ISSN 1870-4069

Recocido simulado y el algoritmo de selección clonal aplicados al problema del agente viajero

Nayeli Joaquinita Meléndez Acosta

Universidad del Istmo,
México

nayelimelendez@ gmail.com

Resumen. Este artículo presenta una evaluación de los algoritmos de Recocido Simulado (RS) y Selección Clonal (CLONALG) aplicados al Problema del Agente Viajero, donde el objetivo es minimizar la distancia del recorrido. Los dos algoritmos han sido estudiados por separado para solucionar este problema y ambos han mostrado ser eficientes respecto a otros algoritmos, sin embargo, no existe comparación entre estos. El rendimiento de ambos algoritmos se determinó utilizando dos conjuntos de datos, el primero de estos entre ciudades generadas aleatoriamente. En la segunda prueba se consideraron instancias de ciudades tomadas de la librería TSPLIB. Los experimentos muestran que los dos algoritmos producen buenos resultados para encontrar la mejor ruta. Recocido Simulado obtuvo mejores soluciones en instancias con mayor cantidad de ciudades y en la mayoría de los casos se obtiene en menos iteraciones. En un futuro se pretende implementar variantes de los algoritmos originales y aplicarlos a problemas reales en municipios de la región.

Palabras clave: Algoritmos metaheurísticos, distancia, recorrido, sistema inmune, enfriamiento, temperatura.

Simulated Annealing and Clonal Selection Algorithm applied to the Traveling Salesman Problem

Abstract. This article presents an evaluation of the Simulated Annealing (SA) and Clonal Selection (CLONALG) algorithms applied to the Traveling Salesman Problem, where the objective is to minimize the distance of the route. The two algorithms have been studied apart to solve this problem and both have been shown to be efficient while compared to other algorithms, however, there is no comparison between them. The performance of both algorithms was determined using two data sets, the first of these between randomly generated cities. In the second test, instances of cities taken from the TSPLIB library were considered. Experiments showed that the two algorithms produce good results for finding the best route. Simulated Annealing obtained better solutions in instances with more cities and in most cases, it is obtained in fewer iterations. In the future, it is

intended to implement variants of the original algorithms and apply them to real problems in the municipal localities.

Keywords: Metaheuristic algorithms, distance, travel, immune system, cooling, temperature.

1. Introducción

Los algoritmos metaheurísticos han mostrado ser buenos para resolver el problema de agente viajero, ya que es un problema NP-difícil, este problema es muy simple con un pequeño número de ciudades, pero la complejidad computacional aumenta exponencialmente con el número de ciudades y no hay un algoritmo conocido que lo resuelva en tiempo polinomial [1, 2].

Los algoritmos metaheurísticos utilizan estrategias de búsqueda para explorar el espacio de búsqueda de manera eficiente, aun en espacios muy grandes. Este tipo de algoritmos, no garantizan que se encuentre una solución óptima, pero pueden proporcionar resultados muy buenos, casi óptimos [3, 4]. La necesidad de encontrar una solución rápida al PAV ha llevado a la creación de muchos algoritmos con resultados aproximados. Algunos algoritmos metaheurísticos utilizados para resolver este problema son: Algoritmos Genéticos (AG), el Recocido Simulado (RS), Optimización por Cúmulo de Partículas *(Particle Swarm Optimization, PSO*), Optimización por Colonia de Hormigas (*Ant Colony Optimization, ACO*), Algoritmo Colonia de Abejas Artificiales (*Artificial Bee Colony algorithm, ABC*), la Búsqueda Tabú y la Búsqueda Cuckoo, por mencionar algunos [2, 5 y 6].

En el PAV existen varias ciudades que debe recorrer un agente, el agente debe visitar cada ciudad solo una vez y regresar a la ciudad donde comenzó. El objetivo del PAV es encontrar en el menor número de iteraciones, el orden en que debe visitar a las ciudades, de modo que la distancia recorrida sea la menor en comparación con cualquier otro orden, reduciendo así el costo total del viaje. Además, este problema tiene muchas aplicaciones en el mundo real, tales como: cableado de computadoras, enrutamiento, comunicación inalámbrica, horarios de transporte, procesos de fabricación y programación de vehículos planificación y logística [2, 3].

Este articulo implementa dos enfoques heurísticos: el algoritmo de recocido simulado y el algoritmo de selección clonal (CLONALG) para resolver el problema del agente viajero, el rendimiento de ambos algoritmos se determinó utilizando dos conjuntos de datos, estos algoritmos ya han sido estudiados por separado para resolver PAV y ambos han mostrado ser eficientes respecto a otros algoritmos, sin embargo, no existe comparación entre ellos. El documento está organizado de la siguiente manera: la sección dos discute los trabajos relacionados. En la tercera sección se presenta una introducción al problema del agente viajero proporcionando una breve descripción sobre cómo funciona el PAV. La sección cuatro da detalles sobre los algoritmos implementados: RS y CLONALG. La sección cinco describe sobre la implementación de los algoritmos, esta sección se enfoca en pseudocódigo y describe el flujo de cada algoritmo.

La sección seis aborda las pruebas realizadas del PAV usando RS y CLONALG, junto con el análisis comparativo entre los dos algoritmos. Finalmente, en la última sección se muestran las conclusiones y trabajo futuro.

2. Trabajos relacionados

La necesidad de encontrar una solución rápida a los problemas de optimización, ha llevado a la creación de muchos algoritmos con resultados aproximados. Algunos de los algoritmos heurísticos más populares para resolver el problema del agente viajero son los algoritmos genéticos, el recocido simulado, la optimización de colonias de hormigas, la optimización por cúmulo de partículas, el algoritmo de búsqueda tabú [2, 5 y 6].

El diseño y el análisis de algoritmos efectivos para resolver el PAV sigue siendo muy importante debido a que se quiere mejorar su eficiencia para resolver otros problemas de optimización en general [7]. La combinación o creación de nuevos algoritmos se realiza con el objetivo de tratar mejor con el PAV, algunos trabajos relacionados donde han utilizado recocido simulado y el algoritmo de selección clonal son:

Sumathi en [2] realizó un análisis comparativo en función del número de iteraciones necesarias para alcanzar el camino óptimo, utilizaron dos algoritmos PSO y RS. Zhou en [6] propuso un nuevo algoritmo basado en recocido simulado y el algoritmo programación de expresión de genes (PEG) para resolver el PAV, utiliza recocido simulado para aumentar la diversidad de la PEG y mejorar la capacidad de búsqueda global. En los experimentos utilizaron seis instancias de referencia y los resultados mostraron que el algoritmo propuesto supera a otros algoritmos heurísticos conocidos, en términos de la mejor solución, la peor solución y el tiempo de ejecución del algoritmo. Pang en [7] diseñó un nuevo algoritmo utilizando CLONALG y la técnica de búsqueda local, este último fue empleado para acelerar la madurez del sistema, usaron varios problemas de referencia en TSPLIB para evaluar el rendimiento del algoritmo propuesto. Sus resultados mostraron que el algoritmo propuesto funciona mejor que el CLONALG estándar y un algoritmo genético.

Muthreja en [8] realizó la implementación del algoritmo de selección clonal y propuso una versión modificada para resolver el problema del agente viajero, la distancia más corta y el tiempo de convergencia más corto fueron utilizados para comparar el rendimiento de ambos algoritmos con dos algoritmos convencionales: un algoritmo genético y el algoritmo Ramificación y Poda (*Branch and Bound, BB*). Los resultados mostraron que el algoritmo de selección clonal y el propuesto dieron mejores soluciones. Lin en [9] propuso un algoritmo de búsqueda hibrido utilizando recocido simulado y el algoritmo de búsqueda tabú para resolver el PAV, además proponen parámetros adaptativos que el algoritmo puede ajustar automáticamente en función de los ejemplos, para probar su algoritmo utilizaron TSPLIB y los resultados mostraron que el algoritmo propuesto pudo encontrar soluciones satisfactorias en un periodo de tiempo razonable, además demostraron una mejora en la precisión y eficiencia comparado con otros algoritmos.

Para resolver el PAV además de los algoritmos implementados en este trabajo, se han utilizado otros algoritmos de búsqueda heurísticos, algunos trabajos relacionados son:

Wang en [10] propuso un algoritmo hibrido inteligente, combinó el algoritmo ABC y el algoritmo PSO, los resultados experimentales mostraron que el algoritmo hibrido es mejor comparado otros algoritmos similares, además tiene resultados satisfactorios en el PAV a gran escala. Myszkowski en [11] combinó los mecanismos de un algoritmo genético de manera única para resolver el PAV multiobjetivo. Wei en [12] realizó un estudio de tres algoritmos heurísticos para resolver el PAV: un AG, PSO y ACO, aplicó una estrategia de inicialización y búsqueda local para acelerar aún más la velocidad de búsqueda. Li en [13] utilizó el algoritmo ABC y le agrego una estrategia de búsqueda local para optimizar el camino, sus resultados mostraron que el algoritmo propuesto es muy confiable para resolver el PAV.

Hammouri en [14] utilizó un algoritmo metaheurístico denominado Algoritmo Libélula (*Dragonfly Algorithm, DA*), para evaluar su calidad se comparó con otros algoritmos metaheurísticos utilizando el mismo conjunto de datos tomado de TSPLIB, los resultados finales mostraron que el algoritmo propuesto es capaz de resolver eficientemente el PAV, produciendo resultados competitivos. Chandrawati en [15] utilizó el Algoritmo Luciérnaga (*Firefly Algorithm, FA*) para resolver tres problemas: planificación de rutas, enrutamiento de vehículos y el agente viajero. Juneja en [16] utilizó un algoritmo genético para encontrar una solución factible para el PAV. Jaradat en [17] propuso una solución para el PAV utilizando el Algoritmo Luciérnaga y el agrupamiento de K-Means, el enfoque propuesto comprende tres pasos principales: agrupar los nodos, encontrar la ruta óptima en cada grupo y volver a conectar los grupos.

Algunos trabajos relacionados para resolver otros problemas semejantes al PAV que han utilizado algoritmos heurísticos son:

Flórez en [18] presentó una evaluación de dos metaheurísticas para resolver el problema de planificación de trabajos (*Job Shop Scheduling*): sistemas inmunes artificiales y el ACO. El objetivo era encontrar la secuencia de trabajos que requirieran la menor cantidad de tiempo para ser ejecutada en las maquinas disponibles. Para medir el desempeño de ambos algoritmos compararon la calidad de las soluciones obtenidas con respecto a la mejor solución conocida. Vairamuthu en [19] utilizó CLONALG para resolver el problema de la planificación de N-trabajos, la implementación fue realizada en Matlab.

3. El problema del agente viajero

El problema del agente viajero se puede definir como un problema donde existen varias ciudades (nodos) y dada una ciudad de partida (nodo inicial), el agente viajero debe visitar cada cuidad solo una vez y regrese a la cuidad de partida, de manera que se minimice la distancia del recorrido [20, 21].

Matemáticamente se puede definir como un grafo $G = (V, E)$, los vértices son las ciudades y las aristas corresponden a las carreteras que unen dos ciudades [22]. V es un vector de ciudades $V = \{1, 2, 3, \dots, \mathrm{n}\}$ donde n es el número de ciudades. Entonces, $E = \{(\mathrm{i}, \mathrm{j}) : \mathrm{i}, \mathrm{j} \in \mathrm{V}, \mathrm{i} \neq \mathrm{j}\}$ es un conjunto de aristas. Una matriz de costos $C = (c_{ij})$ se define en E.

Los vértices son puntos $P_i = (x_i, y_i)$ y $P_j = (x_j, y_j)$. Entonces, la distancia entre estos puntos se define en (1):

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\,. \tag{1}$$

Por lo tanto, el problema se podría formalizar como minimizar (2):

$$f(x) = \sum_{i,j}^{n} c_{ij} \tag{2}$$

para encontrar la distancia total del recorrido más corto, visitando cada ciudad exactamente una vez y regresando a la ciudad de partida [1, 21].

Existen varios tipos de PAV, un PAV simétrico es cuando $c_{ij} = c_{ji}$, para todo i, j y asimétrico en caso contrario, para un PAV asimétrica de n ciudades existen $(n-1)!$ soluciones posibles, mientras que para un simétrico existen $\frac{(n-1)!}{2}$ soluciones posibles. En ambos casos no es posible realizar una búsqueda exhaustiva debido a que la cantidad de soluciones puede llegar a ser muy grande, ya que cuando n crece se hace imposible encontrar el camino más corto en tiempo polinomial [1, 2 y 23]. Otra variante es el PAV Aleatorio, donde las coordenadas de las ciudades se generan aleatoriamente en lugar de tener ejemplos estáticos ya disponibles en TSPLIB.

PAV sigue siendo investigado por tres razones. Primero puede modelar una gran cantidad de problemas del mundo real. Segundo se demostró que era un problema NP-difícil. Tercero los problemas NP- difícil son intratables es decir no se ha encontrado una manera eficiente de resolverlos [20]. Además, los problemas NP- difícil son más o menos equivalentes entre sí, si se logrará resolver uno de ellos, podría resolverse el resto.

4. Algoritmos metaheurísticos

En PAV la búsqueda exhaustiva es ineficiente debido al gran número de soluciones posibles, incluso cuando el problema es de tamaño moderado. Por lo tanto, el objetivo es obtener heurísticamente buenas soluciones en tiempo razonable.

4.1. Recocido simulado

Recocido Simulado es un algoritmo probabilístico inspirado en el recocido metalúrgico, que es una técnica de enfriamiento controlado para reducir defectos [1, 21]. Comienza con una mezcla de metal a alta temperatura y enfría lentamente la mezcla, permitiendo la formación de estructuras optimas a medida que el material se enfría [3].

El algoritmo se describe formalmente de la siguiente manera: comienza con una solución aleatoria x_p, luego se genera una solución vecina aleatoria x_n en cada iteración. Después se calcula la diferencia entre los valores de la función objetivo $\Delta f = f(x_n) - f(x_p)$. Si la solución vecina es una mejor solución (es decir, se mejora la función objetivo) entonces reemplazará a la solución actual.

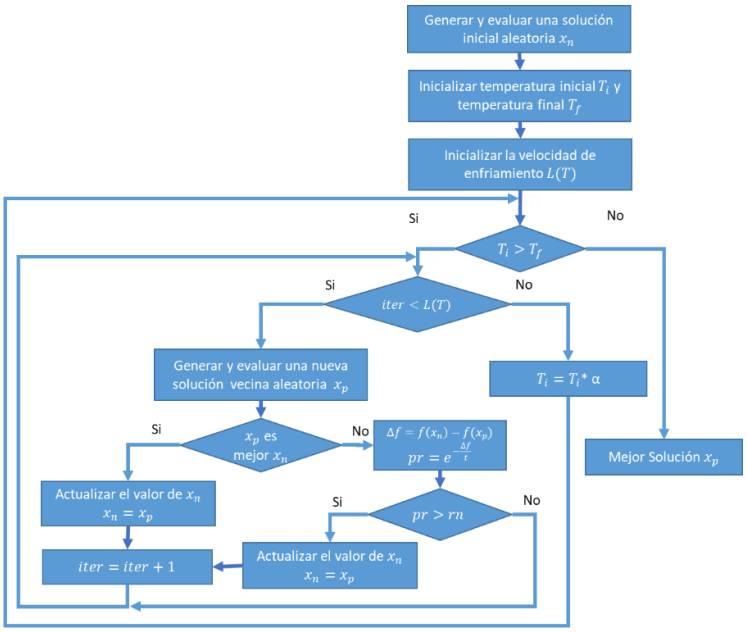


Fig. 1. Diagrama general de Recocido Simulado [1].

Si la solución es peor, se puede elegir reemplazar la solución actual dependiendo de una probabilidad definida en (3) que depende de la temperatura [1, 21 y 24]:

$$pr = e^{-\frac{\Delta f}{t}}, \tag{3}$$

donde pr se reduce a medida que el algoritmo avanza y (t) es la temperatura o el parámetro de control, esto significa que a medida que el algoritmo avanza, el parámetro de temperatura disminuye, dando a las peores soluciones una menor posibilidad de reemplazar a la solución actual [21, 24], es decir las peores soluciones en las primeras iteraciones del algoritmo evitan que la solución converja a un mínimo local, lo que significa que aceptar peores soluciones impide que el algoritmo quede atrapado en los mínimos locales y no pueda explorar más soluciones posibles. Esta aceptación se logra generando un número aleatorio (rn) entre $0 \leq rn \leq 1$ y después se compara con el umbral. Si $pr > rn,$ la solución actual es reemplazada por la solución vecina. El procedimiento se repite hasta que se cumpla la condición de paro.

Los parámetros que afectan el resultado del algoritmo son tres: la temperatura inicial (T_i), la velocidad a la que disminuye la temperatura (es decir, velocidad de enfriamiento) y la condición de parada del algoritmo [1, 21]. La velocidad de enfriamiento $L(t)$ es el número de iteraciones que usan la misma temperatura antes de disminuirla para buscar una mejor solución vecina, es decir se realiza una búsqueda local (conocida como Ciclo de Metrópolis). La velocidad de enfriamiento α es el grado de disminución de la temperatura conforme avanzan los ciclos hasta llegar a la temperatura final (T_f), se recomienda que este entre (0.8, 0.99). El algoritmo RS se describe en el diagrama de flujo de la Fig. 1.

Una ventaja de RS en comparación con otros métodos es su capacidad para no quedar estancado en mínimos locales, lo que significa que el algoritmo no siempre rechaza los cambios. Es decir, es un método probabilístico utilizado para obtener el mínimo de una función que puede tener varios mínimos locales [3].

4.2. Algoritmo de selección clonal

En los últimos años ha habido un gran interés en estudiar sistemas inspirados biológicamente para resolver problemas de optimización [8]. Los sistemas inmunes artificiales (SIA) son una clase de sistemas computacionalmente inteligentes inspirados en los principios y procesos del sistema inmune [19, 25].

El algoritmo de selección clonal es un algoritmo de optimización evolutivo inspirado en los mecanismos del sistema inmune biológico que tienen un tremendo potencial en muchas aplicaciones de optimización [8, 26].

El sistema inmune del cuerpo humano es capaz de reconocer las células extrañas que entran en un cuerpo humano que pueden causar enfermedades. Estas células extrañas se llaman antígenos, nuestro cuerpo aprende a neutralizar los efectos de los antígenos al comprender el patrón de comportamiento y oponiéndose a ellos para mantener la estabilidad del cuerpo. Las células producidas por el cuerpo humano para luchar con antígenos se denominan anticuerpos, el sistema inmune controla la acción del antígeno. El algoritmo de selección clonal tiene la funcionalidad de filtrar los anticuerpos en función de la afinidad. El grado de reconocimiento entre anticuerpo y antígeno se denomina afinidad. Cuanto mejor sea el reconocimiento, mayor será la afinidad y viceversa. Los anticuerpos capaces de luchar contra los antígenos son capaces de multiplicarse a través de un proceso de clonación proporcional a la afinidad, para luchar contra ellos [8].

La teoría de la selección clonal incluye tres procesos: proceso de selección clonal, proceso de proliferación, y proceso de maduración. Cuando un antígeno invade será reconocido por las células de sistema inmune con cierta afinidad. En la etapa de selección se activan las células con alta afinidad para proliferar, las cuales son estimuladas para producir grandes cantidades de clones, es decir realizar el proceso de proliferación. En la etapa final, en el proceso de maduración estos clones se pueden mutar y se convierten en células que secretan grandes cantidades de anticuerpos que retiene el patrón antígeno para futuras infecciones [27]. Así de manera semejante funciona el algoritmo de selección clonal (también llamado CLONALG), el cual se compone de cuatro procesos: selección, expansión clonal o proliferación, hipermutación (madurez de afinidad) y re-selección. La idea principal de CLONALG es que la mutación acumulativa puede dirigir la búsqueda a mejores soluciones. Este algoritmo emplea un operador de clonación e hipermutación en lugar de la operación de selección y cruce como en AG [7].

Básicamente el proceso del algoritmo de selección clonal consiste en generar aleatoriamente una población de soluciones candidatas; después seleccionamos un porcentaje de los mejores individuos, los cuales son clonados, luego a estos individuos se les aplica una hipermuta y finalmente seleccionamos la nueva población para la siguiente generación [26], este proceso se puede observar en la Fig. 2.

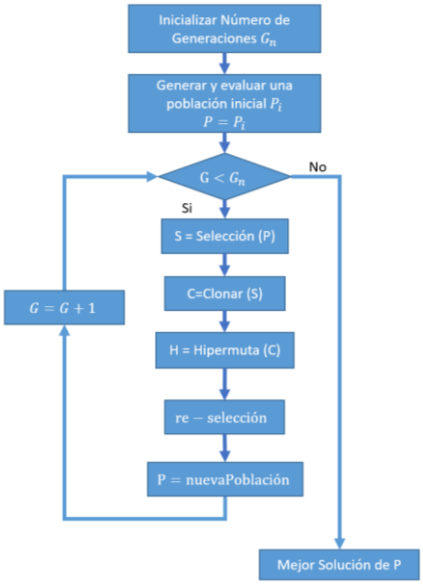


Fig. 2. Diagrama general del Algoritmo de Selección Clonal [8].

El CLONALG puede considerarse como un algoritmo evolutivo basado en la población. Cada individuo (anticuerpo) es esencialmente un buscador local óptimo, y la hipermutación puede considerarse como un proceso de exploración de vecindad ciega en el espacio de búsqueda en relación con el anticuerpo original. En el proceso de re-selección, los anticuerpos proliferados se seleccionan de acuerdo con sus valores de afinidad. Después de un número suficiente de generaciones, se espera que los anticuerpos encuentren las soluciones óptimas o las soluciones óptimas aproximadas [7].

El CLONALG ha sido ampliamente utilizado para resolver una variedad de problemas como reconocimiento de patrones, la optimización de funciones multimodales y el problema del agente viajero [7, 25].

5. Implementación

En esta sección se explica cómo se llevó a cabo la implementación de los algoritmos recocido simulado y el algoritmo de selección clonal para resolver el PAV. Ambos algoritmos están codificados en Scilab 6.0.1.

Un aspecto muy importante para ambos algoritmos es la representación de una solución al problema. Existen principalmente dos métodos para representar el recorrido del PAV: representación de adyacencia y representación de ruta.

Para ambos algoritmos se ha utilizado la representación de ruta, es decir, una solución es representada por un individuo de longitud n, donde n es el número de nodos en el problema. El tipo de representación usada es por permutaciones donde cada gen del cromosoma es una etiqueta del nodo de tal forma que ningún nodo puede aparecer dos veces en el mismo cromosoma.

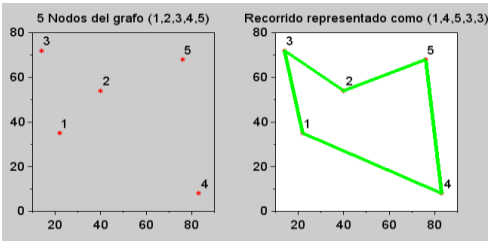


Fig. 3. (a) Nodos del grafo y (b) Recorrido representado por (1,4,5,2,3).

Por ejemplo, si {1, 2, 3, 4, 5} son las etiquetas de los 5 nodos del grafo, entonces un recorrido puede ser representado como (1, 4, 5, 2, 3), como se muestra en la Fig. 3.

5.1. Recocido simulado

Ahora nos centramos en la aplicación de recocido simulado al PAV. Primero se genera una semilla es decir una solución inicial (recorrido aleatorio). Para evaluar una solución se considera como función de aptitud definida en (2) donde $f(x)$ calcula el costo (o valor) del recorrido representado por una solución y la distancia entre estos puntos se define en (3).

Otro aspecto importante en este algoritmo es la generación de la solución vecina, la cual se crea eligiendo aleatoriamente dos posiciones en la solución y después se intercambian los genes (ciudades) localizados en dichas posiciones. Para el ejemplo anterior de 5 ciudades si se eligen las posiciones 2 y 4 tenemos [20]:

(1 4 5 2 3) Solución actual
  | | Posiciones seleccionadas
(1 2 5 4 3) Solución vecina

Para el enfriamiento, se ha utilizado un cambio de temperatura lineal. Esto significa que cada nueva temperatura se determina a partir de la temperatura anterior.

5.2. Algoritmo de selección clonal

El CLONALG se puede describir de la siguiente manera [7, 18 y 25]:

Paso 1. Generar aleatoriamente una población de individuos (M – anticuerpos); es un conjunto de soluciones potenciales, es decir un conjunto de recorridos, donde cada uno representa un posible recorrido,

Paso 2. Se ejecuta de manera iterativa. Para cada generación:

Paso 3. Determinar la afinidad de la población, es decir evaluar cada solución con respecto a la función f a optimizar de la ecuación (2) y seleccionar los n mejores individuos (selección).

Paso 4. Reproducir los n mejores individuos (la mitad de la población inicial) generando una población temporal de clones Mc, se clona con el mismo número de copias

Tabla 1. Parámetros usados para Recocido Simulado y CLONALG.

Recocido Simulado	Valores	CLONALG	Valores
Temperatura inicial	20-50	Número de Generaciones	500-5000
Velocidad de enfriamiento	20-50	Factor de clonación	50%
Grado de Enfriamiento	0.97	Factor de mutación	10%
Número de Ciudades	10-50	Número de Ciudades	10-50
		Tamaño de la población	200

(expansión clonal). La cantidad n de clones, está dado por la ecuación: $\mathrm{Mc} = (\beta * \mathrm{M})$, donde β es un factor de clonación y M es la cantidad de individuos. El factor de clonación el valor de 0,1.

Paso 5. Todas las copias clonadas se someten a un proceso de hipermutación con una tasa proporcional a su afinidad, cuanto mayor sea la afinidad, menor la tasa de mutación. Así se genera una nueva población de clones (C).

Paso 6. Determinar el valor de afinidad de todos los individuos mutados, se agregan estos individuos mutados a la población y se seleccionan M de estos individuos maduros para la siguiente generación (re-selección).

Paso 7. El algoritmo terminará después de un número de generaciones/iteraciones o si se ha encontrado la solución óptima para el caso de las instancias tomadas de TSPLIB.

6. Pruebas y resultados

Los dos algoritmos fueron desarrollados en SCILAB y han sido ejecutados en Windows 10 utilizando un CPU core i5 y 16 GB de RAM. Para comparar ambos algoritmos se realizaron dos pruebas. En la primera se hicieron experimentos sobre instancias de PAV generadas aleatoriamente de entre 10 y 30 ciudades. En la segunda se han probado en instancias de referencia entre 10 y 50 ciudades, las cuales fueron tomadas de TSPLIB [28].

A continuación, se definen los parámetros utilizados en las pruebas, para el recocido simulado: la temperatura inicial es de 20 a 50, la velocidad de enfriamiento es de 20 a 50 y el grado de enfriamiento es de 0.97. Para el algoritmo de selección clonal: el tamaño de la población es de 100 a 300, el factor de clonación 0.5, el factor de mutación 0.1 y un máximo de 500 a 5000 iteraciones.

El número de iteraciones fue determinado dependiendo del número de ciudades en cada problema. En ambos algoritmos los parámetros han sido determinados basados en las recomendaciones de la literatura y después de ejecutar los algoritmos, se obtuvieron las mejores soluciones utilizando los parámetros mostrados en las Tabla 1.

Para evaluar a los algoritmos se realizaron dos pruebas. En ambas pruebas los experimentos se realizaron 10 veces para cada caso, obteniendo el mejor recorrido, el peor, la media el número de generaciones del recorrido para cada algoritmo.

Tabla 2. Resultados para 10, 20 y 30 ciudades generadas aleatoriamente.

Número de nodos	Algoritmo	Mejor	Peor	Media	Número de generaciones
10	RS	270.35	270.35	270.35	40
	CLONALG	270.35	270.35	270.35	30
20	RS	382.5784	407.8863	391.7068	100
	CLONALG	382.5784	402.4819	391.8037	400
30	RS	510.4583	512.4537	511.3585	500
	CLONALG	510.3138	513.1765	511.5687	1000

Tabla 3. Resultados para instancias de referencia tomadas de TSPLIB [28].

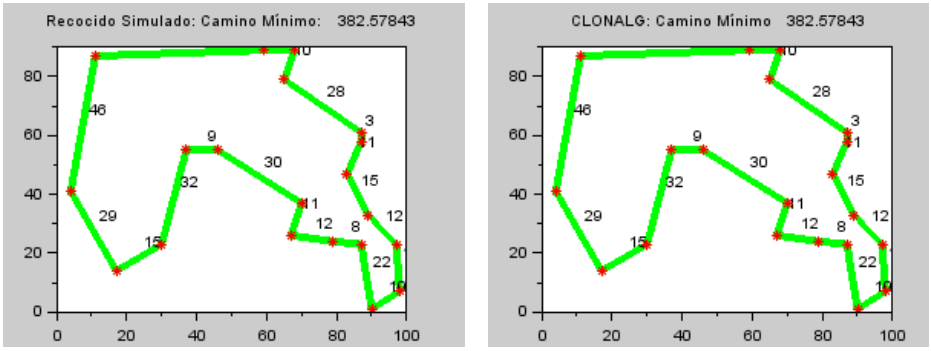
Instancia	Óptimo	Algoritmo	Mejor	Peor	Media
Ulysses16	74.1087	RS	75.08	79.23	76.96
		CLONALG	73.98	74.14	74.04
Ulysses22	75.6651	RS	78.4622	87.2333	83.2954
		CLONALG	75.7948	77.3688	76.4040
Bayg29	9074.1480	RS	9229.2592	9593.4947	9470.5960
		CLONALG	9074.1480	9879.7366	9594.0454
Att48	33523.709	RS	33936.386	34881.618	34398.193
		CLONALG	34700.472	38108.189	35727.915

En la primera prueba los algoritmos son aplicados a conjuntos de datos con diferente número de ciudades generadas aleatoriamente. Los resultados experimentales se presentan en la Tabla 2.

La Tabla 3 muestra los resultados de la segunda prueba se consideraron instancias entre 10 y 50 ciudades tomadas de la librería TSPLIB [28], que es una biblioteca de instancias de muestra para el TSP de varias fuentes y de varios tipos. Las instancias elegidas son de tipo PAV simétrico, donde la distancia del nodo i al nodo j es la misma que la del nodo j al nodo i. Además, se conocía la ruta óptima.

Cuatro instancias han sido seleccionadas de la librería TSPLIB, con diferente número de ciudades. La primera instancia es llamada Ulysses16, este tiene 16 ciudades de la Odisea de Ulises, que representa el viaje de Ulises en la época fenicia a las 16 maravillas del Mediterráneo y el recorrido mínimo es de 74.1087. Ulysses22 tiene 22 ciudades de la Odisea de Ulises y el recorrido mínimo es de 75.6651. Bayg29 tiene las distancias geográficas de 29 ciudades de Baviera, Alemania y el recorrido mínimo es de 9074.1480. Att48 tiene 48 capitales de los EE. UU y el recorrido mínimo es de 33523.709.

La Fig. 4 muestra los mejores resultados para 20 ciudades generadas aleatoriamente de la primera prueba, podemos observar en la Tabla 2 que ambos algoritmos encuentran el mejor camino para 10 y 20 ciudades, aunque RS para 20 ciudades encuentra la peor



(a) Recocido Simulado, mejor solución para 20 ciudades (b) CLONALG, mejor solución para 20 ciudades

Fig. 4. Mejores resultados para ciudades generadas aleatoriamente.

ruta (407.8863), aun así, su media de 391.7068 es casi igual a CLONALG con valor de 391.8037, indicando que RS encuentra más veces una mejor ruta.

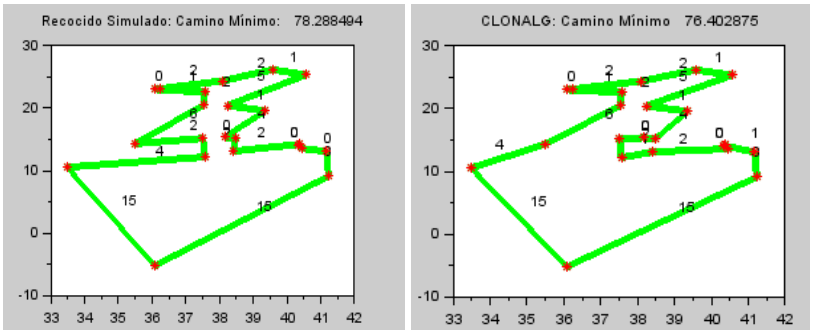
La Fig. 5 muestra los mejores resultados para los problemas ulysses22, bayg29 y att48 de la segunda prueba para ambos algoritmos.

Podemos observar en los resultados de la Tabla 2 y las imágenes de la Fig. 5 que el algoritmo de RS es mejor que CLONALG para un número de ciudades más grande, lo podemos ver en la instancia Att48 con 48 ciudades, donde RS encuentra la ruta más cercana a la mejor ruta con valor de 33936.386, además la media de RS (34398.193) en comparación con la de CLONALG (35727.915) muestran que RS encuentra buenas soluciones. Por otro lado, en la instancia bayg29 con 29 ciudades, RS también tiene mejor promedio (9470.5960) que CLONALG (9594.0454), aunque CLONALG encuentra la peor ruta con valor de 9879.7366, CLONALG sí encuentra la ruta optima (9074.1480), sin embargo, RS nunca encuentra la ruta optima. Ahora para instancias con un número pequeño de ciudades como en Ulysses16 y Ulysses22, CLONALG tiene mejores resultados encontrando la mejor ruta (73.9) para Ulysses16 y acercándose mucho a la mejor ruta para Ulysses22.

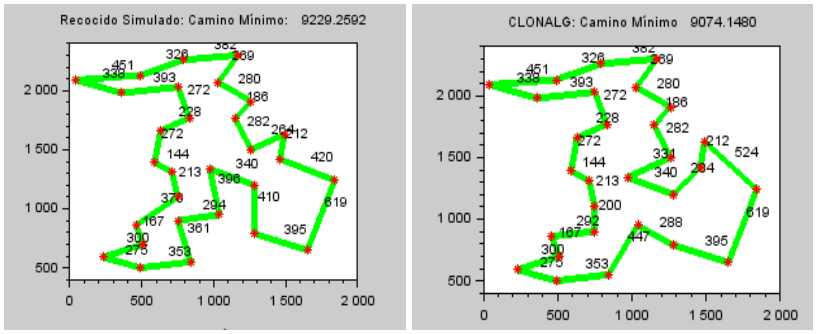
Recocido simulado casi siempre encuentra la mejor ruta en menos iteraciones, se puede observar en la Fig.6. La Fig. 6(a) muestra la evolución de la función objetivo de RS y CLONALG para 30 ciudades aleatorias, se muestra el costo del recorrido sobre el número de generaciones. También cabe destacar que en la mayoría de los casos RS encuentra una solución aceptable en menos generaciones. Finalmente, la Fig. 6(b) muestra la evaluación de RS y de CLONALG para 20 ciudades generadas aleatoriamente.

Podemos observar en la Fig. 6 que cuando las ciudades son generadas aleatoriamente el algoritmo de Recocido Simulado llega primero al valor óptimo, lo que no CLONALG, el cual necesita más generaciones para encontrar el valor óptimo.

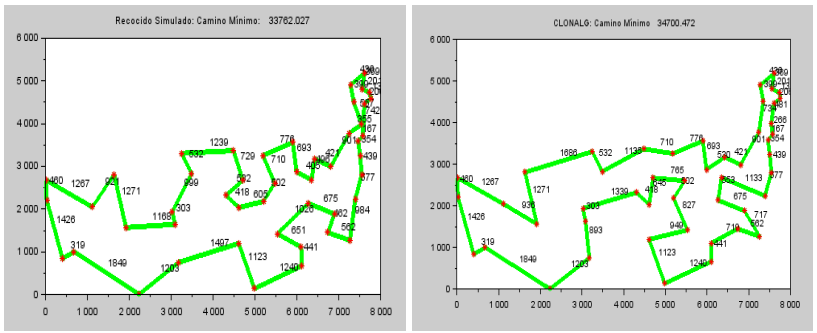
Recocido Simulado realiza una búsqueda más extensa, lo que le permite ser mejor buscando en instancias con un número de ciudades grande, pero a veces no encuentre la mejor solución.



(a) Recocido Simulado, mejor solución para Ulysses22 (b) CLONALG, mejor solución para Ulysses22



(c) Recocido Simulado, mejor solución para Bayg29 (d) CLONALG, mejor solución para Bayg29

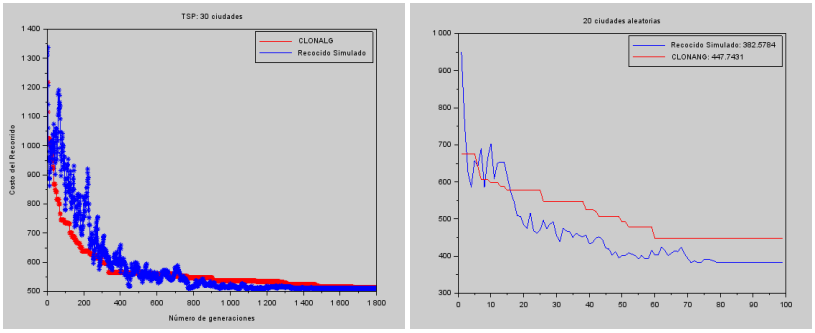


(e) Recocido Simulado, mejor solución para Att48 (f) CLONALG, mejor solución para Att48

Fig. 5. Mejores resultados. (a) Recocido Simulado y (b) CLONALG para Ulysses22. (c) Recocido Simulado y (d) CLONALG para Bayg29, (e) Recocido Simulado y (f) CLONALG para Att48.

7. Conclusiones y trabajo futuro

En esta investigación presentamos la implementación de dos algoritmos de optimización heurísticos, Recocido Simulado y el Algoritmo de Selección Clonal aplicados a resolver el Problema del Agente Viajero.



(a) 30 ciudades aleatorias (b) 20 ciudades aleatorias

Fig. 6. Evolución de los resultados de Recocido Simulado y CLONALG para (a) 30 ciudades generadas aleatoriamente y (b) 20 ciudades generadas aleatoriamente.

Para realizar una evaluación de ambos algoritmos se realizaron dos pruebas. En la primera prueba los algoritmos son aplicados a conjuntos de datos con diferente número de ciudades generadas aleatoriamente y la segunda prueba se realizó sobre instancias tomadas de la librería TSPLIB. Los algoritmos están codificados en Scilab.

Los dos algoritmos han sido estudiados por separado para solucionar este problema y ambos han mostrado ser eficientes respecto a otros algoritmos. En los resultados de este trabajo ambos algoritmos proporcionaron buenos resultados para problemas con un número de ciudades pequeños, pero si el tamaño aumenta pueden no encontrar la ruta optima.

En la primera prueba los dos algoritmos encuentran el mejor camino para 10 y 20 ciudades, aunque la diferencia entre el promedio de RS (391.7068) en comparación con CLONALG (391.8037) es mínima, saber que RS encuentra la peor ruta con valor de 407.8863, indica que RS encuentra más veces una mejor ruta.

En la segunda prueba, RS es mejor en la instancia Att48 con 48 ciudades, RS encuentra la ruta más cercana al óptimo con valor de 33936.386, además su media (34398.193) es mejor en comparación con la de CLONALG (35727.915), indicando que encuentra mejores soluciones. Pero para bayg29 con 29 ciudades CLONALG sí encuentra la ruta optima (9074.1480), sin embargo, RS nunca encuentra la ruta optima.

El tiempo de ejecución y la distancia de la ruta aumentan con un número creciente de ciudades. El algoritmo de Recocido Simulado realiza muchos saltos buscando una mejor solución, ocasionando que a veces no encuentre la mejor solución, pero más soluciones en el espacio de búsqueda, ocasionando que encuentra buenos resultados cuando el número de ciudades es mayor.

En un futuro se pretende probar con más instancias, así como implementar alguna variante de los algoritmos originales. También se pretende comparar su desempeño con otros algoritmos heurísticos presentes en la literatura como el Algoritmo Luciérnagas o el Algoritmo Libélula. Por último, se quiere aplicar a problemas reales de la región de Oaxaca. Algunas aplicaciones del PAV en situaciones de la vida real son en horarios de transporte, en procesos de fabricación, en ordenar n trabajos y reparto de productos

por mencionar algunos. Con esta investigación se pretende encontrar el algoritmo más apropiado y eficiente para aplicarlo a problemas reales en municipios de la región.

Referencias

1. Alhanjouri-Mohammed, A.: Optimization techniques for solving travelling salesman problem. International Journal of Advanced Research in Computer Science and Software Engineering, 7(3), IJARCSSE, pp. 165−174 (2017)
2. Sumathi, M., Rahamathunnisa, U., Anitha, A., Das, D., Nallakaruppan, M.K.: Comparison of particle swarm optimization and simulated annealing applied to travelling salesman problem. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 8(6), pp. 1578−1583 (2019)
3. Adewole, A.P., Otubamowo, K., Egunjobi, T.O.: A comparative study of simulated annealing and genetic algorithm for solving the travelling salesman problem. International Journal of Applied Information Systems (IJAIS), 4(4), pp. 6−12 (2012)
4. Shimomura, M., Takashima, Y.: Application of monte-carlo tree search to traveling-salesman problem. In: 20th Workshop on Synthesis and System Integration of Mixed Information Technologies, (SASIMI) Proceedings, pp. 352−356 (2016)
5. Panda, M.: Performance comparison of genetic algorithm, particle swarm optimization and simulated annealing applied to TSP. International Journal of Applied Engineering Research, 13(9), pp. 6808−6816 (2018)
6. Zhou, A., Zhu, L., Hu, B., Deng, S., Song, Y., Qiu, H., Pan, S.: Traveling-salesman-problem algorithm based on simulated annealing and gene-expression programming. Information, 10 (2019)
7. Pang, W., Wang, K., Wang, Y., Ou, G., Li, H., Huang, L.: Clonal selection algorithm for solving permutation optimisation problems: a case study of travelling salesman problem. International Conference on Logistics Engineering, Management and Computer Science (LEMCS), Atlantis Press, pp. 575−580 (2015)
8. Muthreja, I., Kaur, D.: A comparative analysis of immune system inspired algorithms for traveling salesman problem. In: International Conference on Artificial Intelligence, ICAI'18, pp. 164−170 (2018)
9. Lin, Y., Bian, Z., Liu, X.: Developing a dynamic neighborhood structure for an adaptive hybrid simulated annealing–tabu search algorithm to solve the symmetrical traveling salesman problem. Appl. Soft Comput., 49, pp. 937–952 (2016)
10. Wang, Y.: Improving artificial bee colony and particle swarm optimization to solve TSP problem. In: International Conference on Virtual Reality and Intelligent Systems (ICVRIS), Changsha, pp. 179–182 (2018)
11. Myszkowski, P.B., Laszczyk, M., Dziadek, K.: Non-dominated sorting tournament genetic algorithm for multi-objective travelling salesman problem. In: Federated Conference on Computer Science and Information Systems (FedCSIS) (2019)
12. Wei, F., Chen, W., Hu, X., Zhang, J.: An empirical study on evolutionary algorithms for traveling salesman problem. In: 9th International Conference on Information Science and Technology (ICIST), pp. 273–280 (2019)
13. Li, X., Zheng, Y.: Artificial bee colony algorithm and its application in traveling salesman problems. In: Chinese Control and Decision Conference (CCDC), Nanchang, China, pp. 1–5 (2019)

14. Hammouri, A.I., Samra, E.T.A., Al-Betar, M.A., Khalil, R.M., Alasmer, Z., Kanan, M.: A dragonfly algorithm for solving traveling salesman problem. In: 8th IEEE International Conference on Control System, Computing and Engineering (ICCSCE), Penang, Malaysia, pp. 136–141 (2018)
15. Chandrawati, T.B., Sari, R.F.: A review of firefly algorithms for path planning, vehicle routing and traveling salesman problems. In: 2nd International Conference on Electrical Engineering and Informatics (ICon EEI), pp. 30–35 (2018)
16. Juneja, S.S., Saraswat, P., Singh, K., Sharma, J., Majumdar, R., Chowdhary, S.: Travelling salesman problem optimization using genetic algorithm. In: Amity International Conference on Artificial Intelligence (AICAI), Dubai, United Arab Emirates, pp. 264–268 (2019)
17. Jaradat, A., Matalkeh, B., Diabat, W., Solving traveling salesman problem using firefly algorithm and k-means clustering. In: IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), Amman, Jordan, pp. 586–589 (2019)
18. Flórez, E., Díaz, N., Gómez, W., Bautista, L., Delgado, D.: Evaluación de algoritmos bioinspirados para la solución del problema de planificación de trabajos. I+D Revista de Investigaciones, 11(1), pp. 141–155 (2018)
19. Vairamuthu, M.K., Porselvi, S., Balaji, D.H., Babu, J.R.: Artificial immune system algorithm for multi objective flow shop scheduling problem. International Journal of Innovative Research in Science, Engineering and Technology, 3(3), pp. 1391–1395 (2014)
20. Chandekar, N., Jayachandran Pillai, M.: A comparative study of GA and ACO for solving travelling salesman problem. International Journal of Mechanical and Production Engineering (IJMPE), 5(11), pp. 34–37 (2017)
21. Sureja, N.M., Chawda, B.V.: Random travelling salesman problem using SA. International Journal of Emerging Technology and Advanced Engineering, 2(4), pp. 621–624 (2012)
22. Taiwo, O.S., Mayowa, O.O., Ruka, B.K.: Application of genetic algorithm to solve traveling salesman problem. International Journal of Advance Research, (IJOAR), 1(4), pp. 27–46 (2013)
23. Dwivedi, V., Chauhan, T., Saxena, S., Agrawal, P.: Travelling salesman problem using genetic algorithm. International Journal of Computer Applications (IJCA) National Conference on Development of Reliable Information Systems, Techniques and Related Issues (DRISTI), pp. 25–30 (2012)
24. Elhaddad, Y.R., Sallabi, O.M.: A new hybrid genetic and simulated annealing algorithm to solve the traveling salesman problem. In: Proceedings of the World Congress on Engineering, WCE 2010, I (2010)
25. Revathi, M., Arthi, K.: Application of artificial immune system algorithms in dataset classification. International Journal of Innovative Research in Advanced Engineering (IJIRAE), 1(6), pp. 291–293 (2014)
26. Ortiz-Aguilar, L.D., Valadez, J.M., Soberanes, H.J., González, C.L., Ramírez, C.L., Soria-Alcaraz, J.A.: Comparativa de algoritmos bioinspirados aplicados al problema de calendarización de horarios. Research in Computing Science, 94, pp. 33–43 (2015)
27. Zhu, Y., Gao, S., Dai, H., Li, F., Tang, Z.: Improved clonal algorithm and its application to traveling salesman problem. IJCSNS International Journal of Computer Science and Network Security, 7(8), pp. 109–113 (2007)
28. Reinelt, G.: TSPLIB. http://comopt.ifi.uniheidelberg.de/software/TPLIB95/ (2020)