. *Evolutionary Computation*, 5:401–418, 1998.

[9] E. J. Vladislavleva, G. F. Smits, and D. den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, April 2009.

[10] David R. White, James McDermott, Mauro Castelli, Luca Manzoni, Brian W. Goldman, Gabriel Kronberger, Wojciech Jaśkowski, Una-May O'Reilly, and Sean Luke. Better GP benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29, Mar 2013.

Un análisis preliminar de nuevos modelos de mutación dirigida en algoritmos genéticos.

B. Rodríguez-Puerta, Francisco Díaz-Barrancas, F. Chávez, F. Fernández de Vega
Dpto. de Ingeniería en Sistemas Informáticos y Telemáticos
Universidad de Extremadura
06800 Mérida, España.
Email: {brpuerta, frdiaz, fchavez, fcofdez}@unex.es

***Resumen*—En este artículo se presenta un análisis preliminar de un modelo de operador de mutación dirigida para problemas con codificación binaria y sin epistasis. Esta versión del operador permite asociar una probabilidad de mutación a cada gen de cada individuo, proporcional a la influencia que ha tenido dicho gen sobre la calidad del individuo durante el proceso evolutivo. Estos valores de probabilidad, permiten a cada individuo realizar mutaciones de manera dirigida, con el objetivo de reducir el tiempo de convergencia del algoritmo.**

El conjunto de experimentos realizado con el nuevo operador de mutación demuestra que el algoritmo genético converge a soluciones en etapas más tempranas del proceso evolutivo, en comparación con la mutación clásica. Se han llevado a cabo una serie de experimentos con un problema clásico de test, donde aplicando el nuevo operador de mutación dirigida, se consiguen buenos resultados. Aunque estos resultados son aún muy preliminares, esperamos poder continuar el estudio en problemas más complejos en el futuro, y mostrar así la utilidad de esta versión de la mutación dirigida en otros contextos.

I. INTRODUCTION

Los algoritmos evolutivos [5] (AEs), son un conjunto de técnicas de optimización y búsqueda de soluciones basados en la teoría de la evolución de Charles Darwin. Inspirados en esta teoría, utilizan una población de individuos, soluciones candidatas al problema que pretende resolverse, que tratan de adaptarse mediante evolución al entorno para resolver el problema en cuestión.

Dentro de la familia de los AEs se encuentran los algoritmos genéticos [6] (AGs). Estos nos ofrecen técnicas robustas de búsqueda y optimización para la obtención de resultados en una gran variedad de problemas. Por ejemplo, son ampliamente utilizados en los problemas conocidos como "NP completos", donde ofrecen la posibilidad de encontrar soluciones satisfactorias en tiempos razonables. En la actualidad los AGs son ampliamente utilizados por la comunidad científica.

Un AG típico se compone de los siguientes elementos:

- Un individuo: es la representación de una posible solución al problema.
- Una población: compuesta por un conjunto de individuos.
- Una función de fitness: que permite evaluar la calidad de un individuo.
- Operadores genéticos: que permiten generar una nueva población a partir de la anterior (operador cruce, operador mutación y operador selección).
- Parámetros de control: que permiten controlar la ejecución del algoritmo.

Para maximizar la eficiencia de los AGs es posible modificar los parámetros y operadores genéticos anteriormente mencionados.

Entre los componentes del algoritmo, hay tres parámetros fácilmente ajustables que permiten modificar el proceso de búsqueda de soluciones: probabilidad de mutación (P_m), probabilidad de cruce (P_c) y tamaño de la población (T). Son numerosos los estudios existentes que se centran en proponer heurísticas para optimizar dichos parámetros con la finalidad de obtener mejores resultados.

En este artículo se presenta una alternativa a uno de los operadores utilizados en el proceso de un AG, el operador de mutación. Está demostrado que este operador es necesario para garantizar la convergencia del algoritmo evolutivo. Su objetivo es la modificación del valor de un gen determinado, previamente seleccionado al azar, perteneciente un individuo. Este operador está contralado por el parámetro conocido como probabilidad de mutación, que determina cuándo se aplica dicha mutación.

La importancia del operador mutación ha sido estudiada ampliamente en la literatura, y ha quedado

demostrado que sin él no se garantiza la convergencia del algoritmo [9]. A pesar de todo, el proceso de convergencia puede ser lento, dependiendo de la dificultad del problema. Frecuentemente, esta dificultad implica el uso de cromosomas de tamaño grande, y el tamaño del cromosoma afecta a la eficacia del operador de mutación clásico, que selecciona al azar una posición a mutar: cuanto más largo es el cromosoma, más tiempo será necesario para poder mutar alguna posición concreta del mismo.

En este artículo se presenta una nueva implementación para el operador genético mutación, para problemas en los cuales no existe epistasis entre los genes que conforman el cromosoma de un individuo. Existe epistasis entre los genes de un individuo cuando la expresión de uno o más genes depende de la expresión de otro gen del mismo individuo. Esta interacción sucede cuando la modificación de un gen influye de manera directa sobre el valor en uno o varios genes diferentes del mismo individuo.

La nueva propuesta que se introduce en este trabajo, que como veremos más adelante es una variación de algunos modelos de mutación existente, pretende asociar una probabilidad de mutación a cada gen dentro del cromosoma del individuo. Esta probabilidad se debe ir actualizado con cada mutación sufrida y va a depender de la influencia que dicha mutación pueda tener sobre el valor del fitness global del individuo.

Los resultados presentados en este trabajo demuestran que es posible alcanzar mejores soluciones en un tiempo menor de cómputo, así como utilizando menores generaciones en el proceso evolutivo.

El resto del artículo se divide de la siguiente forma: en la Sección II se detalla un estado del arte de trabajos anteriores donde se aborda el problema de mutación dirigida, la Sección III se describe la metodología utilizada, los resultados de este trabajo se presentan en la Sección IV. Finalmente las conclusiones del mismo se abordan en la Sección V.

II. Estado del arte

La modificación de los diferentes operadores de un AG es un campo ampliamente estudiado por la comunidad científica. Desde el inicio, los investigadores no han dejado de proponer mejoras para los operadores genéticos con la finalidad de obtener AGs que converjan a una solución de forma más rápida. En esta sección nos centraremos en el estudio de posibles mejoras del operador de mutación, fijándonos en particular en versiones del operador que permitan dirigir su acción hacia zonas específicas del cromosoma que puedan ser de mayor interés.

Una de las primeras mejoras para el operador mutación que se propusieron fue dotar al AG de la capacidad de adaptar la probabilidad de mutación establecida [1], esta técnica es conocida como mutación adaptativa. La idea es simple pero efectiva. Dependiendo del valor de fitness obtenido por los individuos de la población en las diferentes generaciones, la probabilidad se adapta para permitir más o menos mutaciones, de esta manera cuando el fitness global de la población es malo, cuando el proceso de convergencia se estanca, el AG aumentará la probabilidad de mutación, con la esperanza de que estas mutaciones mejoren a los individuos, y viceversa, disminuirá la probabilidad de mutación cuando el fitness de los individuos sea buena para no degradar la calidad de la solución. Esta idea también ha sido aplicada con la probabilidad de cruce. Existen propuestas más avanzadas dentro de la mutación adaptativa, por ejemplo en [2] se propone un operador que modifica la probabilidad de mutación en función del fitness de los individuos de la población, utilizando para ello el algoritmos k-means. Otro enfoque muy utilizado dentro de la mutación adaptativa ha sido la modificación del AG para que en lugar de disponer de una probabilidad de mutación global, cada individuo cuente con una propia. Esta idea se ha codificado tradicionalmente con un gen más al principio del cromosoma y su cálculo se obtiene al generar un número aleatorio dentro de una distribución normal [3]. A lo largo de los años se han utilizado otro tipo de distribuciones de probabilidad, por ejemplo una distribución Levy [4], pero manteniendo la idea anteriormente descrita.

Entre las técnicas utilizadas para mejorar el operador genético de mutación se encuentra la que se conoce como mutación dirigida. Ésta, a diferencia de la mutación adaptativa, basa su funcionamiento en guiar al AG hacia mutaciones más beneficiosas, entendiendo como mutaciones más beneficiosas aquellas que hacen que el valor del fitness de un individuo mejore. En el artículo presentado por Dinabandhu Bhandari et al [7] se propone un operador de mutación que, dependiendo del valor de fitnees obtenido en la generaciones anteriores, calcula un punto de mutación para los individuos que integran la población actual.

En el trabajo presentado por Bäck et al.[8] propone una idea novedosa dentro de la mutación adaptativa. Se asocia una probabilidad de mutación a cada uno

de los genes del cromosoma del individuo. Esta probabilidad es un valor aleatorio dentro de un rango. Utilizando esta idea como punto de partida, en este artículo exploramos otras posibilidades, haciendo una primera implementación de una idea similar.

La mutación dirigida que presentamos más adelante, asocia una probabilidad de mutación a cada uno de los genes que componen el cromosoma, en función a la influencia histórica que cada gen ha tenido sobre la mejora de la calidad del individuo, a lo largo del proceso evolutivo. A diferencia del trabajo propuesto en [8], esta propuesta tiene en cuenta cómo ha evolucionado cada individuo de la población, para tratar de guiarlo hacia mutaciones más beneficiosas. El operador que se propone dirige las mutaciones a genes concretos del cromosoma, que mejoraran la calidad del individuo, pero en ningún momento modifica el valor de probabilidad de mutación del AG.

III. Propuesta del nuevo operador mutación

En esta sección se describe en detalle el nuevo operador genético de mutación propuesto, al cual denominamos *mutación dirigida*, y que se considera en el contexto de problemas de codificación binaria sin epistasis. El objetivo es asociar un valor de probabilidad de mutación a cada gen del cromosoma, y utilizar estos valores de probabilidad en el proceso de mutación. Este nuevo modelo conlleva la utilización de un nuevo vector de probabilidades asociado a cada cromosoma, lo que implicará cambios en el operador de cruce. Estos cambios serán mínimos y solo se utilizan para mantener esta información actualizada en el proceso de cruce.

El nuevo operador de mutación dirigida asigna diferentes probabilidades de mutación a cada uno de los genes de cada individuo de la población, valores que dependerán de la influencia histórica que cada gen ha tenido en la mejora de ese individuo a lo largo del proceso evolutivo.

Para conocer en cada momento la influencia que cierto gen ha tenido sobre la calidad del individuo que lo contiene, es necesario disponer en la codificación del individuo de una nueva estructura capaz de almacenar dicha información. En nuestro caso se ha optado por utilizar un vector, denominado *vector de probabilidad de mutación* (VPM). La Figura 1 muestra un ejemplo gráfico del nuevo cromosoma.

El vector de probabilidades se encarga de almacenar la probabilidad de mutación asociada a cada gen

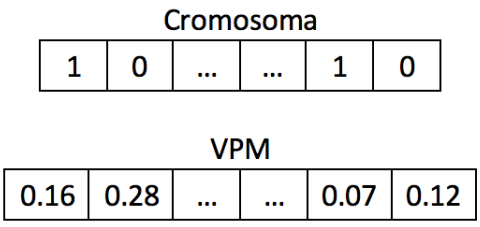

Figura 1. Ejemplo de codificación del VPM

del cromosoma (PMG). La suma de todas las probabilidades contenidas en el VPM debe ser siempre 1, ver figura 1, no pudiendo superar ni disminuir este valor en ningún momento del ciclo evolutivo del AG.

$$\sum_{i=1}^{T} PMG_i = 1 \tag{1}$$

donde T es el número de genes del cromosoma (Tamaño) y PMG_i es la probabilidad de mutación del gen i-ésimo.

La actualización del VPM es realizada cada vez que un individuo de la población sufre una mutación. Antes de comenzar a explicar en detalle cómo funciona el nuevo operador es importante indicar como se inicializan las PMG del VPM para cada individuo en el proceso de generación de la población inicial. Al generar la población inicial, aún no se tienen datos del histórico de evolución de la fitness, de ahí que todas las posiciones del vector VPM de cada individuo comienzan con la misma probabilidad, esta es calculada según la expresión 2

$$PMG = \frac{1}{T} \tag{2}$$

donde T es el tamaño del cromosoma

III-A. Mutación

Con el vector de probabilidad de mutación inicializado, los pasos a realizar por el nuevo operador de mutación, cada vez que queremos aplicar una mutación a un individuo, son los siguientes:

1. Comprobar si el individuo debe mutar, esta acción viene determinada por el parámetro probabilidad de mutación (P_m), presente en la implementación clásica de cualquier AG.
2. Haciendo uso del método de selección conocido como *ruleta* sobre el VPM del individuo, se selecciona el gen que sufrirá la mutación. Esta técnica posibilita que cualquier gen pueda

ser seleccionado para sufrir una mutación, favoreciendo la selección de aquellos genes con un mayor índice de probabilidad -PMG- en el VPM.

3. Cálculo de la fitness del individuo antes de sufrir la mutación. Hay que recordar, que el proceso de mutación se aplica después del cruce, por lo que los individuos recién cruzados no disponen aún de valor de fitness.
4. Realizar la mutación del gen seleccionado en el paso anterior y cálculo del nuevo fitness del individuo.
5. Actualizar el VPM según el valor resultante, al comparar el fitness del individuo antes de sufrir la mutación y después. Para realizar esta operación se aumentará o disminuirá, una cantidad *alpha* (dependiendo de si la mutación mejoró o no al individuo) la probabilidad asociada a la posición del gen mutado, y proporcionalmente, se disminuirán o aumentará el resto de valores correspondientes al resto de los genes. La suma total de los valores del vector debe seguir siendo uno. Este valor *alpha* será a partir de ahora un nuevo parámetro del algoritmo.

III-B. Cruce

El operador genético de cruce también sufre una modificación con respecto a su implementación clásica, debido a la necesidad de manejar correctamente los vectores de probabilidad de los individuos.

En el operador cruce clásico *one-point crossover* se genera una posición aleatoria del cromosoma que se utiliza para cruzar los dos padres intercambiando material genético en los hijos.

El nuevo operador cruce realiza la misma operación sobre los cromosomas, y además, repite ese mismo proceso con los PMGs de cada padre; pero dado que la suma de los PMGs resultantes de los nuevos individuos no tiene porqué ser 1, al finalizar el cruce es necesario realizar una normalización de estos valores -escalado para que la suma sea 1- para que la suma del nuevo VPM resultante de cada individuo hijo sea 1.

IV. Experimentos y resultados

En esta sección se describen los experimentos que se han realizado para comparar el nuevo operador genético de mutación que se propone en este trabajo, con el operador clásico de mutación en problemas sin epistasis entre los genes de los individuos.

Para poder comparar ambos operadores de mutación se ha utilizado un problema bien conocido dentro de la literatura. El problema escogido ha sido el clásico *max-one*. Se trata de un problema de codificación binaria el cual ha sido ejecutado utilizando el operador clásico de mutación y el nuevo enfoque propuesto en este trabajo. La Tabla IV muestra los resultados de los diferentes experimentos que se han realizado. Como puede observarse, estamos particularmente interesados en conocer la utilidad del operador en problemas de tamaño grande: cuando el cromosoma del individuo tiene una longitud tal que son necesarias muchas generaciones para encontrar la solución. Así, hemos probado el operador en tamaños que van de 20 hasta 80 bits.

Para seleccionar un valor alpha adecuado para los experimentos, se ha realizado un estudio previo. Con un tamaño de cromosoma igual a 20 bits se han realizado 150 ejecuciones para cada uno de lo valores del parámetro alpha que hemos querido probar. De esta manera hemos podido determinar el valor alpha que menos generaciones necesitaba para converger a la solución. Como se aprecia en Figura 2, que muestra los promedios para 150 ejecuciones de cada configuración, el valor de alpha que obtiene mejores resultados es 0.1.

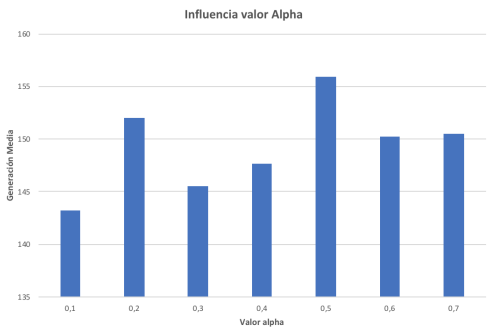

Figura 2. Resultado estudio valor,*Alpha*, cromosoma tamaño 20 bits

Una vez seleccionado el valor alpha adecuado, hemos procedido con los experimentos que se muestran en Tabla IV, de nuevo con 150 ejecuciones de cada configuración. La Figura 3 muestra la generación media que ha tardado el AG en converger a la solución para cada una de las combinaciones mostradas. Como se puede observar el operador de mutación dirigida toma menos generaciones en alcanzar la solución, son necesarias menos generaciones para obtener el mismo resultado que con el operador clásico. Como se aprecia en la gráfica la ventaja obtenida por

Experimento	AG	Tam. Población.	Tam. Cromosoma	Num. Generaciones	Prob. Mutación	Alpha
Ex1	C	10	20	20.000	0.01	-
Ex2	I	10	20	20.000	0.01	0.1
Ex3	C	10	40	20.000	0.01	-
Ex4	I	10	40	20.000	0.01	0.1
Ex5	C	10	60	20.000	0.01	-
Ex6	I	10	60	20.000	0.01	0.1
Ex7	C	10	80	20.000	0.01	-
Ex7	I	10	80	20.000	0.01	0.1

Cuadro I
EXPERIMENTOS REALIZADOS

el nuevo operador respecto al clásico en cuanto al número de generaciones necesarias para obtener una solución aumenta con el tamaño del cromosoma de los individuos de la población. Cuanto el tamaño del individuo es mayor -la dificultad del problema es mayor- la diferencia entre la generación media necesaria para alcanzar solución es mayor entre ambas implementaciones, aportando un mejor rendimiento el operador de mutación dirigida.

La Figura 4 muestra el tiempo medio para alcanzar una solución por ambas implementaciones de los algoritmos. Al igual que sucede con las generaciones, el tiempo medio necesario para obtener la solución por el operador mutación dirigida es menor que la clásica, siendo la diferencia mayor cuanto mayor es el cromosoma del individuo.

Para tamaño de cromosomas superiores a 80 bits ambas implementaciones no siempre convergen a la solución. Aun así, el operador de mutación dirigida converge a la solución un número mayor de veces que el operador clásico (en las 150 ejecuciones realizadas). Para un tamaño de cromosoma igual a 120 bits el operador clásico ha convergido a la solución en 51 ocasiones mientras que el nuevo operador propuesto lo ha hecho 134.

Como decíamos anteriormente, lo aquí mostrado es un análisis preliminar de una variación del operador de *mutación dirigida*. En próximos trabajos pretendemos hacer una comparativa con alguno de los métodos de mutación más cercanos al aquí propuesto, y realizar una selección más adecuada de problemas de test, para estudiar con mayor detalle la potencialidad del nuevo operador. No obstante, estos resultados preliminares muestran el camino a seguir.

V. Conclusiones

Este artículo presenta un análisis preliminar de una nueva versión del operador de mutación adaptado a problemas en los que no existe epistasis. El objetivo de la propuesta es ayudar a determinar en cada momento las posiciones del cromosoma que más pueden ayudar en la mejora de la calidad del individuo tras una mutación.

El nuevo operador asocia una probabilidad de mutación a cada gen del individuo, posición dentro del cromosoma, que es proporcional a la influencia histórica que ha tenido esa posición sobre la mejora de la calidad del individuo a lo largo del proceso evolutivo. Así, se utiliza un vector de probabilidades VPM, con una posición por cada gen que existe en el cromosoma PMG, para realizar mutaciones dirigidas sobre genes concretos que probablemente beneficiarán la calidad del individuo. Este vector de probabilidades es actualizado en cada proceso de mutación, con un valor que depende de la mejora (o empeoramiento) que la mutación produce en el nuevo individuo cuando se compara con el individuo sin mutación.

Utilizando el clásico problema max-one, de codificación binaria y sin epistasis entre los genes del individuo, hemos comparado el AG clásico con la nueva propuesta que utiliza mutación dirigida, con diferentes configuraciones del problema, variando el tamaño de las poblaciones, el tamaño de los individuos y la probabilidad de mutación del AG. En todos los casos la mutación dirigida ha permitido encontrar soluciones en menor número de generaciones, con una influencia que crece cuando lo hace el tamaño del cromosoma.

Esperamos poder extender este trabajo, con una comparativa más completa con otros operadores de mutación similares, y con un mayor número de problemas, tanto de codificación binaria como real, para entender mejor el impacto del nuevo operador de mutación propuesto.

Agradecimientos

Agradecemos el apoyo del Ministerio de Economía y Competitividad proyecto TIN2017-85727-C4-{2,4}-P, Junta de Extremadura, Consejería de Comercio y Economía, proyecto IB16035 a través del

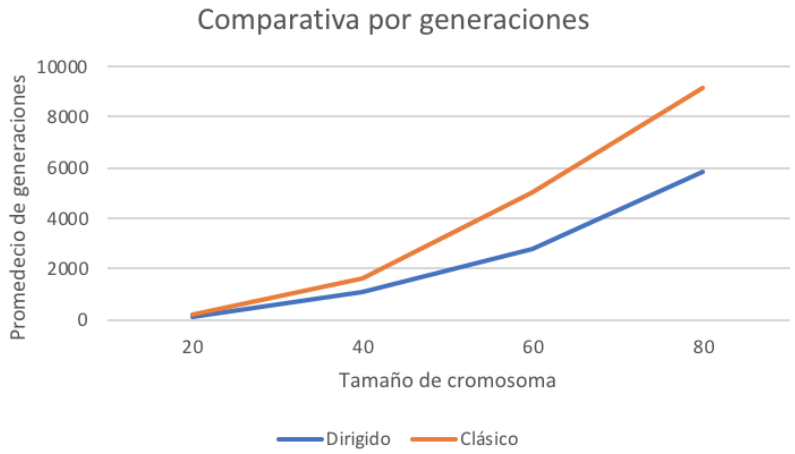

Figura 3. Resultados al comparar las dos implementaciones por número medio de generaciones necesarias para obtener una solución

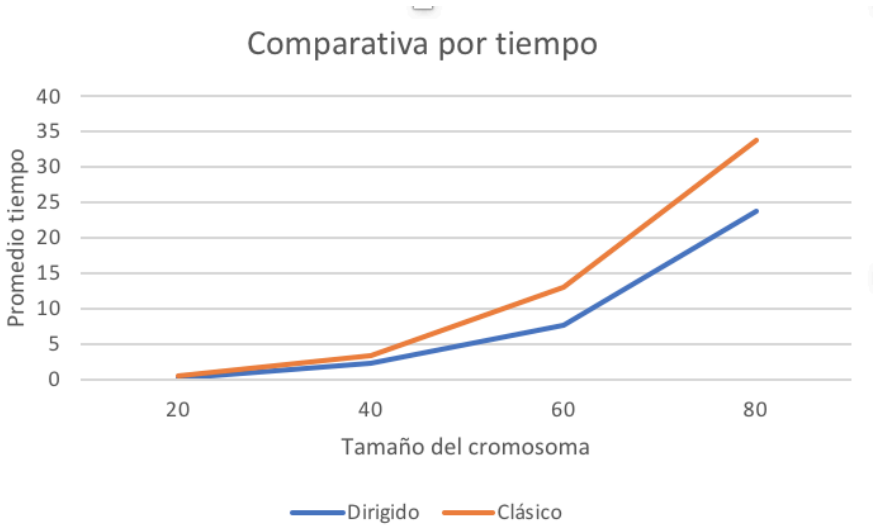

Figura 4. Resultados obtenidos al comparar las dos implementaciones por el tiempo medio necesario para obtener una solución.

Fondo Europeo de Desarrollo Regional, "Una manera de hacer Europa".

Referencias

[1] Srinivas, M., Patnaik, L. M. *"Adaptive probabilities of crossover and mutation in genetic algorithms,"* IEEE Transactions on Systems, Man, and Cybernetics, 24(4), 656-667, 1994.

[2] Zhang, J., Chung, H. S. H., Lo, W. L. *"Clustering-based adaptive crossover and mutation probabilities for genetic algorithms,"* IEEE Transactions on Evolutionary Computation, 11(3), 326-335, 2007

[3] Fogel, D. B., Atmar, J. W. *"Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems,"* Biological Cybernetics, 63(2), 111-114, 1990.

[4] Lee, C. Y., Yao, X. *"Evolutionary programming using mutations based on the Lévy probability distribution,"* IEEE Transactions on Evolutionary Computation, 8(1), 1-13, 2004

[5] Back, T. *"Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms,"* Oxford university press. 1996

[6] Goldberg, D. E., Holland, J. H. *"Genetic algorithms and machine learning. Machine learning,"* 3(2), 95-99, 1988.

[7] Bhandari, D., Pal, N. R., Pal, S. K. *"Directed mutation in genetic algorithms,"* Information Sciences, 79(3-4), 251-270. 1994

[8] Bäck, T. *"Self-adaptation in genetic algorithms,"* In Proceedings of the first european conference on artificial life, 263-271, MIT Press. 1992

[9] Michalewicz, Z. *"Evolution strategies and other methods.,"* In Genetic algorithms+ data structures= evolution programs (pp. 159-177). Springer, Berlin, Heidelberg. 2016

Choosing population sizes to enhance Brain Storm Optimization algorithms

Ricardo García-Ródenas, Luis Jiménez Linares, and Julio Alberto López-Gómez
{*Ricardo.Garcia, Luis.Jimenez, JulioAlberto.Lopez*}*@uclm.es*

***Abstract*—Nature-inspired algorithms (NIA) are a very powerful tool to solve plenty of complex science and engineering problems. In the last ten years, a new kind of NIA algorithms, so-called human-inspired algorithms, has arisen in optimization tasks obtaining promising results. In this paper, the effect of population size in Brain Storm Optimization algorithm (BSO) is studied with the purpose of choosing good enough population sizes which improve its performance. Moreover, it is also analyzed the possibility of using population samples instead of the whole population, studying its performance in terms of time and computational cost. To do that, hybrid functions of IEEE CEC competitions have been used as benchmark problems. The results show that twenty five individuals is a good enough population size in BSO algorithms while population sampling improves the performance of the algorithm.**

***Index Terms*—Nature-inspired algorithms, Human-inspired algorithms, Brain storm optimization, Population size, Population sampling**

I. Introduction

Traditionally, complex optimization problems have required effective and sophisticated methods to deal with them. In this context, Nature-Inspired Algorithms (NIA) appeared in the 1960s. One of the main features of NIA algorithms is they take their knowledge, contents and procedures departing from nature. Thus, algorithms like Particle Swarm Optimization (PSO) [1], Genetic Algorithm (GA) [2] or Ant Colony Optimization (ACO) [3] among others, have been widely applied in operational research.

During the last ten years, there has been a change in the source of inspiration to build NIA algorithms. Nowadays there is a trend which tries to model the behaviour of humans in problem solving. Along this line, a new kind of NIA algorithms has appeared which is called Human-Inspired Algorithms (HIA). There are currently six HIA algorithms which are the following: Seeker Optimization Algorithm (SOA) [4], Imperialist Competitive Algorithm (ICA) [5] , Social Emotional Optimization Algorithm (SEOA) [6], Teaching Learning-Based Optimization (TLBO) [7], Team Effectiveness Based Optimization (TEBO) [8] and Brain Storm Optimization algorithm (BSO) [9] which is the algorithm used in this paper.

This paper has two targets: On the one hand, it tries to choose good enough population sizes in Brain Storm Optimization algorithm when it is applied to hybrid functions. On the other hand, it studies whether is better to use the whole population in the algorithm or only a sample of it, in terms of computational cost and quality of solutions.

The rest of the document is structured as follows: section 2 studies the related work in BSO algorithm. Section 3 describes BSO algorithm and ADMBSO algorithm, which is the version used in the experiments of this paper. Later, section 4 defines the experiments carried out and studies the results obtained and finally, section 5 introduces the conclusions about this work and some future works.

Ricardo García-Ródenas, Department of Mathematics at University of Castilla la Mancha, Spain

Luis Jiménez Linares, Department of Technology and Information Systems of University of Castilla la Mancha, Spain

Julio Alberto López Gómez, Department of Technology and Information Systems at University of Castilla la Mancha, Spain.

II. Related Work

To the best of our knowledge, there are 75 papers, 8 theses and 5 patents in total on the development and application of the BSO algorithm. The current research in BSO is concerned with three main research lines which are the following:

- **Study and analysis of the algorithm.** In this line, all studies about new brainstorming operators [10], modifications [11], clustering methods [12], parameters adjustment [13], hybridizations [14], etc are grouped.
- **Algorithm's variants.** New variants of brain storm algorithms are required to solve successfully different complex optimization problems like multimodal [15], hybrid or multiobjective problems [16], among others.
- **BSO applications.** There is a lack of BSO applications to real world problems, although there are many applications in fields like wireless sensor networks [17], multiple UAV formation flights [18], stock index forecasting [19] and economic dispatch [20].

III. Brain Storm Optimization Algorithms

Brain Storm Optimization (BSO) algorithm is a human-inspired algorithm which was born in 2011 and developed by Shi [9]. This NIA algorithm is inspired by the human brainstorming process. In it, a group of people or brainstorming group proposes different ideas to solve a problem. Then, new ideas are generated from the existing ones in order to reach the optimum idea. The brainstorming process ends when an optimum idea has been achieved.

This way, brainstorming departs from a set of random initial solutions. After initialization, new ideas are generated departing from the existing ones through convergence and divergence operations. Convergence operation is focused on searching new solutions in a set of search areas where good solutions are found. Formally, these areas corresponds with

local minima in the objective function. To carry out convergence operation, cluster analysis techniques have been widely applied, using mainly the $k-$means algorithm. Moreover, divergence operation is in charge of building new solutions. In brainstorming, new solutions can be generated departing from one or more existing solutions. For simplicity, BSO algorithm considers only two possibilities: generating a new solution departing from one existing solution or two. The first option corresponds with local search in an optimization algorithm and the second is related to exploration. The kind of generation will be made using a set of probabilities. After BSO, different variants of the algorithm have appeared with the purpose of improving BSO capabilities and guaranteeing a good tradeoff between exploration and exploitation. In this paper, Advanced Discussion Mechanism based on Brain Storm Optimization Algoritm (ADMBSO) [11] is used instead of the original BSO. ADMBSO has been used since it is the most recent and standard version of BSO algorithm. In it, exploration is encouraged in the first iterations and exploitation in the last iterations. Moreover, ADMBSO adds different rules to generate new ideas in each epoch in intra and inter cluster generation. On the one hand, inter cluster generation will be possible in this algorithm building a new idea from two ideas in the same cluster. Briefly, this algorithm is shown in Algorithm 1.

Algorithm 1 Advanced Discussion Mechanism based on BSO algorithm

1: POPULATION INITIALIZATION: Initialize n random solutions in population pop.
2: **while** Termination Condition is not satisfied **do**
3: $P_{intra} = P_{low} + P_{high}\frac{N_{cur}gen}{N_{max}gen}$
4: $P_{inter} = 1 - P_{intra}$
5: /* Update ϵ */
6: $\epsilon(t) = logsig\left(\frac{\frac{max_{iter}}{2} - n_{iter}}{k}\right) * random(t)$
7: POPULATION EVOLUTION: Generate n new ideas departing from the current population
8: **BEGIN Converging Operation**
9: Group pop in m clusters
10: **END Converging Operation**
11: **for** $i = 1$ to $card(pop)$ **do**
12: **BEGIN Diverging Operation**
13: **if** $rand() < P_{intra}$ **then**
14: /* Generate individuals from one cluster*/
15: **if** $rand() < p_{ceni}$ **then**
16: /* Add noise to center of the cluster */
17: $c^i_{new} = c^i_{old} + \epsilon(t) * random(t)$
18: **else if** $rand() < p_{indi}$ **then**
19: /*Combine two ideas from this cluster and add noise */
20: Select x_{old1} and x_{old2}
21: $x^i_{new} = (x^i_{old1} - x^i_{old2}) * random(t)$
22: **else**
23: /*Add noise to a random idea*/
24: $x^i_{new} = x^i_{old} + \epsilon(t) * random(t)$
25: **end if**
26: **else**
27: /*Generate individuals from two clusters*/
28: **if** $rand() < p_{cenii}$ **then**
29: /*Select two centroids c_{old1}, c_{old2}, combine them and add noise/*
30: $x^i_{old} = w_1 * c^i_{old1} + w_2 * c^i_{old2}$
31: $x^i_{new} = (c^i_{old1} - c^i_{old2}) * random(t)$
32: **else if** $rand() < p_{indii}$ **then**
33: /*Generate randomly an idea depart from two individuals of two clusters/*
34: Select x_{old1} and x_{old2}
35: $x^i_{old} = w_1 * x^i_{old1} + w_2 * x^i_{old2}$
36: $x^i_{new} = (x^i_{old1} - x^i_{old2}) * random(t)$
37: **else**
38: /* Generate randomly a new idea */
39: **end if**
40: **end if**
41: **END Diverging Operation**
42: REPLACE MECHANISM: If the new solution built is better than the current one, substitute it, otherwise, maintain the current solution
43: /* Update Solution */
44: **if** $f(x_{new}) < f(x_{old})$ **then**
45: $x_{old} = x_{new}$
46: **end if**
47: **end for**
48: **end while**

IV. Experiments

In this section, the two experiments carried out to choose good enough population sizes in Advanced BSO algorithm (ADMBSO) are described in detail.

A. *Experiment* 1*: Choosing population size in BSO*

The problem of choosing a good enough population size for a NIA algorithm is difficult to address. The reason is because it is generally problem-dependent and in general, the settings of one algorithmic parameter could be related to the settings of other algorithmic parameters, in terms of optimization performance. Besides, it depends on the particular version of BSO used, the dimensionality problem and the complexity of the search space. That is why, in this experiment, different population sizes widely applied in other algorithms have been tested over ADMBSO algorithm and over a set of concrete benchmark functions, in this case, hybrid functions from IEEE CEC conference competition.

In order to recommend good population sizes for BSO algorithms, the definition of robustness analysis proposed in [21] is taken. According to that, an optimization algorithm is robust if inequality

$$|f(x^*_{alg}) - f(x^*)| < \epsilon_{rel} \cdot |f(x^*)| + \epsilon_{abs} \qquad (1)$$

holds, where x^*_{alg} is the best value reached by the algorithm, x^* is a global minimum of the objective function f (it is assumed, given) and ϵ_{rel}, ϵ_{abs} are two parameters which control the accuracy of the algorithm.

Thus, different population sizes can be proposed and tested with different values of ϵ_{abs} in order to detect if different

population sizes are capable of reaching determined levels of accuracy. Moreover, the convergence speeds of the same algorithm with different population sizes can also be compared using the expression proposed in [21] and shown in equation 2:

$$s = \frac{NFE_{pop1}}{NFE_{pop2}} \quad (2)$$

where s is the convergence speed ratio, NFE_{pop1} is the number of functions evaluations which the algorithm with the population size $pop1$ needs to satisfy inequality (1) and NFE_{pop2} is the same using $pop2$. To do that, three ϵ_{abs} values have been proposed in order to test the performance and robustness of different population sizes while ϵ_{rel} has been fixed to $10e^{-4}$ like in [21]. These values are $\epsilon_{abs1} = 10e^{-1}, \epsilon_{abs2} = 10e^{-3}, \epsilon_{abs3} = 10e^{-5}$

Moreover, four population sizes have been tried in this experiment. Firstly, one hundred ideas has been taken as a reference value, since it is used in most of the articles published in BSO and cited here. After that, and following the fact that lower values are efficient in other metaheuristics like in PSO, other population sizes have been proposed. Thus, pop_{25} has twenty five ideas or individuals, pop_{50} has fifty ideas, pop_{75} has seventy five individuals and pop_{100} has one hundred ideas. The sample of benchmark functions consists of eight objective functions with thirty dimensions taken from IEEE CEC conference competitions. They are well-known optimization problems. In this paper, we have focused on hybrid functions, which are the most complex benchmarks in CEC competition. They are the following: rotated hybrid composition function 1, rotated hybrid composition function 1 with noise in fitness, rotated hybrid composition 2, rotated hybrid composition 2 with a narrow basin for the global optimum, rotated hybrid composition 3, rotated hybrid composition 3 with high condition number matrix, rotated hybrid composition function 4 and rotated hybrid composition function 4 without bounds. These problems are named as P1,P2,...,P8.

Then, ADMBSO algorithm has been executed for each population size using the parameters employed in [11]. The configuration is the following: $m = 5, P_{cen} = 0.7, P_{ind} = 0.2, P_{rnd} = 0.1, P_{cens} = 0.7, P_{low} = 0.2, P_{high} = 0.7$. The total number of function evaluations computed in each execution is 100000. Ten trials have been run for each population size in order to minimize the random effect in the experiment.

Average values of mean, maximum, minimum, variance and standard deviation have been obtained for each population size. The total time spent in all executions is reported too. These results are shown in table I. The minimum values for each measure are typed in bold. Thus, it is possible to see that pop_{25} needs less time than the rest of populations to achieve its results and it is the population which reaches the minimum values regarding the rest of population sizes in more test functions. Moreover, figure 1 shows the temporal cost of each population size for each problem. This way, it is possible to see that pop_{25} provides the best results regarding quality of solution and temporal cost.

Furthermore, regarding robustness analysis, table II shows the convergence speed of different population sizes in comparison with the population of one hundred individuals (used in [11] and considered as reference). Note that problems 3 and 4 do not appear in this table because they did not achieve any robustness threshold. In this manner, when s values are less than 1, it means that the proposed configuration is better than pop_{100} taken as reference. If it is greater than 1, the performance of pop_{100} is better. The best rates are typed in bold. It is possible to conclude that all the sizes tested in this paper are better than p_{100} used in the most of articles published about BSO.

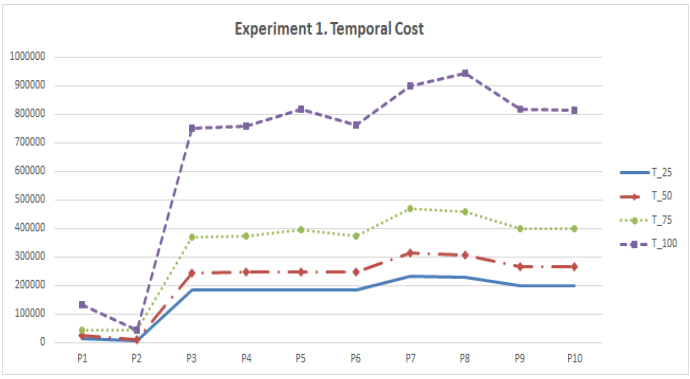

Fig. 1. Temporal Cost for different population sizes

TABLE II
Analysis of convergence speed in ADMBSO algorithm

	s1	s2	s3
P1_25	**0,86**	-	-
P1_50	0,90	-	-
P1_75	0,93	-	-
P2_25	-	-	-
P2_50	**0,96**	-	-
P2_75	0,98	-	-
P5_25	**0,56**	-	-
P5_50	0,75	-	-
P5_75	0,99	-	-
P6_25	**0,59**	-	-
P6_50	1,04	-	-
P6_75	0,77	-	-
P7_25	**0,72**	0,95	
P7_50	0,82	1,01	1,01
P7_75	1,09	**0,89**	**0,90**
P8_25	1,05	1,19	1,17
P8_50	**0,92**	**0,64**	**0,63**
P8_75	1,01	0,96	0,94

B. Experiment 2: Population sampling in BSO

One solution to improve the quality of population could be to choose a sample of the population in the search process. It would imply a better initial population but include the problematic of choosing a good sample which combines good fitness values in the individuals chosen and good ideas diversity. Besides, the sampling process adds computational cost to the algorithm.

To do that, a multinomial probability distribution can be established according to the fitness values. This way, it will determine the probability of choosing an individual in the population according the fitness values of the whole population.

TABLE I
ADMBSO STATISTICAL RESULTS FOR HYBRID CEC BENCHMARK FUNCTIONS

	Mean	Best	Worst	Variance	Std	Time
P1_25	**2,93E+02**	2,08E+02	1,55E+03	1,79E+04	1,33E+02	**1,82E+05**
P1_50	2,96E+02	2,21E+02	**1,10E+03**	**1,28E+04**	**1,13E+02**	2,44E+05
P1_75	3,57E+02	**1,66E+02**	1,06E+03	3,86E+04	1,96E+02	3,68E+05
P1_100	3,38E+02	2,21E+02	1,123E+03	1,54E+04	1,24E+02	7,50E+05
P2_25	3,94E+02	2,81E+02	1,54E+03	3,87E+04	1,96E+02	**1,84E+05**
P2_50	3,74E+02	2,40E+02	**1,21E+03**	3,68E+04	1,91E+02	2,46E+05
P2_75	3,76E+02	2,38E+02	1,54E+03	2,93E+04	1,71E+02	3,72E+05
P2_100	**3,58E+02**	**2,21E+02**	1,39E+03	**2,68E+04**	**1,64E+02**	7,59E+05
P3_25	9,01E+02	9,00E+02	1,45E+03	3,17E+02	1,7E+01	**1,84E+05**
P3_50	9,00E+02	9,00E+02	1,02E+03	**5,70E+01**	**7,59E+00**	2,46E+05
P3_75	8,74E+02	8,57E+02	1,41E+03	1,98E+03	4,45E+01	3,93E+05
P3_100	**8,68E+02**	**8,53E+02**	**9,06E+02**	3,21E+01	1,79E+01	8,17E+05
P4_25	9,00E+02	9,00E+02	**9,18E+02**	**5,39E+00**	**2,32E+00**	**1,84E+05**
P4_50	8,78E+02	8,66E+02	1,17E+03	2,73E+02	1,65E+01	2,46E+05
P4_75	**8,63E+02**	**8,47E+02**	1,39E+03	2,39E+02	4,89E+01	3,74E+05
P4_100	8,82E+02	8,66E+02	1,43E+03	8,33E+02	2,88E+01	7,63E+05
P5_25	**7,11E+02**	5,15E+02	1,48E+03	7,75E+ 04	2,78E+02	**2,33E+05**
P5_50	7,43E+02	**5,03E+02**	1,54E+03	8,17E+04	2,85E+02	3,12E+05
P5_75	8,91E+02	5,99E+02	**1,44E+03**	8,48E+04	2,91E+02	4,70E+05
P5_100	7,93E+02	5,33E+02	1,48E+03	**6,72E+04**	**2,59E+02**	8,99E+05
P6_25	**6,64E+02**	**5,38E+02**	2,48E+03	6,65E+04	2,57E+02	**2,29E+05**
P6_50	8,14E+02	6,69E+02	2,08E+03	**5,42E+04**	**2,32E+02**	3.07E+05
P6_75	7,54E+02	5,83E+02	1,91E+03	6,97E+04	2,64E+02	4,60E+05
P6_100	7,93E+02	6,07E+02	**1,87E+03**	6,90E+04	2,62E+02	9,42E+05
P7_25	**4,20E+02**	**2,29E+02**	1,57E+03	**1,37E+05**	**3,70E+02**	**1,97E+05**
P7_50	4,52E+02	2,40E+02	1,53E+03	1,57E+05	3,96E+02	2,64E+05
P7_75	5,25E+02	2,39E+02	1,52E+03	1,97E+05	4,44E+02	3,99E+05
P7_100	4,90E+02	2,39E+02	**1,52E+03**	1,72E+05	4,15E+02	8,17E+05
P8_25	4,12E+02	2,31E+02	1,65E+03	1,22E+05	3,49E+02	**1,98E+05**
P8_50	**4,03E+02**	**2,30E+02**	**1,48E+02**	1,31E+05	3,62E+02	2,66E+05
P8_75	4,17E+02	2,37E+02	1,51E+03	**1,15E+05**	**3,40E+02**	4,00E+05
P8_100	4,35E+02	2,44E+02	1,525E+03	1,28E+05	3,58E+02	8,15E+05

Thus, the probability of choosing an individual i is computed through equation.

$$P(i) = \frac{\frac{1}{f(i)}}{\sum_{j=1}^{n} \frac{1}{f(j)} + \epsilon} \quad (3)$$

where n is the population size, $f(i)$ is the fitness value for individual i and ϵ is a value near to zero to avoid to divide between 0. In previous experiment pop_{25} has been chosen as the best option in terms of population size. Now, ADMBSO algorithm has been executed using a population of one hundred individuals, but in this case, only twenty five individuals are selected to execute the algorithm. The selection mechanism is based on the multinomial probability distribution.

Now, ADMBSO algorithm with a sampled population of twenty five individuals have been executed under the same conditions as Experiment 1. Mean values of mean, minimum, maximum, variance and standard deviation as well as total time (in seconds) have been computed. These results are shown in table III. The results which improve the original results of pop_{25} are typed in bold.

With regard to the results, it can be noted this way of sampling population improves the original results of pop_{25}. Concretely, the major improvements are made in the last two problems which corresponds with the most difficult in CEC competitions (where pop_{25} did not improve the results of the rest of population sizes) and it is now the best configuration. Furthermore, it can be seen that the time is not improved in this case. To show that, figure 2 shows the temporal cost of the population sizes made in experiment 1 in comparison with the temporal cost of population sampling.

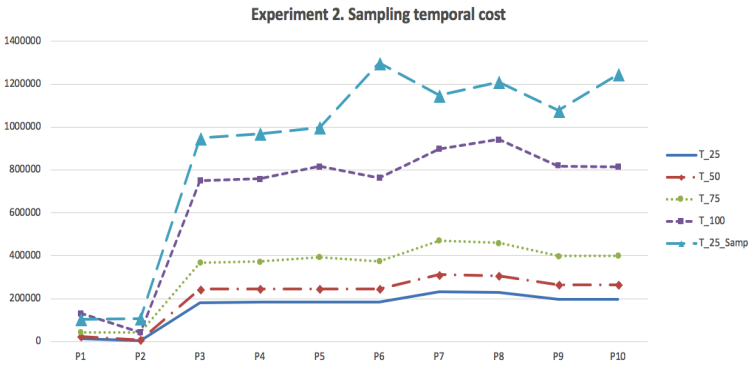

Fig. 2. Temporal Cost with original population sizes and sampling

Regarding robustness analysis, table IV shows the convergence speed (s) of sampling population with respect to pop_{100} which was considered as reference. Once again, values which are equals or better than pop_{25} convergence speeds are typed in bold. The results show that sampling population increase the convergence speed of the algorithm in comparison with the original version.

V. Conclusions and further works

In this article the problem of choosing good enough population sizes in BSO algorithms, has been carried out. To do

TABLE III
STATISTICAL RESULST OF POPULATION SAMPLING

	Mean	Best	Worst	Variance	Std	Time
P1_25_Sam	**2,91E+02**	2,27E+02	**9,11E+02**	**1,42E+04**	**1,19E+02**	9,49E+05
P2_25_Sam	**3,63E+02**	**2,81E+02**	**1,12E+02**	**1,46E+04**	**1,20E+02**	9,68E+05
P3_25_Sam	**9,00E+02**	**9,00E+02**	**1,29E+03**	**2,44E+02**	**1,56E+01**	9,97E+05
P4_25_Sam	**9,00E+02**	**9,00E+02**	9,84E+01	7,51E+00	2,74E+00	1,29E+06
P5_25_Sam	7,14E+02	5,28E+02	1,56E+02	7,897E+04	2,81E+02	1,14E+06
P6_25_Sam	8,17E+02	6,78E+02	**1,76E+03**	**4,51E+02**	**2,12E+02**	1,21E+06
P7_25_Sam	**3,76E+02**	**2,26E+01**	1,64E+03	**9,26E+04**	**3,04E+02**	1,07E+06
P8_25_Sam	**3,82E+02**	**2,63E+01**	**1,604E+02**	**9,84E+04**	**3,13E+02**	1,24E+05

TABLE IV
CONVERGENCE SPEED OF SAMPLING POPULATION

	s1	s2	s3
P1_Sam	**0,55**	0,88	**0,86**
P2_Sam	**0**	-	-
P3_Sam	0,87	-	-
P4_Sam	-	-	-
P7_Sam	**0,53**	**-**	**-**
P8_Sam	**0,95**	**-**	**-**
P9_Sam	**0,70**	**0,66**	**0,66**
P10_Sam	**0,85**	**0,66**	**0,65**

that, different population sizes have been defined in order to study the solutions achieved and the time spent for each size. After that, the best population size has been chosen and an strategy of sampling population has been carried out in order to study if this strategy improves the results of the original algorithm. The main conclusions reached are the following:

- Despite the fact that one hundred individuals have been widely used in the majority of articles published about BSO algorithms, it has been demonstrated that less individuals guarantee better results in terms of quality of solution in hybrid functions.
- Sampling population is a good solution to enhance the results of the original algorithm in terms of solutions achieved in the case of hybrid functions.
- Sampling population strategy speeds up the convergence speed of ADMBSO algorithm when it is applied over hybrid functions.

As future works, these experiments can be applied over different benchmark functions and new sampling strategies can be studied in order to reduce the temporal cost of the algorithm, as well as speeding up NIA algorithms through local search procedures in order to guarantee better robustness thresholds.

ACKNOWLEDGMENT

This research was supported by *Ministerio de Economía, Industria y Competitividad*-FEDER EU grants with number TRA2016-76914-C3-2-P

REFERENCES

[1] Y. Shi and R. C. Eberhart, "A Modified Particle Swarm Optimizer," in *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE Computer Society, May 1998, pp. 69–73.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[3] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B*, vol. 26, no. 1, pp. 29–41, 1996.

[4] C. Dai, Y. Zhu, and W. Chen, "Seeker optimization algorithm," in *Computational Intelligence and Security*, Y. Wang, Y.-m. Cheung, and H. Liu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 167–176.

[5] E. Atashpaz-Gargari and C. Lucas, "Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition," *2007 IEEE Congress on Evolutionary Computation*, pp. 4661–4667, 2007.

[6] Z. Cui, Z. Shi, and J. Zeng, "Using social emotional optimization algorithm to direct orbits of chaotic systems," in *Swarm, Evolutionary, and Memetic Computing*, B. K. Panigrahi, S. Das, P. N. Suganthan, and S. S. Dash, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 389–395.

[7] R. Rao, V. Savsani, and D. Vakharia, "Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems," *Computer-Aided Design*, vol. 43, no. 3, pp. 303 – 315, 2011.

[8] X. Feng, M. Ji, Z. Li, X. Qu, and B. Liu, "Team effectiveness based optimization," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, June 2017, pp. 2248–2257.

[9] Y. Shi, "Brain storm optimization algorithm," in *Advances in Swarm Intelligence*, Y. Tan, Y. Shi, Y. Chai, and G. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 303–309.

[10] D. Zhou, Y. Shi, and S. Cheng, "Brain storm optimization algorithm with modified step-size and individual generation," in *Advances in Swarm Intelligence*, Y. Tan, Y. Shi, and Z. Ji, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 243–252.

[11] Y. Yang, Y. Shi, and S. Xia, "Advanced discussion mechanism-based brain storm optimization algorithm," vol. 19, 10 2015.

[12] J. Chen, Y. Xie, and J. Ni, "Brain storm optimization model based on uncertainty information," in *2014 Tenth International Conference on Computational Intelligence and Security*, Nov 2014, pp. 99–103.

[13] Z. Zhan, W. Chen, Y. Lin, Y. Gong, Y. Li, and J. Zhang, "Parameter investigation in brain storm optimization," in *2013 IEEE Symposium on Swarm Intelligence (SIS)*, April 2013, pp. 103–110.

[14] R. García-Ródenas, L. J. Linares, and J. A. López-Gómez, "A cooperative brain storm optimization algorithm," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, June 2017, pp. 838–845.

[15] X. Guo, Y. Wu, and L. Xie, "Modified brain storm optimization algorithm for multimodal optimization," in *Advances in Swarm Intelligence*, Y. Tan, Y. Shi, and C. A. C. Coello, Eds. Cham: Springer International Publishing, 2014, pp. 340–351.

[16] X. Guo, Y. Wu, L. Xie, S. Cheng, and J. Xin, "An adaptive brain storm optimization algorithm for multiobjective optimization problems," in *Advances in Swarm and Computational Intelligence*, Y. Tan, Y. Shi, F. Buarque, A. Gelbukh, S. Das, and A. Engelbrecht, Eds. Cham: Springer International Publishing, 2015, pp. 365–372.

[17] R. Ramadan and A. Khedr, "Brain storming algorithm for coverage and connectivity problem in wireless sensor network," 04 2016.

[18] H. Qiu, H. Duan, and Y. Shi, "A decoupling receding horizon search approach to agent routing and optical sensor tasking based on brain storm optimization," *Optik - International Journal for Light and Electron Optics*, vol. 126, no. 7, pp. 690 – 696, 2015.

[19] J. Wang, R. Hou, C. Wang, and L. Shen, "Improved v -support vector regression model based on variable selection and brain storm optimization for stock price forecasting," *Applied Soft Computing*, vol. 49, pp. 164 – 178, 2016.

[20] K. R. Ramanand, K. R. Krishnanand, B. K. Panigrahi, and M. K. Mallick, "Brain storming incorporated teaching–learning–based algorithm with application to electric power dispatch," in *Swarm, Evolutionary, and Memetic Computing*, B. K. Panigrahi, S. Das, P. N. Suganthan, and P. K. Nanda, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 476–483.

[21] F. Kang, J. Li, and Z. Ma, "Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions," *Information Sciences*, vol. 181, no. 16, pp. 3508 – 3531, 2011.

Distance-based Exponential Probability Models for Constrained Combinatorial Problems*

*Note: The full contents of this paper have been published in the volume *Lecture Notes in Artificial Intelligence 11160* (LNAI 11160)

Josu Ceberio, Alexander Mendiburu
University of the Basque Country UPV/EHU
Donostia, Spain
josu.ceberio@ehu.eus

Jose A. Lozano
University of the Basque Country UPV/EHU
Basque Center for Applied Mathematics (BCAM)
Bilbao, Spain

***Abstract*—Estimation of Distribution Algorithms (EDAs) have already demonstrated their utility when solving a broad range of combinatorial problems. However, there is still room for methodological improvement when approaching problems with constraints. The great majority of works in the literature implement repairing or penalty schemes, or use ad-hoc sampling methods in order to guarantee the feasibility of solutions. In any of the previous cases, the behavior of the EDA is somehow denaturalized, since the sampled set does not follow the probability distribution estimated at that step. In this work, we present a general method to approach constrained combinatorial optimization problems by means of EDAs. This consists of developing distance-based exponential probability models defined exclusively on the set of feasible solutions. In order to illustrate this procedure, we take the 2-partition balanced Graph Partitioning Problem as a case of study, and design efficient learning and sampling methods to use distance-based exponential probability models in EDAs.**

***Index Terms*—Constraint, estimation of distribution algorithm, distance-based exponential model, Graph Partitioning Problem**