© Revista de Matemática: Teoría y Aplicaciones 2019 **26**(1) : 99–114
cimpa – ucr issn: 1409-2433 (Print), 2215-3373 (Online)
doi: https://doi.org/10.15517/rmta.v26i1.35528

MÉTODO DE BÚSQUEDA TABÚ PARA OPTIMIZACIÓN COMBINATORIA APOYADO CON EL SOFTWARE WOLFRAM MATHEMATICA

TABU SEARCH METHOD FOR COMBINATORIAL OPTIMIZATION SUPPORTED WITH WOLFRAM MATHEMATICA SOFTWARE

Erasmo López* Enrique Vílchez†

Received: 9/Mar/2018; Revised: 7/Nov/2018;
Accepted: 23/Nov/2018

*Colegio Técnico Profesional de Upala, Ministerio de Educación Pública, Upala, Costa Rica. E-Mail: erasmolopezlopez@gmail.com
†Escuela de Informática, Universidad Nacional de Costa Rica, Heredia, Costa Rica. E-Mail: enrique.vilchez.quesada@una.cr

Resumen

En este trabajo se presentan los resultados obtenidos de un algoritmo basado en la Búsqueda Tabú que fue programado utilizando el software comercial *Wolfram Mathematica*. En *Wolfram Language* se realizaron distintas implementaciones de instancias aleatorias y otras disponibles en la biblioteca TSPLIB, comparándolas posteriormente con los resultados provistos del mismo algoritmo en el ambiente de programación *Visual Basic 6.0*. Las mejoras que se obtuvieron obedecen a la estructuración de funciones prediseñadas que permitieron analizar específicamente dos aspectos: la optimización de la solución y su exploración en las vecindades donde ya se conocía la presencia del óptimo. Para ello, nos centramos en desarrollar una oscilación en la matriz *tabú* de manera análoga a lo que se aplica a las soluciones en donde se percibe que podría estar el óptimo global. Finalmente, se muestran resultados concluyentes que permiten observar el buen desempeño del programa *Wolfram Mathematica* para tratar este tipo de problemas, mediante la estructuración adecuada de sus funciones internas.

Palabras clave: software comercial; búsqueda tabú; oscilación; funciones internas; problema del agente viajero; solución óptima.

Abstract

In this paper we present the results obtained from an algorithm based on the Tabu Search that was programmed using the commercial software *Wolfram Mathematica*. In *Wolfram Language* different implementations of random instances and others available in the library TSPLIB were made, comparing them later, with the results provided with the same algorithm, in the programming environment *Visual Basic 6.0*. The improvements that were obtained are due to the structuring of predesigned functions that allowed specifically analyzing two aspects: the optimization of the solution and its exploration in the neighborhoods where the presence of the optimum was already known. For this we focus on developing an oscillation in the *tabu* matrix analogously to what is applied to solutions where it is perceived that the global optimum could be. Finally, conclusive results are shown that allow observing the good performance of the program *Wolfram Mathematica* to deal with this type of problems, through the proper structuring of its internal functions.

Keywords: commercial software; tabu search; oscillation; internal functions; traveling salesman problem; optimal solution.

Mathematics Subject Classification: 65N55.

1 Introducción

En este trabajo, se presentan una serie de resultados obtenidos a través del uso del software comercial *Wolfram Mathematica*, relacionados con la programación de funciones diseñadas con el propósito de implementar el algoritmo Búsqueda Tabú.

Un primer paso de la investigación fue elaborado con el propósito de corroborar resultados y tiempos obtenidos a través del empleo del lenguaje de programación *VB 6.0* durante los años 2011 y 2012. Un segundo avance, se centra en las mejoras de esos tiempos y la implementación de la oscilación en el manejo dinámico de la matriz *tabú*. Se ha buscado caracterizar las prácticas que constituyen el significado propio de la convergencia del algoritmo, tiempo y óptimo encontrado. Se hace la conjetura de que el uso del ordenador es simplemente un recurso poderoso en la velocidad de cálculo, aportando muy pocos elementos de transformación en la búsqueda de la solución, por lo que es fundamental dotar los insumos esenciales para que su desempeño sea optimizado.

Posteriormente, se diseñaron diversas funciones internas que contribuyeron con el ordenador para alcanzar descargas en tiempos más eficientes. Por último, la experimentación en las instancias y la evaluación subsiguiente de dichos resultados, permitió terminar de diseñar las propuestas de mejora sobre la función objetivo.

2 Metodología utilizada

El problema de optimización combinatoria propuesto, dada su categoría *NP-Hard* necesita ser abordado mediante métodos de resolución que posibiliten la salida de soluciones factibles en tiempos razonables. Las técnicas metaheurísticas han probado ser una alternativa válida en el tratamiento de este tipo de problemas. En este trabajo se ha aplicado una metodología de resolución basada en la heurística de Búsqueda Tabú (Tabu Search, TS) enriquecida con una técnica de oscilación estratégica (SO). A continuación se describen los aspectos fundamentales de los criterios estratégicos utilizados.

Basado en la experiencia computacional mostrada en [6] y los datos experimentales generados en los algoritmos allí planteados, se ha fijado un intervalo en donde la matriz *tabú* mueve el valor del parámetro que permite el número de prohibiciones que se toman en cuenta para elegir el admisible.

En la Figura 1, se muestra la forma en la que se manipulan los parámetros iniciales (iteraciones, longitud *tabú* y las diversificaciones) con el objetivo de lograr una solución convergente al valor óptimo. Al conocerse de antemano la mejor solución encontrada hasta el momento, se permitió al oscilador *tabú* mover sus restricciones en el intervalo señalado por [6].

```
OsciladorTabu[20, 0, 10, 10, 4, 5, 0]
```

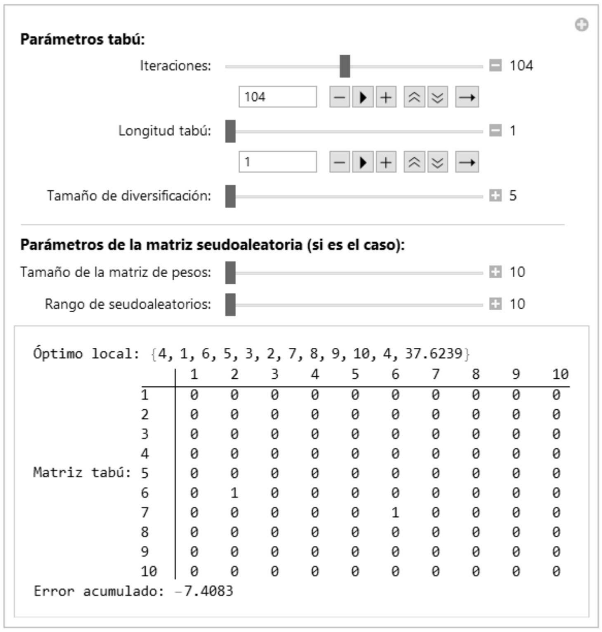

Figura 1: Escenario del oscilador.

El oscilador representado en la Figura 1, permite variar los parámetros involucrados en el método de optimización Tabú, arrastrando el deslizador correspondiente a saber: el número de iteraciones ejecutadas en el proceso, la longitud *tabú* del algoritmo y el tamaño de diversificación. En el comando `OsciladorTabu[20, 0, 10, 10, 4, 5, 0]`: el primer argumento indica el número de iteraciones que corresponden por defecto al valor ingresado multiplicado por diez; el segundo parámetro si es igual a cero genera una matriz de adyacencia de pesos seudoaleatoria del tamaño indicado en los dos parámetros siguientes (en lugar de cero es posible incluir directamente la matriz de adyacencia de pesos del problema a resolver); el argumento número cinco, donde aparece un 4 en este caso, corresponde a la longitud *tabú*; el valor siguiente es el tamaño de diversificación y el último argumento igual a cero, indica el inicio del proceso con la permutación trivial [10].

Para la instancia *gr17* disponible en la TSPLIB, el oscilador *tabú* debe moverse en el intervalo $\left[\frac{17}{2}-3, \frac{17}{2}+3\right] \approx [5, 11]$, de igual forma se estimaron los intervalos para las otras instancias mostradas en la Tabla 1, cuya primera columna representa el valor de la longitud *tabú* utilizada por [7] en su algoritmo programado en el ambiente *VB 6.0*.

La estrategia de oscilación modelada en la matriz *tabú* opera en el intervalo considerado en la Tabla 1, logrando gravitar la búsqueda cerca de la región donde se ha localizado el mejor admisible reportado. Si las restricciones *tabú* no se movieran en este intervalo, normalmente el algoritmo debería detenerse, pero en lugar de estancarse, el efecto oscilatorio en las restricciones *tabú* permite que las soluciones que podrían desecharse sean aceptadas, luego se pasa la frontera del máximo de movimientos prohibidos permitidos y nuevamente se regresa a él, creando el efecto oscilatorio en las restricciones [1].

También, se puede considerar esta forma de mover las restricciones *tabú* como un manejo dinámico en sus prohibiciones [1]. En un problema de n instancias (ciudades), si no hay mejoramiento, entonces el movimiento se considera *tabú* si:

$$\liminf \leq tabu(i,j) \leq \limsup,$$

donde:

$$\liminf = \frac{n}{2} - 3, \ \ \limsup = \frac{n}{2} + 3, \ \text{ con } n \geq 6.$$

Además, se utiliza la parte entera del límite superior e inferior con el objetivo de evitar resultados fraccionarios.

Instancia	Long. Tabú	Intervalo tabú Long. Interv.	Interv. Utiliz.
C.Carmen11	7	$\left[\frac{11}{2}-3, \frac{11}{2}+3\right]$	$[2, 8]$
Dos Bocas11	7	$\left[\frac{11}{2}-3, \frac{11}{2}+3\right]$	$[2, 8]$
gr17	11	$\left[\frac{17}{2}-3, \frac{17}{2}+3\right]$	$[5, 11]$
gr21	11	$\left[\frac{21}{2}-3, \frac{21}{2}+3\right]$	$[7, 13]$
gr24	13	$\left[\frac{24}{2}-3, \frac{24}{2}+3\right]$	$[9, 15]$
bayg29	11	$\left[\frac{29}{2}-3, \frac{29}{2}+3\right]$	$[11, 17]$
baygs29	15	$\left[\frac{29}{2}-3, \frac{29}{2}+3\right]$	$[11, 17]$
fri26	13	$\left[\frac{26}{2}-3, \frac{26}{2}+3\right]$	$[10, 16]$
gr48	27	$\left[\frac{48}{2}-3, \frac{48}{2}+3\right]$	$[21, 27]$
Berlin52	27	$\left[\frac{52}{2}-3, \frac{52}{2}+3\right]$	$[23, 29]$

Tabla 1: Estimación de los intervalos *tabú*.

3 Diseño de funciones internas

Wolfram Mathematica fue uno de los primeros programas de cálculo simbólico y numérico capaz de ejecutarse en diversos sistemas operativos. Se escribió en *C* en 1988 por el doctor *Stephen Wolfram* y desde entonces, se usa en numerosos campos de la ciencia y la tecnología, y también, ha tenido una buena acogida entre los estudiantes de carreras en las que las matemáticas son esenciales para su formación. Antes de mostrar el trabajo realizado en el presente artículo con el software *Mathematica*, conviene conocer algunas de sus potencialidades fundamentales [12].

No es fácil describir un programa como *Wolfram Mathematica*, aunque de una manera muy simplificada se puede decir que es un software destinado a la computación numérica, simbólica y gráfica, que ofrece herramientas interactivas de cálculo y un lenguaje de programación de alto nivel (denominado *Wolfram Language*).

La calculadora de tipo numérico tiene implementadas aproximadamente unas 750 funciones y además trabaja con la precisión que se desee (incluyendo precisión infinita).

El paquete de subrutinas para cálculo matemático permite hacer operaciones que requieran el uso de funciones o de procedimientos especiales como la integración numérica, la optimización de funciones, la programación lineal, entre otras, que se pueden utilizar directamente. En resumen, *Mathematica* ofrece potentes herramientas de cálculo simbólico y numérico, donde por ejemplo, es posible derivar e integrar funciones, resolver ecuaciones diferenciales, calcular límites, manipular series de potencias, utilizar matrices, entre otras múltiples opciones. Cuenta con un paquete gráfico para dibujar en dos o tres dimensiones, elegir perspectivas, sistemas de representación, sistemas de coordenadas y ejecutar con facilidad animaciones de distinta naturaleza.

El lenguaje que provee admite realizar programación en tres niveles: programación de tipo estructurada (uso de bloques, iteraciones y ciclos, recursiones, entre otras), programación funcional (con la posibilidad de definir funciones y operadores funcionales) y programación basada en reglas (suministrando reglas que indican cómo operar o transformar expresiones simbólicas y funciones).

3.1 La programación interna

Una de las características más importantes de *Wolfram Mathematica* reside en constituirse en un sistema computacional extensible. En el programa se conservan una cantidad importante de instrucciones ya diseñadas, que al ser reutilizadas en el contexto de su lenguaje de programación *Wolfram Language*, facilitan la tarea de añadir nuevas funciones al entorno. Para muchos tipos de cálculos, lo existente en la versión estándar de *Mathematica* será suficiente. Sin embargo, si se trabaja en un ámbito científico particular, donde el área de estudio o de investigación es de tendencia especializada, puede plantearse la consecuente necesidad de contar con funciones o comandos no incorporados por defecto en el software. En tales casos, se puede emplear o crear un *package* (paquete) externo de *Wolfram Mathematica* que contenga todas las funciones que se necesitan. Los paquetes de *Wolfram Mathematica* son archivos escritos en su lenguaje de programación. Los mismos consisten en colecciones de definiciones hechas en *Wolfram Language*, las cuales se aplican en áreas particulares.

Existen diversos tipos de sutilezas asociadas a esta clase de archivos (paquetes) de extensión *.m*. Por ejemplo, los conflictos que pueden suscitarse entre los nombres de las funciones que pertenecen a un mismo paquete u otras librerías del programa. Un aspecto importante en este sentido, consiste en el hecho de no invocar una función que se leerá desde un paquete antes de que esta sea cargada realmente en la sesión del *kernel* de *Wolfram Mathematica*. Si se hace esto por equivocación, se deberá ejecutar el comando `Remove[name]` para liberar de la memoria de corto plazo del ordenador la definición que se hizo antes de ejecutar el paquete. Si no se emplea `Remove`, *Mathematica* usará la versión de dicha instrucción (si existiera) alojada en el *kernel*, en sustitución del comando creado en el paquete. *Mathematica* es una aplicación que facilita extender sus bibliotecas por *default* usando paquetes de elaboración propietaria, lo cual lo hace una herramienta de software con partes relativamente ilimitadas. En lo que a la utilización concierne, no hay en realidad ninguna diferencia entre las funciones definidas en paquetes y las funciones incorporadas en *Mathematica*, desde un punto de vista técnico.

En la Figura 2, se muestra el paquete programado en *Wolfram Language* para efectos del presente trabajo. En las secciones posteriores se analizará el desempeño de sus funciones internas, diseñadas en la implementación del algoritmo basado en la Búsqueda Tabú.

3.2 Construcción y mejora del algoritmo tabú en *Wolfram Mathematica*

El algoritmo es una extensión de lo propuesto por [7]. En él se utiliza la memoria a corto y largo plazo, ya que hace uso de la matriz de frecuencias que expone [1], permitiendo la exploración de nuevas regiones liberando de esta manera los movimentos señalados como *tabú*.

Figura 2: Escenario del paquete.

La matriz *tabú* se emplea de manera dinámica usando una estrategia similar a la oscilación de la función objetivo. La matriz de frecuencias se construye al inicio del procedimiento y lleva la historia de los movimientos admisibles y también es la responsable de definir el criterio, aspiración o la búsqueda exhaustiva. Sin embargo, es importante aclarar que el método genera un conjunto de soluciones y se escoge de ellas, aquella cuya función de costo sea mejor. Además, para empezar las iteraciones, es necesario elegir una solución inicial mediante la permutación natural y a través de esta, se encuentran las posibles vecindades. Cada vecindad es explorada con detalle y se establece sobre sus elementos las restricciones necesarias con la intención de reducir el número de posibilidades en cada iteración. El método de restringir es semejante al expuesto por [1].

La diferencia entre este enfoque y el expuesto por [7], se observa en el tratamiento dado a la oscilación en la matriz *tabú* y de igual forma, la heurística que se emplea en este estudio, se clasifica como intuitiva y mejora a través de la experiencia almacenada. Toma como solución inicial la permutación natural construida por un algoritmo de inserción, la $longitudTabu$ es el número máximo de prohibiciones que se establecen sobre la posición de un determinado nodo. $CiclosDiver$ es un parámetro que le permite al algoritmo revisar el número de iteraciones en las que el óptimo local no se ha actualizado, facultando de esta forma la diversificación para explorar nuevas regiones.

El algoritmo básico tratado con *VB 6.0* se muestra a continuación:

Algoritmo 3.1 *EraDeterminístico*
Inicialización

1. *Ingresar: MatrizDistancia, longitudTabú. MaxIter, CiclosDiver, ListaTabú.*
2. *Generar la solución inicial (permutación natural) y la distancia d(x).*
3. *Mientras Iter* $\leq$ *MaxIter.*
   - 3.1. *Actualizar MatrizTabu.*
   - 3.2. *Menor distancia* $\leftarrow d(x)$.
   - 3.3. *Generar las* $n-1$ *permutaciones.*
   - 3.4. *Determinar la distancia de cada permutación.*
   - 3.5. *Clasificar los movimientos en prohibidos y no prohibidos.*
   - 3.6. *Determinar la menor distancia de los movimientos no prohibidos y seleccionar la permutación.*
   - 3.7. *Si todos los movimientos son prohibidos, aplicar aspiración. Si no parar.*
   - 3.8. *Actualizar* $d(x)$.
   - 3.9. *Si* $d(x) < Op(x)$, *entonces actualizar* $Op(x)$. *Pasar a la instrucción 3.1.*
   - 3.10. *Si en* k *iteraciones no se actualiza* $Op(x)$, *entonces diversificar. Pasar a la instrucción 3.1.*
4. *El óptimo encontrado es* $Op(x)$. *Fin.*

Agregando la oscilación en la matriz *tabú* el nuevo algoritmo corresponde a:

Algoritmo 3.2 *EraDeterminístico*
Inicialización

1. *Ingresar: MatrizDistancia, longitudTabú. MaxIter, CiclosDiver, ListaTabú.*
2. *Generar la solución inicial (permutación natural) y la distancia d(x).*
3. *Mientras Iter* $\leq$ *MaxIter.*
   - 3.1 . *Actualizar MatrizTabu.*

3.1.1. Oscilar tabú.

3.1.2. Actualizar MatrizTabu.

3.2. Menor distancia $\leftarrow d(x)$.

3.3. Generar las $n-1$ *permutaciones.*

3.4. Determinar la distancia de cada permutación.

3.5. Clasifique los movimientos en prohibidos y no prohibidos.

3.6. Determinar la menor distancia de los movimientos no prohibidos y seleccionar la permutación.

3.7. Si todos los movimientos son prohibidos, aplicar aspiración. Si no parar.

3.8. Actualice $d(x)$.

3.9. Si $d(x) < Op(x)$*, entonces actualice* $Op(x)$*. Pasar a la instrucción 3.1.*

3.10. Si en k *iteraciones no se actualiza* $Op(x)$*, entonces diversificar. Pasar a la instrucción 3.1.*

4. El óptimo encontrado es $Op(x)$*. Fin.*

El método anterior funciona de manera iterativa. Una vez cumplido el criterio de parada, se necesita evaluar la solución inicial y la resultante, además del manejo de la matriz *tabú* y los parámetros asociados a ella, como lo es el tamaño del rango de oscilación y el número de iteraciones consumadas, a fin de observar la mejora. Si el valor es inferior al inicial, se considera que se ha encontrado un nuevo elemento mínimo y además, se almacena en una lista llamada $OptimoLocal$, nombre dado a la trayectoria mínima. También, la lista de nodos en el orden en que se da esta condición es retenida en una variable [6].

4 Resultados numéricos

La modificación en el algoritmo se ejecutó para varias instancias del *Problema del Agente Viajero* (PAV) cuyas distancias euclideanas se representan por una matriz $n \times n$. Los valores recibidos en los tiempos de salida se compararon con los de [7] a fin de observar la diferencia entre el manejo de la matriz *tabú* sin la oscilación y en los resultados obtenidos en presencia de ella. Estos datos fueron utilizados para fijar los límites de prohibición superior e inferior para la oscilación estratégica. Los valores propuestos en la oscilación de la matriz *tabú* fueron: $\left[\frac{n}{2}-3, \frac{n}{2}+3\right]$ [7], donde n es el tamaño de la instancia de cada problema explorado.

Con el algoritmo expuesto, se realizó el tratamiento del PAV, aplicando los procesos a los casos particulares mencionados y en forma resumida, los resultados producidos al compararlos con algunos disponibles en la TSPLIB.

4.1 Comparación con la TSPLIB

Con base en las observaciones expuestas en la sección anterior, se decidió realizar el experimento a las poblaciones donde se conoce con antelación las soluciones óptimas para cada instancia tratada. Con esta finalidad, se utilizaron resultados disponibles en la TSPLIB [8].

Para comparar los resultados alcanzados en los experimentos, se aplicó la medida de la eficiencia utilizando la ecuación [1]:

$$e = 1 - \frac{Z - Z_{opt}}{Z_{opt}}.$$

En la Tabla 2, se observa que en muchos casos la eficiencia está muy cercana a 1, lo que nos indica que el algoritmo propuesto retorna resultados satisfactorios y de no ser así, este da la oportunidad de dirigir la búsqueda hacia nuevas regiones mediante la diversificación o a través de una permutación inicial que no sea la natural.

Instancia	Resultados Literatura y López					Resultados con oscilación			e
	Long. Tabú	Long. Diver.	Num. Iter.	Opt. Lit.	Opt. López	Long. Inic.	Opt. Enc.	Tiempo (segundos.)	
C.Carmen11	7	500	3300	421,21	421,21	437,84	421,21	0,491715	1,00
Dos Bocas11	7	2	2	464,15	464,15	479,60	464,00	0,004655	1,00
gr17	11	600	4000	2085,00	2085,00	7722,00	2085,00	39,78850	1,00
gr21	11	600	1500	2707,00	2863,00	6285,00	2863,00	235,7460	0,94
gr24	13	600	2700	1272,00	1465,00	3513,00	1377,00	27,91560	0,92
bayg29	11	600	1100	1610,00	2063,00	4718,00	1868,00	77,45500	0,84
baygs29	15	10000	15000	2020,00	2901,00	5601,00	2311,00	190,5570	0,86
fri26	13	40	100	937,00	944,00	1100,00	944,00	0,309869	0,99
gr48	27	20	1000	5046,00	6001,00	18806,00	6001,00	8,178920	0,81
Berlin52	27	100	1000	7542,00	8741,00	21768,00	8741,00	9,735260	0,84

Tabla 2: Comparación con la literatura y López.

4.2 Comparación con *VB 6.0*

Los esfuerzos computacionales que conllevan la implementación de un algoritmo de fuerza bruta, generan que los ordenadores utilicen mucha de su arquitectura en busca de las mejores soluciones. La idea de construir un algoritmo basado en la experiencia computacional obtenida empleando el ambiente de programación *VB 6.0* y cuyos resultados se observan en [7], emerge al realizar una lectura de distintas instrucciones que optimizan los tiempos de ejecución mediante la programación de funciones internas y el diseño de paquetes que se instalan en la arquitectura del sistema operativo, utilizando como plataforma de programación el software *Wolfram Mathematica* [9]. Los resultados de este algoritmo implementado en *VB 6.0* se muestran en la Tabla 3, añadiendo una comparación con otra implementación programada en *Wolfram Language*. Como se aprecia, los recorridos en *Wolfram Mathematica* son aún mejores de los ya obtenidos por [7].

Cabe destacar que cuando se decidió utilizar esta plataforma (*Wolfram Mathematica*), no se tenía claro si los tiempos de ejecución iban a mejorar, sin embargo, intuitivamente los comandos ya integrados en el software [12], dieron un indicio de lo prometedor que resultaría si se utilizaban las estructuras ya programadas con el objetivo de optimizar los procedimientos rutinarios al correr el algoritmo.

Instancia	Opt. Lit.	Resultados con *VB 6.0* Tiempo (seg.)	Num. Iter.	Long. Tabú.	Opt. López	Resultados con *Mathematica* Long. Inic.	Opt. Enc.	Tiempo (seg.)	e
C.Carmen11	421,21	0,546	3300	7	421,21	437,84	421,21	0,026	1,00
Dos Bocas11	464,15	0,000	2	7	464,15	479,60	464,00	0,000	1,00
gr17	2085,00	3,867	4000	11	2085,00	7722,00	2085,00	1,583	1,00
gr21	2707,00	1,094	1500	11	2863,00	6285,00	2863,00	0,004	0,94
gr24	1272,00	1,986	2700	13	1465,00	3513,00	1377,00	0,006	0,92
bayg29	1610,00	1,161	1100	11	2063,00	4718,00	1868,00	0,001	0,84
baygs29	2020,00	14,892	15000	15	2901,00	5601,00	2311,00	7,782	0,86
fri26	937,00	0,093	100	13	944,00	1100,00	944,00	0,000	0,99
gr48	5046,00	7,747	1000	27	6001,00	18806,00	6001,00	3,421	0,81
Berlin52	7542,00	8,841	1000	27	8741,00	21768,00	8741,00	3,265	0,84

Tabla 3: Comparación con TSPLIB, de los Cobos y López.

Los resultados mostrados en la Tabla 3, demuestran el buen desempeño trazado en todas las instancias, siendo las más representativas: Dos Bocas, **C. Carmen**, *gr24* y *fri26*. Las otras instancias se comportaron de manera similar. Aunque las soluciones no mejoraron, sí lo hicieron los tiempos de ejecución, al determinar la misma respuesta óptima.

4.3 Comparación con el comando AgenteViajero[]

La instrucción AgenteViajero[] resuelve el *Problema del Agente Viajero* sobre un grafo *G* simple, no dirigido creado en el *Wolfram System*, o bien, con el paquete *Combinatorica* de *Wolfram Mathematica*. Es decir, el comando encuentra en caso de existir, un circuito de Hamilton de longitud más corta. Si el grafo no es ponderado, la longitud se asume en función de la cantidad de aristas involucradas. Presenta la opción "ruta $\longrightarrow$ True" que muestra adicionalmente una animación del circuito encontrado. Sintaxis: AgenteViajero[G], o bien, AgenteViajero[G, ruta $\longrightarrow$ True]. La instrucción retorna la longitud del circuito y su recorrido representado mediante una secuencia de aristas [11].

En esta sección se emplea el comando de software AgenteViajero, el cual forma parte de un paquete escrito en el lenguaje *Wolfram*, llamado por su autor: *VilCretas* [10]. AgenteViajero resuelve el *Problema del Agente Viajero* utilizando internamente funciones propias del software *Wolfram Mathematica* y aunque se desconoce el algoritmo que lo caracteriza, resultó interesante en la trama del presente trabajo, comparar los resultados arrojados en correspondencia con lo programado por los autores de esta propuesta.

Instancia	Opt. Lit.	Resultados con *Math. AgenteViajero[]* Tiempo (seg.)	Solución Encontrada	Resultados con *Math. TS* Tiempo (seg.)	Solución Encontrada
C.Carmen11	421,21	0,263870	421,21	0,491715	421,21
Dos Bocas11	464,15	0,237668	464,15	0,004655	464,00
gr17	2085,00	0,268222	2085,00	39,7885	2085,00
gr21	2707,00	0,277437	2707,00	235,746	3246,00
gr24	1272,00	0,402047	1272,00	27,9156	1377,00
bayg29	1610,00	0,409893	1627,00	77,455	1868,00
baygs29	2020,00	0,428600	2020,00	190,557	2311,00
fri26	937,00	0,354873	937,00	0,309869	944,00
gr48	5046,00	1,474920	5039,00	8,17892	6001,00
Berlin52	7542,00	1,652040	7526,00	9,73526	8741,00

Tabla 4: Comparación con Math. AgenteViajero[].

La sentencia AgenteViajero[] fue creada utilizando un comando propietario de *Wolfram Mathematica* llamado: FindShortestTour. AgenteViajero[] facilita el procesamiento de grafos creados o no en el *Wolfram System*, además de permitir la generación de una animación con un circuito hamiltoniano de costo mínimo. Estas características hacen de AgenteViajero[] un comando único disponible en el paquete *VilCretas*, biblioteca que puede ser descargada gratuitamente desde: http://www.escinf.una.ac.cr/discretas/index.php/archivos/category/7-packages.

Al estudiar los resultados arrojados por el algoritmo implementado en *Wolfram Mathematica* y el que caracteriza a la instrucción AgenteViajero[] de *VilCretas*, se derivan conclusiones satisfactorias, en diez ejecuciones mediante una comparación con la TSLIB. El análisis de la eficiencia sobre AgenteViajero no se muestra en la Tabla 4, pues para estas instancias ella es igual a 1 y en los resultados de la propuesta ya están considerados dentro de las Tablas 2 y 3. Si se observa que los tiempos con TS son grandes con respecto a la instancia tratada, se debe a que el algoritmo se implementó en su totalidad, esto es, utilizando la oscilación en la matriz *tabú*.

5 Conclusiones

El presente trabajo ha permitido obtener varios resultados importantes que ubican a *Wolfram Mathematica* como un software muy prometedor en el tratamiento de problemas combinatorios, basados en la investigación práctica y en la optimización de la solución. A continuación se señalan cada uno de ellos.

El primer logro esencial se puede observar en las facilidades que ofrece el software *Wolfram Mathematica*, donde al recurrir a sus funciones internas, se minimizan los tiempos de cálculo en aquellas partes de sumas sucesivas necesarias para determinar la longitud de cada recorrido en las instancias exploradas.

Aunque existen muchos otros programas comerciales en los que se puede tratar el *Problema del Agente Viajero*, el aquí utilizado desempeñó un papel lógico y oportuno en los tiempos de ejecución, como se muestra en la Tabla 3.

Emplear los comandos nativos de *Wolfram Mathematica* se ha constituido en una herramienta muy versátil al promover una fácil integración con bases de datos, hojas de cálculos y archivos de *Excel*. La confianza que se puede tener en el lenguaje de *Wolfram Mathematica* se ve reflejada en las pruebas realizadas a las instancias donde ya se conocía el óptimo global, cuya eficiencia medida con la ecuación considerada por [1] es muy cercana a 1.

El tratamiento dinámico de la matriz *tabú* que se utilizó, considerando los intervalos de la Tabla 1, permitió que el algoritmo se moviera en una frontera de prohibiciones cuyas soluciones estaban muy cerca del óptimo global.

El propósito principal de programar en *Wolfram Language* el algoritmo Búsqueda Tabú y compararlo con la implementación de [6] en *VB 6.0*, fue alcanzado. Los tiempos de cálculo sobrepasan en buena medida (Tabla 3) los reportados por [7] en el tratamiento de las mismas instancias con los mismos parámetros.

Las funciones programadas en *Wolfram Language* por los autores de este artículo, se integraron en un paquete de software denominado: BusquedaTabu. Este *package* se encuentra disponible en la siguiente dirección *URL*: http://www.escinf.una.ac.cr/discretas/index.php/archivos/category/7-packages.

Dentro de un trabajo futuro se propone realizar un sistema más complejo que explore otros métodos de optimización combinatoria, como lo son los algoritmos genéticos, hormigas, enjambre o recocido simulado.

Agradecimientos

En agradecimiento a la dirección de la Escuela de Informática de la Universidad Nacional de Costa Rica. El presente artículo es un resultado alterno del proyecto de investigación titulado: *VilCretas un recurso didáctico a través del* *uso del software Mathematica para el curso EIF-203 Estructuras Discretas para Informática*, código SIA: 0080-15.

Referencias

[1] De los Cobos, S.; Goddard, J.; Gutiérrez, M.; Martínez, A. (2010) *Búsqueda y Exploración Estocástica*. Universidad Autónoma Metropolitana, México.

[2] Glover, F. (1989) "Tabu search, part I", *ORSA Journal on Computing* **1**(3): 190–206.

[3] Glover, F. (1990) "Tabu search, part II", *ORSA Journal on Computing* **2**(1): 4–31.

[4] Glover, F.; Laguna M. (1993) "Tabu search", in: Colin R. Reeves (Ed.) *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford: 70–150.

[5] Glover, F.; Melián, B. (2003) "Búsqueda tabú", *Revista Iberoamericana de Inteligencia Artificial* **7**(19): 29–48.

[6] López, E. (2011) *El Agente Viajero: Un Algoritmo Determinístico*. Tesis de Licenciatura en la Enseñanza de Matemática, Facultad de Ciencias Exactas y Naturales, Universidad Nacional, Heredia, Costa Rica.

[7] López, E.; Salas, O.; Murillo, A. (2014) "El problema del agente viajero: un algoritmo determinístico usando búsqueda tabú", *Revista de Matemática: Teoría y Aplicaciones* **21**(1): 127–144.

[8] Reinelt, G. (2004) "TSPLIB, travelling salesman problem", en `https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`

[9] Vílchez, E. (2012) *Álgebra Lineal Apoyada con Mathematica*. Editorial Tecnológica de Costa Rica, Cartago, Costa Rica.

[10] Vílchez, E. (2016) "VilCretas package: educational resource through the use of Mathematica software in the field of discrete mathematics", in: *Wolfram Technology Conference 2016*, Champaign, Illinois.

[11] Vílchez, E. (2018) *Matemática Discreta a Través del Uso del Paquete VilCretas*. Revista Digital Matemática, Educación e Internet, Costa Rica.

[12] *Wolfram Mathematica 11*: Documentation Center. (2017). *Mathematica* functions and tutorials. Recuperado de: http://reference.wolfram.com/mathematica/guide/Mathematica.html