Comparación de Discrete Cuckoo Search frente a Ant Colony System

Jose Ignacio Alba Rodríguez

Universidad Politécnica de Madrid, Madrid, España

Abstract. Este artículo introduce diferentes implementaciones realizadas para adaptar Cuckoo Search a problemas continuos y realiza una comparación de su eficacia frente a Ant Colony System

1 Motivación

A lo largo de la literatura se han desarrollado un gran número de metaheurísticas de optimización basadas en grupos de animales. Una de las más citadas es Cuckoo Search [7]. Esta se basa en el comportamiento de los cucos y en la búsqueda local mediante "Lévy flights", una forma de paseos aleatorios. Debido a su relación con el movimiento, esta metaheurística esta enfocada a resolver problemas continuos. Posteriormente, Ouaarab et al. llevaron a cabo una modificación del algoritmo con el objetivo de que este sea capaz de resolver problemas discretos, como el problema del viajante (TSP), obteniendo unos resultados espectaculares [4].

Sin embargo, el algoritmo no queda completamente explicado, dando lugar a ambigüedades. Por esta razón, en este trabajo se trata de replicar el algoritmo para comprobar los resultados obtenidos y posteriormente compararlo con Ant Colony System (ACS), el cual sí es un algoritmo de carácter discreto, resolviendo problemas del viajante (TSP) obtenidos de TSPLIB [5].

2 Implementación de Discrete Cuckoo Search

Para poder replicar el algoritmo Cuckoo Search (CS) de forma correcta, se comparó el pseudocódigo presentado en el paper original [7], el cual deja algunos detalles sin concretar, con implementaciones del algoritmo que se pueden encontrar online como la de la librería NiaPy [6] y la de la librería SwarmLib [3], encontrando algunas discordancias. Por esta razón, se han implementado diferentes versiones de DCS para compararlas. En esta sección se determina cómo se ha implementado cada una de las versiones del algoritmo, explicando las diferencias y cómo se ha realizado la discretización.

2.1 Diferencias entre el algoritmo original y las implementaciones

En primer lugar, se introduce la versión mejorada de CS, denominada Improved Cuckoo Search (ICS) [4], siguiendo las ideas expuestas en su pseudocódigo de forma literal.

- Se inicializa una lista de n nidos donde cada nido corresponde con un camino aleatorio, empleando una permutación de los nodos.
- Se calcula el valor de la función objetivo para cada elemento.
- Durante un número máximo de iteraciones it:
  - Una fracción p_c de los cucos son designados como "cucos inteligentes" que realizan una búsqueda local. Es decir, $\lfloor n \cdot p_c \rfloor$ de los n cucos son seleccionados aleatoriamente para calcular una nueva solución x' mediante un Lévy flight. Si x' mejora la solución anterior, la sustituye.
  - Se selecciona un nido al azar a partir del cual se calcula una nueva solución x' mediante un Lévy flight. Se elige una posición aleatoria j de la lista para alojarla. Si $x' \leq x_j$, entonces, x' reemplaza a la solución x_j.
  - Finalmente, una fracción p_a de los peores nidos son abandonados. Es decir, las $\lfloor n \cdot p_a \rfloor$ peores soluciones de la lista son sustituidas por nuevas soluciones, aunque no se especifica cómo se generan estas.

En cambio, en las implementaciones online [6][3], todos los cucos realizan la búsqueda local dentro de su propio nido y no existe un cuco individual que pueda modificar un nido diferente. Además, en lugar de abandonar una fracción de los peores nidos, cada nido x_k tiene una probabilidad p_a de ser abandonado. En ese caso, se crea una solución intermedia entre otras dos x_i y x_j escogidas al azar como $x' = r \cdot (x_j - x_i)$ con $r \sim U(0, 1)$. En caso de que x' mejore a x_k, la sustituye, en caso contrario, se conserva x_k. El lo sucesivo denominaré Complete Cuckoo Search (CCS) a la segunda versión. Se han implementado ambas versiones con el objetivo de poder compararlas. Además, también se compararán con CS sin mejorar, donde $p_c = 0$, siguiendo la línea del paper en el que fue publicado DCS [4].

2.2 Discretización del problema y detalles sobre la implementación

Tal y como se ha descrito en la sección anterior, CS puede resolver problemas continuos, pero no tiene una forma de trabajar con problemas discretos. Para conseguir que resuelva las instancias de TSPLIB [5], necesitaremos adaptar el algoritmo. En [4] se menciona qué el análogo de los movimientos corresponderá con un movimiento local de "2-opt" sobre un camino del grafo para los "pasos pequeños", mientras que para un "paso grande" se empleará un movimiento de "double-bridge". También se menciona que el tamaño del paso será decidido mediante la distribución de Lévy, sin detallar el límite entre grande y pequeño. Para diferenciar entre "paso grande" y "paso pequeño" en esta implementación, se ha añadido el parámetro $s_0 = 1$. De esta forma, al realizar un movimiento, se genera un número aleatorio $s \sim Levy(\alpha = 1, \beta = 1.5)$ y si $s \leq s_0$, entonces se lleva a cabo un movimiento de "2-opt", en caso contrario se llevará a cabo el movimiento "double-bridge". En la versión discreta, cada vez que un cuco quiere realizar un movimiento, realizará este proceso para obtener una nueva solución.

Para generar nuevas soluciones en los nidos abandonados en ICS, donde no se especifica como reemplazarlos, se decidió tomar inspiración de CCS. Al no poder combinar dos soluciones como en el caso continuo, la forma en la que se

genera una solución es escogiendo un nido x_j aleatorio y mutarlo mediante una serie de entre una y cinco permutaciones de dos elementos consecutivos en el camino. De esta forma, se sustituyen los nidos abandonados por otros obtenidos a partir de una mutación de algún nido de la población. En ICS, los peores nidos serán reemplazados, mientras que en CCS, será con una probabilidad p_a y solo en caso de que se produzca una mejora al reemplazar.

Finalmente, se probó inicializar los nidos a partir del resultado obtenido por la heurística Nearest Neighbour (NN) partiendo de un nodo aleatorio, en lugar de generar una solución aleatoria. Mediante esta inicialización se mejoraron en gran medida los resultados obtenidos, ya que se parte de soluciones que necesitaban menos perturbaciones locales para alcanzar el óptimo. Por esta razón, en la implementación se ha decidido mantener la inicialización mediante NN.

3 Resultados

Se han llevado a cabo diferentes pruebas empleando estas implementaciones, además de una implementación propia del algoritmo ACS[2], con diferentes hiperparámetros para resolver las instancias de TSPLIB [5]. Los hiperparámetros finales para algoritmo se presenta en la Tabla 1.

CS Standar	$n = 5$	$it = 50000$	$p_a = 0.25$	$p_c = 0$	$s_0 = 1$	$\alpha = 1$	$\beta = 1.5$
CS Mejorado	$n = 50$	$it = 4000$	$p_a = 0.25$	$p_c = 0.5$	$s_0 = 1$	$\alpha = 1$	$\beta = 1.5$
Complete CS	$n = 25$	$it = 4000$	$p_a = 0.25$	$p_c = \emptyset$	$s_0 = 1$	$\alpha = 1$	$\beta = 1.5$
Ant Colony System	$n = 50$	$it = 200$	$\rho = 0.1$	$\phi = 0.1$	$q_0 = 0.9$	$\alpha = 1$	$\beta = 2$

Tabla 1. Hiperparámetros empleados en las pruebas

En la Tabla 2 se presentan los resultados obtenidos por cada uno de los algoritmos para 15 de las instancias de TSPLIB, que corresponden con las empleadas en [4] para comparar el algoritmo DCS con el algoritmo GSA-ACS-PSOT [1]. Se comparan los datos de el mejor valor encontrado, el valor medio y la desviación típica (STD) tras 30 repeticiones del algoritmo. Posteriormente, en la Figura 1 se representa la desviación porcentual del valor medio de cada uno de los algoritmos con respecto a la solución óptima (PDav%).

4 Conclusiones

La primera observación que se hace al ejecutar los algoritmos es que las diferentes versiones de CS son más rápidas que ACS. Esto se debe a que la complejidad de ACS aumenta en mayor medida con respecto del número de nodos, ya que cada calcula probabilidades para cada nodo en cada paso, mientras que los movimientos locales solo dependen del número de nodos al evaluar la función objetivo. Además, en general, ACS obtiene peores resultados que las variantes de DCS,

Instancia	Óptimo	CS			ICS			CCS			ACS		
		Mejor	Media	STD	Mejor	Media	STD	Mejor	Media	STD	Mejor	Media	STD
eil51	426	428	437.7	5.8	427	432.9	3.8	431	434.0	2.1	445	452.6	4.9
berlin52	7542	**7542**	7833.3	214.1	**7542**	7621.6	141.2	**7542**	7649.9	94.9	7547	7750.7	84.0
eil76	538	554	567.8	8.8	552	558.9	5.5	552	561.6	4.2	559	569.8	5.1
kroA100	21282	21469	22208.2	554.2	21353	21836.4	307.6	22167	22693.0	295.4	22702	23386.6	288.6
kroB100	22141	22767	23279.7	301.3	22604	22970.6	246.1	22620	23325.0	298.6	22645	23352.3	280.0
kroC100	20749	21383	22022.9	429.1	21113	21536.3	240.8	21388	22005.6	358.8	21296	21896.2	273.0
kroD100	21294	22007	23026.7	509.4	21796	22760.7	372.6	22594	23207.6	317.6	22831	23334.4	263.8
kroE100	22068	22553	23142.2	449.8	22261	22665.9	260.6	22574	23124.1	355.0	23052	23994.1	335.1
eil101	629	639	658.4	10.3	635	654.3	10.6	655	670.2	7.6	691	706.8	8.7
lin105	14379	14678	15216.0	251.3	14713	15059.5	122.7	14804	15553.7	261.5	14808	15276.0	179.1
bier127	118282	121038	126564.0	3140.2	120467	122548.9	1536.1	122724	127265.8	1970.9	124462	128129.1	1897.8
ch130	6110	6355	6549.6	98.6	6313	6461.3	87.7	6554	6701.0	80.3	6446	6747.0	122.6
ch150	6528	6556	6763.7	78.9	6575	6632.7	40.5	6695	6849.4	57.3	6865	7011.8	85.7
kroA150	26524	27484	28635.6	535.9	28274	28671.8	217.0	29327	29935.6	375.0	28786	29834.2	446.2
kroB150	26130	26650	27734.6	679.8	26508	27293.4	309.5	28286	29267.2	463.0	28677	29461.1	416.3
kroA200	29368	30219	31256.9	469.8	30420	31239.5	447.8	32703	33404.3	352.2	33276	34165.3	399.1
kroB200	29437	31472	32287.9	491.1	31346	32018.5	327.4	33105	33919.1	411.0	32845	34067.6	434.0
lin318	42029	45461	47049.7	770.3	45746	46406.8	412.1	47854	48963.3	723.4	48202	49762.3	703.0

Tabla 2. Resultados tras 30 repeticiones sobre cada problema

sobre todo a medida que aumenta el número de nodos, por lo que podemos afirmar que es interesante intentar replicar las ideas de algoritmos continuos a problemas discretos.

De entre las versiones de Cuckoo Search, CCS es la que obtiene peores resultados. Esto se debe a que solo acepta movimientos de mejora, por lo que es más probable que quede atrapada en mínimos locales. Una posible mejora pasaría por permitir que movimientos que empeoren la solución en alguna ocasión.

En general, no existe demasiada interacción entre los nidos, pues solo intercambian información cuando alguno es abandonado. Por esta razón, no suele ser tan interesante aumentar el número de nidos tanto como el de iteraciones, destacando en la versión de CS estándar. En esta, el número de cucos empleado es muy bajo, ya que solo se desplaza un cuco por iteración, por lo que una mayor n reduce la probabilidad de concatenar búsquedas locales sobre la misma solución.

Por otro lado, se podrían considerar movimientos que modifiquen menos la solución que "2-opt", como la permutación de dos elementos consecutivos. Otra posible mejora podría consistir en reformular estos movimientos, por ejemplo empleando permutaciones como "paso pequeño" y tantos movimientos "2-opt" como $\lfloor Levy(1, 1.5) \rfloor$ para "pasos grandes".

Finalmente, comparando los resultados obtenidos con los mostrados en el paper en el que es presentada DCS [4], se observa que aquí ICS no mejora a GSA-ACS-PSOT, quedando muy lejos de alcanzar los excelentes resultados que el mismo algoritmo ICS obtiene en [4], que en muchas ocasiones alcanza el óptimo. A pesar de haber empleado un número mucho mayor de iteraciones con el mismo algoritmo, existe una diferencia muy considerable en los resultados alcanzados, por lo que se concluye que, o bien faltan detalles en la implementación que mejoran mucho el algoritmo, o bien puede existir alguna manipulación en los datos expuestos. Por esta razón, creo que sería interesante que existiese una mayor transparencia a la hora de definir el algoritmo, para permitir su replica-

bilidad con exactitud y así poder comprobar la veracidad de los experimentos. Para poder demostrar la fiabilidad de los resultados obtenidos en este artículo, el código empleado será accesible mediante el siguiente enlace de GitHub.

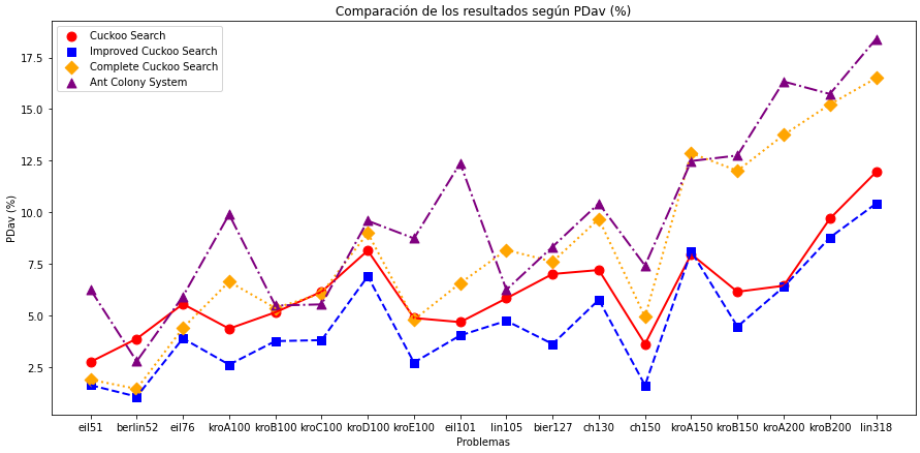

Fig. 1. PDav(%) de la media de las solución con respecto de la solución óptima

References

1. Chen, S.M., Chien, C.Y.: Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. Expert Syst. Appl. **38**, 14439–14450 (11 2011). https://doi.org/10.1016/j.eswa.2011.04.163
2. Dorigo, M., Gambardella, L.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation **1**(1), 53–66 (1997). https://doi.org/10.1109/4235.585892
3. Hanisch, L.: Swarmlib (2020), https://github.com/HaaLeo/swarmlib, fecha de acceso: 14 de enero de 2024
4. Ouaarab, A., Ahiod, B., Yang, X.S.: Discrete cuckoo search algorithm for the travelling salesman problem. Neural Computing and Applications **24** (06 2014). https://doi.org/10.1007/s00521-013-1402-2
5. Reinelt, G.: TSPLIB–a traveling salesman problem library. ORSA Journal on Computing **3**(4), 376–384 (1991)
6. Vrbančič, G., Brezočnik, L., Mlakar, U., Fister, D., Fister Jr., I.: NiaPy: Python microframework for building nature-inspired algorithms. Journal of Open Source Software **3** (2018). https://doi.org/10.21105/joss.00613, https://doi.org/10.21105/joss.00613
7. Yang, X.S., Deb, S.: Cuckoo search via lévy flights. In: 2009 World Congress on Nature Biologically Inspired Computing (NaBIC). pp. 210–214 (2009). https://doi.org/10.1109/NABIC.2009.5393690