

Resolución del cuadrado mágico con algoritmos genéticos

Jorge Casajús Setién
Luis Galocha Domínguez
Carlos Morencos Sánchez



POLITÉCNICA

UNIVERSIDAD
POLITÉCNICA
DE MADRID

¿Qué es un cuadrado mágico?

Un cuadrado mágico es un **cuadrado de orden n** en el que cada celda contiene un número del 1 al n^2 de forma que los elementos de toda fila, columna y diagonal principal suman el mismo número, la **constante mágica**.

8	1	6
3	5	7
4	9	2

Ejemplo de cuadrado mágico de orden 3

Constante mágica

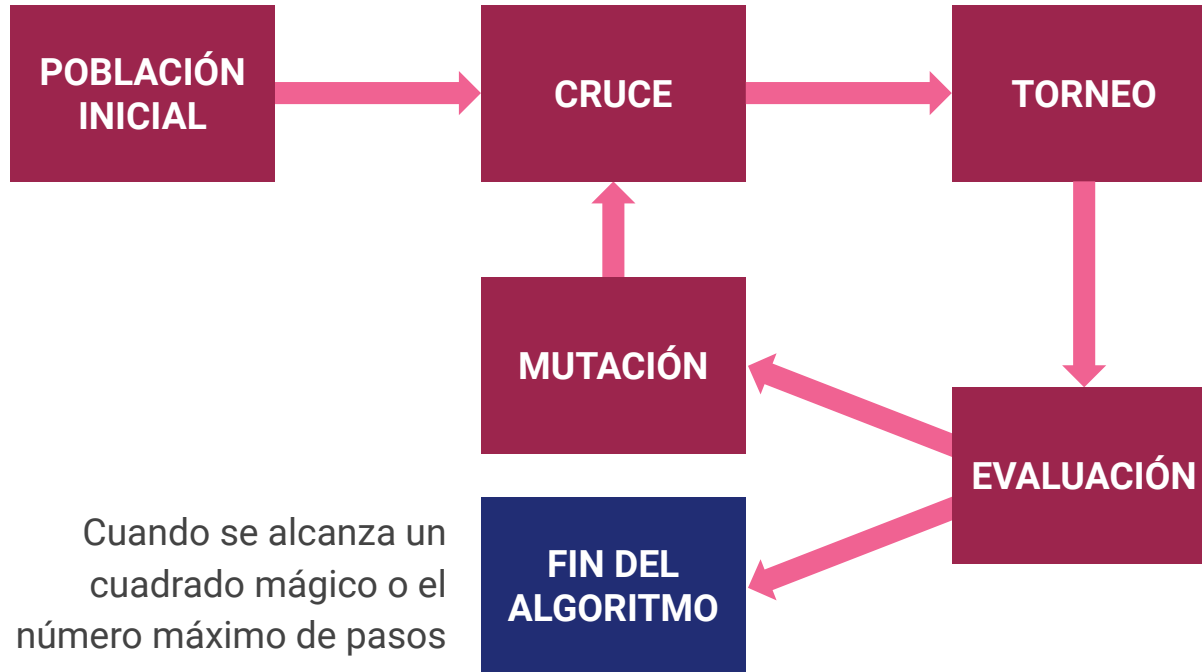
$$m = \frac{n \cdot (n^2 + 1)}{2}$$

Para 4x4: 7040 cuadrados mágicos de $2 \cdot 10^{13}$

Objetivos de la práctica

- **Objetivo principal:** Implementar un **algoritmo genético** para alcanzar una solución del problema del **cuadrado mágico** de orden 4
- **Analizar** el comportamiento del algoritmo en función de sus **parámetros**
- Resolver el problema para **cuadrados de orden mayor**

Estructura del algoritmo utilizado



Algoritmo genético | Codificación de cromosomas

Cada **elemento** del cuadrado candidato corresponde a un **gen codificado como un número entero**.

Un **cromosoma** es el conjunto ordenado de números que componen el cuadrado candidato y se puede representar de dos modos:


- **Matricialmente:** Es el cuadrado transcrito a una matriz, resulta conveniente para aplicar la función evaluación sobre el cromosoma
- **Vectorialmente:** Es la matriz desarrollada en un vector, conveniente para aplicar el cruce y la mutación sobre el cromosoma

Algoritmo genético | Función de coste

La función de coste establece cómo se **evalúa la bondad de las soluciones**.

Es importante definir una buena función de coste ya que de ella dependerá en gran parte que el algoritmo resuelva el problema y lo haga eficientemente.

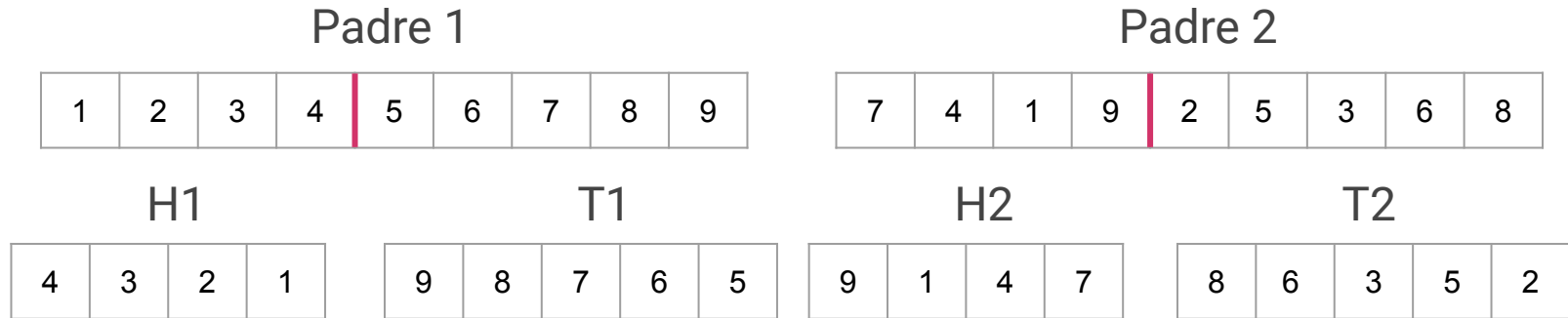
Para este trabajo se ha utilizado una función de coste basada en la **desviación de las sumas** con respecto de la constante mágica m :

$$f = \left(\sum_{i=1}^4 |C_i - m| \right) + \left(\sum_{i=1}^4 |R_i - m| \right) + \left(\sum_{i=1}^2 |D_i - m| \right) \quad \text{Elhaddad and Alshaari (2014)}$$


Diferencia de las columnas Diferencia de las filas Diferencia de las diagonales

Algoritmo genético | Operador de cruce SIC 1 punto

Se trabaja con los cromosomas como vectores. Se escoge un punto de corte al azar **dividiendo a los padres en dos partes**: cabeza y cola, cuyo orden se invierte. Visualmente:



Algoritmo genético | Operador de cruce SIC 1 punto

Se procede eliminando cada fragmento del padre contrario:



Algoritmo genético | Operador de cruce SIC 1 punto



Algoritmo genético | Operador de cruce SIC 1 punto

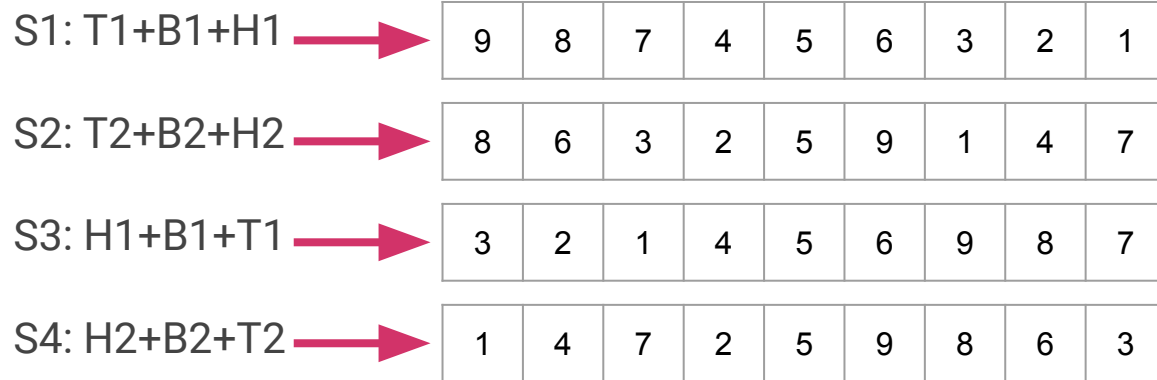
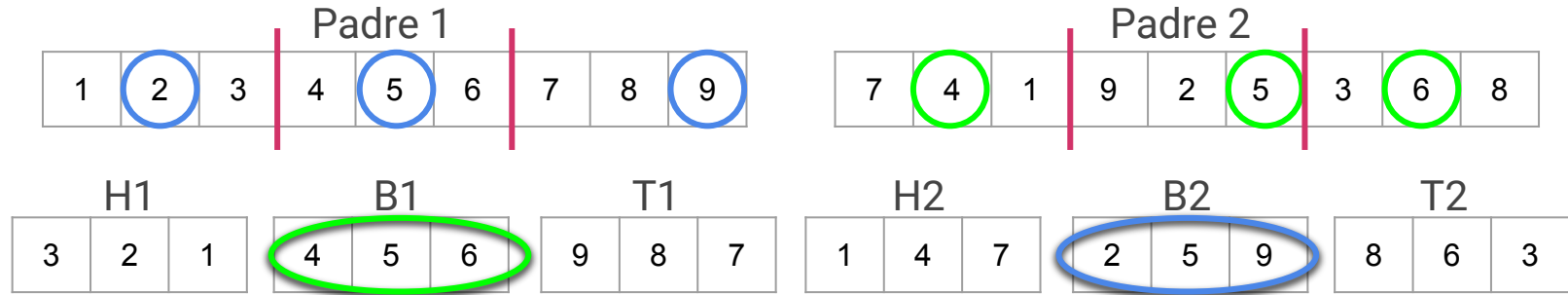


Algoritmo genético | Operador de cruce SIC 2 puntos

Este operador es similar al de 1 punto pero esta vez se escogen aleatoriamente dos puntos de corte de forma que quedan 3 fragmentos de cada padre.



Algoritmo genético | Operador de cruce SIC 2 puntos



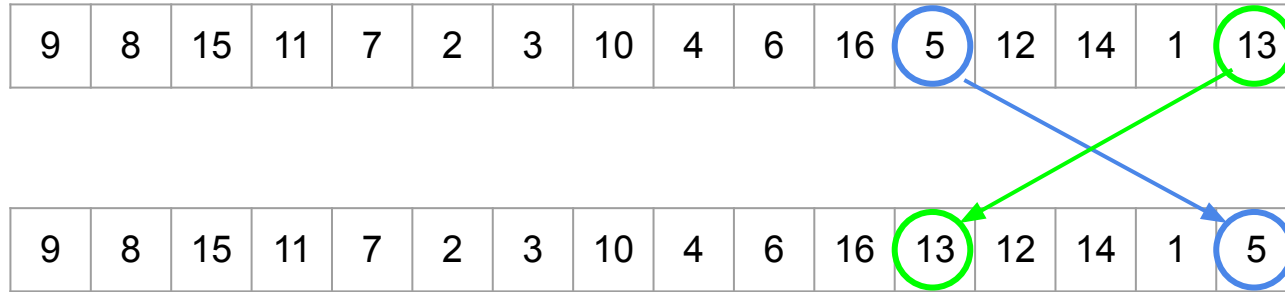
Algoritmo genético | Operadores de mutación

Se han implementado **dos operadores** de mutación, actuando cada uno con una probabilidad del 50% (en caso de suceder una mutación)

- **Mutación simple:** se intercambian de posición dos genes seleccionados al azar
- **Mutación cuádruple:** se intercambian de posición dos cadenas de 4 genes escogidas aleatoriamente

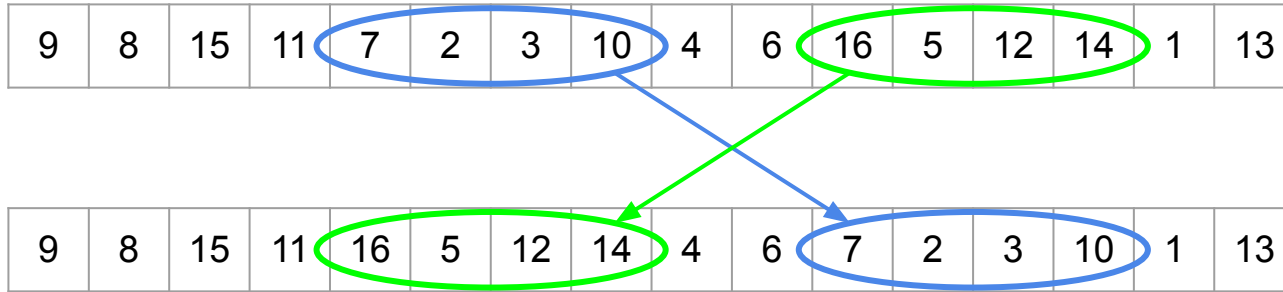
Algoritmo genético | Operadores de mutación

Ejemplo del operador de **mutación simple**



Algoritmo genético | Operadores de mutación

Ejemplo del operador de **mutación cuádruple**



Algoritmo genético | Torneo

Para mantener la población en cada generación se deben **escoger los individuos más aptos**.

Se ha optado por realizar una **selección por torneo**. Del conjunto de los hijos y los padres se realiza una competición con eliminatoria **enfrentándolos por parejas** hasta que solo queda un número de finalistas igual a la población inicial.

Algoritmo genético | Torneo

Para mantener que la **selección por torneo** funcione correctamente:

- La población a mantener debe ser una **potencia de 2**
- La **mutación** debe cobrar un papel relevante para compensar que la selección por torneo es muy intensiva en la **explotación de soluciones**.

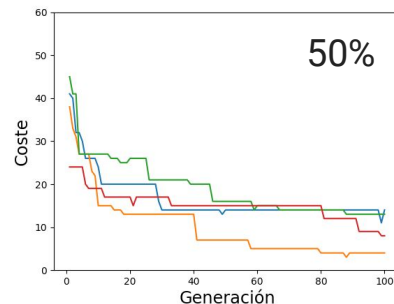
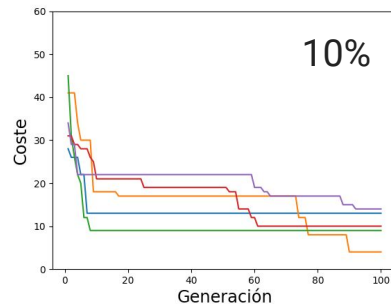
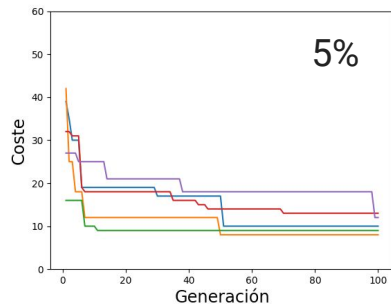
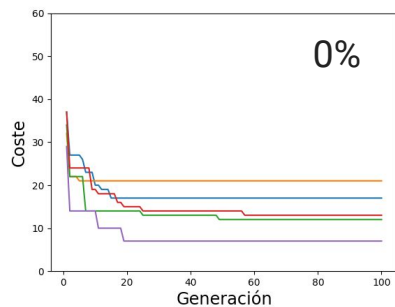
Resultados y discusión



SPOILER

Resultados | Influencia de la probabilidad de mutación

Fijando la población inicial a 8 cromosomas y para cuadrados 4x4:



Poca mutación
(estancamiento
en óptimos loc.)

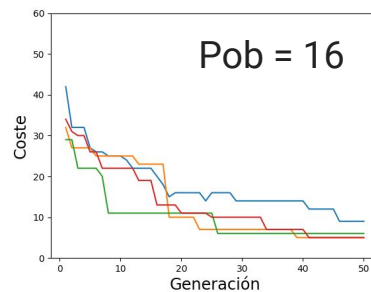
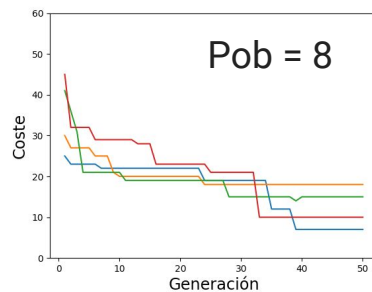
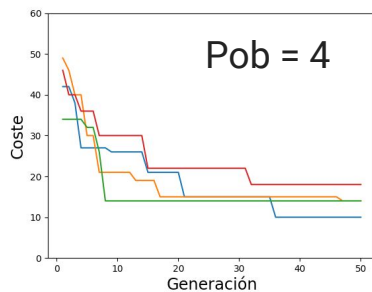
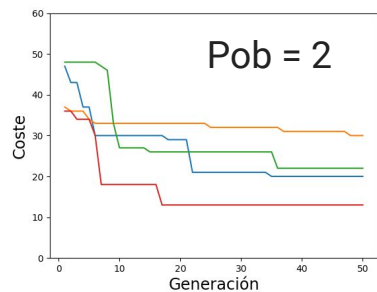


Mucha mutación
(pérdida de buenos
candidatos)

Empíricamente hemos seleccionado como adecuado para la **probabilidad de mutación** un valor de **en torno al 30%**

Resultados | Influencia del tamaño de la población

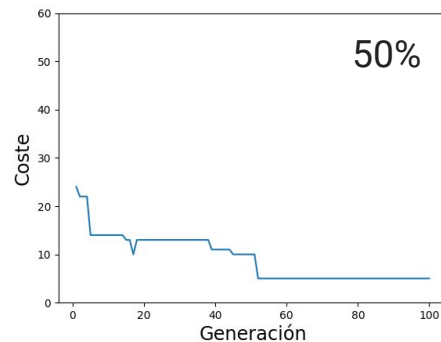
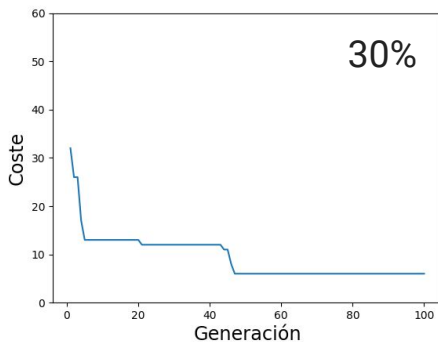
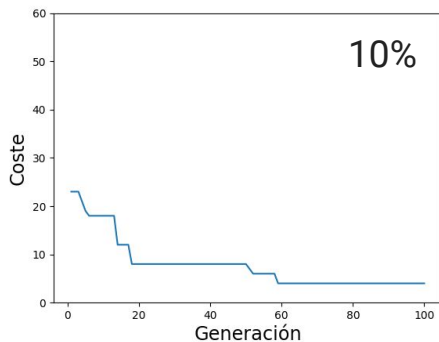
Como cabe esperar, cuanto **mayor es la población**, más probable es que el algoritmo encuentre **mejores soluciones** en las mismas o incluso menos iteraciones, aunque también **aumenta el tiempo de cómputo**.



(Prob. de mutación fijada al 30%)

Resultados | Influencia del tamaño de la población

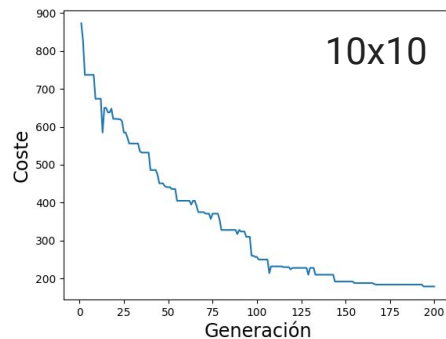
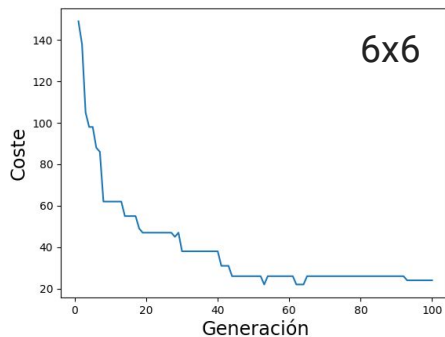
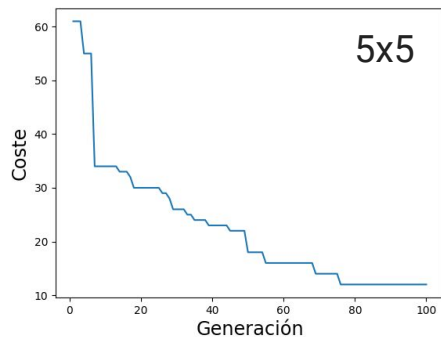
Pero también, **mayor población** implica una disminución del efecto de las **mutaciones** sobre el estancamiento del algoritmo. Para una población de 32:



¡Apenas varía! La población ya tiene mucha exploración de por sí

Resultados | Cuadrados de mayor orden

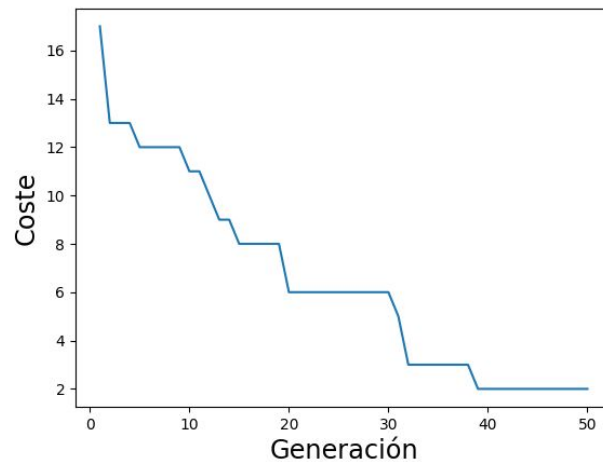
Como el **código** se ha programado para que sea **genérico**, se ha podido comprobar que también funciona con **cuadrados de mayor orden**, si bien aumenta sensiblemente el tiempo de cómputo



(Prob. de mutación 30%, 32 cromosomas)

Mejor resultado | 256 cromosomas, 30% prob. mutación

11	3	6	14	34	
5	16	9	4	34	
10	2	7	15	34	
8	13	12	1	34	
33	34	34	34	34	35



Conclusiones

Ventajas del algoritmo:

- Encuentra **soluciones suficientemente buenas** en un tiempo aceptable
- **Genérico**, lo que hace sencillo el hacer muchas pruebas con distintos tamaños de poblaciones

Contras:

- Cae en **óptimos locales** de los cuales le cuesta salir

Conclusiones

Posibles mejoras:

- Programación del algoritmo con **procesamiento paralelo** usando la API OpenMP.
- Implementar una función que recoja los datos en un CSV de forma automática.
- Implementar una **heurística para que la probabilidad de mutación crezca** si se detectan óptimos locales.

Enlace al [proyecto](#)

Referencias

- Elhaddad, Younis R. and Alshaari, Mohamed A. Genetic Algorithm to Construct and Enumerate 4×4 Pan-Magic Squares. 2014.
- Sallabi, Omar M. and Elhaddad, Younis. An improved genetic algorithm to solve the traveling salesman problem. 2009.
- COELLO, Carlos A. Coello, et al. Evolutionary algorithms for solving multi-objective problems. New York: Springer, 2007.