Ingeniare. Revista Chilena de Ingeniería
ISSN: 0718-3291
facing@uta.cl
Universidad de Tarapacá
Chile

Cockbaine Ojeda, Juan; Silva Urrea, Rubén
Perfeccionando algoritmos heurísticos para el problema NP-C E-TSP
Ingeniare. Revista Chilena de Ingeniería, vol. 21, núm. 2, agosto, 2013, pp. 196-204
Universidad de Tarapacá
Arica, Chile

Perfeccionando algoritmos heurísticos para el problema NP-C E-TSP

Improving heuristic algorithms for NP-C E-TCP problem

Juan Cockbaine Ojeda[1] Rubén Silva Urrea[2]

Recibido 19 de enero de 2012, aceptado 8 de marzo de 2013
Received: January 19, 2012 Accepted: March 8, 2013

RESUMEN

El desarrollo de algoritmos heurísticos eficientes y exactos, de orden polinomial, que logren buenas soluciones para problemas complejos pertenecientes a la clase NP-C, continúa siendo un gran reto. Se propone una estrategia para perfeccionar algoritmos heurísticos, específicamente para el problema del vendedor viajero euclidiano, conocido como E-TSP. Se define un conjunto de indicadores que informan en qué medida la heurística del programador se refleja en el algoritmo. La retroalimentación obtenida, al utilizar los indicadores, facilita el proceso de mejora del algoritmo inicial, lográndose mejores soluciones en promedio. Es factible obtener retroalimentación desde la propia ejecución del algoritmo heurístico y basándose en tal información perfeccionar el algoritmo.

Palabras clave: Indicadores, E-TSP, algoritmos heurísticos, NP-C, ambientes TEL.

ABSTRACT

Development of efficient and accurate heuristic algorithms of polynomial order, which achieve good solutions to complex problems in class NP-C, remains a great challenge. A strategy is proposed to improve heuristic algorithms, specifically for the euclidean traveling salesman problem, known as E-TSP. A defined set of indicators informs the extent to which programmed heuristics is reflected in the algorithm. The feedback obtained through indicators facilitates the process of refining the initial algorithm, in order to achieve better solutions.

Keywords: Indicators, E-TSP, heuristic algorithms, NP-C, TEL environments.

INTRODUCCIÓN

Existen problemas para los cuales no es posible encontrar una solución óptima en un tiempo razonable, circunstancia que justifica el desarrollo y perfeccionamiento de algoritmos capaces de abordar ese tipo de problemas que pertenecen a la clase NP-C [1]. Aquellos algoritmos son llamados heurísticos.

En este contexto, los problemas son tratables o del tipo P para los que existe un algoritmo de complejidad polinomial capaz de resolverlos; o son problemas NP para los cuales no se conoce un algoritmo de complejidad polinomial capaz de resolverlos. Adicionalmente existe un subconjunto de la familia NP, llamado NP-C, que presenta un mayor grado de dificultad de resolución [6-7].

Entre los problemas que se pueden resolver en tiempo polinomial se encuentran diversas variedades de orden, como los logarítmicos (log(n)), los lineales (n), los cuadráticos (n^2), los cúbicos (n^3). En los problemas NP la complejidad de los algoritmos es del tipo exponencial (2^n) o factorial (n!), donde si se aumentan las variables de entrada (n), el tiempo de resolución del problema pudiese crecer

[1] Departamento de Ingeniería Informática. Universidad de Santiago de Chile. Av. Ecuador 3659 Santiago, Chile. E-mail: juancarlos.cockbaine@usach.cl
[2] Área de Sistemas. Academia Politécnica Militar. Valenzuela Llanos 623. Santiago, Chile. E-mail: r_silva@acapomil.cl

significativamente. En general estos problemas se reducen a maximizar o minimizar una función objetivo sujeta a restricciones, por ejemplo, encontrar un camino óptimo, o de costo mínimo, al recorrer un conjunto de ciudades pasando solamente una vez por cada ciudad.

En la actualidad no se conoce un algoritmo capaz de encontrar soluciones óptimas para la familia de problemas pertenecientes a la clase NP-C, esto es, encontrar óptimos en tiempo polinomial, utilizando una máquina determinística, para instancias complejas.

Una instancia es la especificación de un problema en particular que pertenece a una familia de problemas, por ejemplo, la instancia indica la ubicación de ciudades y costo que demanda recorrerlas, otra instancia indicará otras ciudades y costos. Este tipo de especificación corresponde a instancias del Problema NP-C del Vendedor Viajero (PVV) o Traveling Salesman Problem (TSP) en inglés.

Es necesario considerar, que el conjunto de soluciones (rutas factibles no necesariamente óptimas) puede crecer exponencialmente en relación con el tamaño del problema que es especificado por una instancia. Luego, podría resultar imposible implementar un algoritmo que encuentre soluciones óptimas en tiempo polinomial. Por ello, se utilizan métodos aproximados, como es el desarrollo de heurísticas que acercan a una solución óptima, factible y de utilidad práctica. En contraposición a los métodos exactos, que proporcionan una solución óptima al problema, los métodos heurísticos se limitan a entregar una buena solución pero que no es necesariamente la más adecuada.

Para los objetivos de este trabajo se considera uno de los problemas pertenecientes a la familia NP-C, el E-TSP. Un TSP se puede representar por un grafo, G = (V, A) en que V es el conjunto de vértices y A es el conjunto de aristas. Se asocia a G una matriz de costos C, en que cada elemento c_{ij} representa el costo o distancia de la arista (i, j). Un tour (T) o Ciclo Hamiltoniano es un ciclo simple que pasa por todos los vértices del grafo. El problema consiste en determinar un T de costo mínimo. Para El E-TSP se define el costo c_{ij} como la distancia euclidiana entre dos puntos (i, j) en un plano 2D. Los costos de esta ecuación son simétricos, por lo tanto: d(i, j) = d(j, i) además d(i, j) + d(j, k) ≥ d(i, k).

En este trabajo se utiliza un algoritmo desarrollado en el contexto de un proyecto de investigación aplicada en optimización, en el Departamento de Ingeniería Informática de la USACH. La utilización de este algoritmo es suficiente para mostrar el propósito de este trabajo. El algoritmo heurístico utilizado logra soluciones cercanas a óptimos conocidos [2] para el conjunto de instancias reportadas en Tabla 1 (comparar las columnas "Óptimo" y "V1").

Con el fin de explicar parte de la heurística implementada, la Figura 1 ilustra como el algoritmo logra un tour para la instancia 20puntos10interiores: a) identifica grupos de ciudades cercanas, b) fija las ciudades que bordean a otras ciudades (casco convexo), c) une las ciudades al interior de un grupo que se conecta con una ciudad perteneciente al casco convexo y d) procede a unir otro grupo, conectándolo a otra ciudad del casco convexo, logrando finalmente un tour.

TSPLIB [8], sitio de dominio público, provee instancias, soluciones obtenidas hasta la fecha y, para algunos casos, su solución óptima. La Figura 2 ilustra la forma en que se representa una instancia E-TSP.

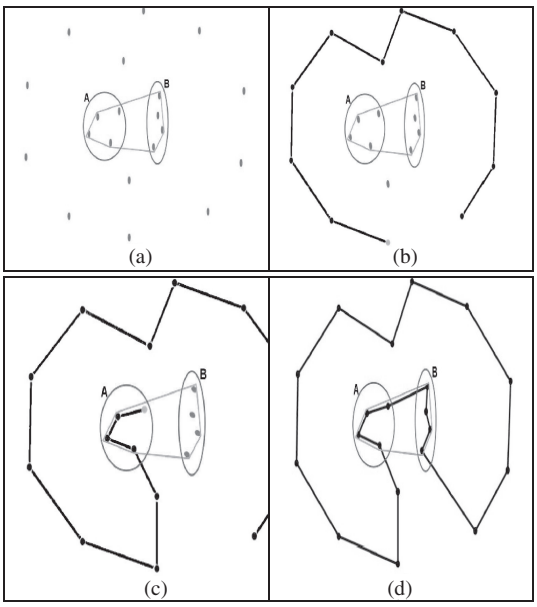

Figura 1. Heurística aplicada a una instancia de problema del E-TSP.

En la Figura 2 NAME indica el nombre de la instancia, TYPE o tipo genérico de problema, COMMENT indica la solución óptima conocida hasta

la fecha, DIMENSION es la cantidad de puntos del problema, en este caso 5, EDGE_WEIGHT_TYPE para este caso euclidiano de dos dimensiones, NODE_COORD_SECTION lista de coordenadas en el plano representando la ubicación de las ciudades.

```
NAME : 5puntos1
TYPE : TSP
COMMENT : 5 puntos - Solución Óptima: 2036
DIMENSION : 5
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 982   377
2 682   385
3 037   111
4 841   474
5 615   288
EOF
```

Figura 2. Instancia E-TSP 5puntos1.

El algoritmo inicial que se decidió utilizar fue probado con 64 instancias extraídas de TSPLIB [8], Rooij [5], Mac Gregor [4] y otras que se generaron aleatoriamente. El algoritmo obtuvo un 6,9% de desviación (en promedio) respecto de los óptimos conocidos. El porcentaje representa la calidad de la solución y corresponde a la razón entre el costo del circuito encontrado por el algoritmo y el costo de las soluciones óptimas conocidas.

Si bien el objetivo de este artículo no es el desarrollo de algoritmos para solucionar el problema NP-C E-TSP, la estrategia propuesta se alinea con ese gran objetivo, esto es, se busca mejorar el promedio obtenido por un algoritmo heurístico, respecto del promedio de los óptimos conocidos para un conjunto de instancias.

Uno de los aspectos relevantes de la estrategia es la construcción de indicadores cuya fuente de datos pueda ser generada automáticamente. En [3] se desarrolla y aplica el aporte de indicadores basados en fuentes de datos generados por un ambiente TEL, Technology Enhanced Learning. En este caso, el propio algoritmo heurístico, al momento de ejecutarse, debe generar los datos necesarios. Posteriormente, al analizar los resultados arrojados por los indicadores, que han transformado dichos datos en información, es factible intervenir el algoritmo con el objeto de perfeccionarlo, de la forma en que se explica en las secciones posteriores de este trabajo. Lógicamente se busca que los algoritmos perfeccionados encuentren mejores soluciones en promedio, que sus versiones anteriores.

La estrategia propuesta requiere ejecutar versiones del algoritmo heurístico, tabular y facilitar la comparación de resultados, administrar versiones del algoritmo y sus instancias de prueba, representar gráficamente resultados obtenidos por los indicadores formulados, entre otras funcionalidades que son necesarias para cada una de las etapas que en la sección "Estrategia" de este trabajo se presentan. Las necesidades indicadas demandaron el desarrollo de un "Ambiente de Pruebas" para facilitar tales tareas.

TRABAJOS RELACIONADOS

Existen diversas implementaciones de heurísticas para lograr buenas soluciones para el TSP [9-12, 26]. También la computación evolutiva, que ha inspirado el desarrollo de algoritmos genéticos y la programación genética [20] como una técnica generadora de código computacional que se basa en el autoaprendizaje e instancias de un problema. Por otro lado las hiperheurísticas que escogen un conjunto de heurísticas de bajo nivel, utilizando algún mecanismo de aprendizaje y adaptando métodos heurísticos o metaheurísticos [18].

Desde hace tiempo se puede observar el interés en el problema: Dantzing [15]; Lin [21]; Lin & Kernighan [27]; Aarts, Korst, & Van Laarhoven [14]; E. Goldbarg, M. Goldbarg, & Farias, [16]; Potvin [23]; Knox [17]; Zhu & Chen [13]; Neumann [22]; Rasmussen [24] entre otros. Aún así se continúan los esfuerzos por crear heurísticas para la obtención de buenas soluciones en tiempos razonables.

Se observa que no es inmediato deducir con precisión si una heurística específica, para un conjunto de instancias, está actuando como su creador pensó que debería hacerlo, esto es, muchas son cajas negras con algún grado de parametrización o afinamiento que arroja un resultado, en este caso los valores cercanos a óptimos encontrados. En la literatura no se observan estrategias claras para intervenir el código de la heurística e introducir indicadores debidamente formulados y programados en puntos precisos cuya ejecución entregue información para facilitar la perfección de la heurística.

Del teorema no-free-lunch (NFL) [25] se puede afirmar que hay instancias de un problema en que un algoritmo es mejor que otro para un subconjunto

de esas instancias y el otro algoritmo es mejor que el primero para otro subconjunto de instancias, o sea, no podría existir un par de algoritmos que resuelva todos los problemas y sea uno superior al otro. Se interpreta de este teorema que la selección de una mejor heurística no es una tarea trivial, pues es necesario conocer el tipo de problema y el algoritmo. Una forma de conocer el algoritmo es logrando información del mismo, con el objeto de perfeccionarlo y facilitar la decisión respecto de que algoritmo escoger. Se puede aventurar entonces que la construcción de heurísticas computacionales continuará siendo una estrategia útil y práctica pero necesaria de perfeccionarla constantemente, estrategia que aborda este trabajo.

ESTRATEGIA

La estrategia se basa en generar datos en forma automática, a partir de la propia ejecución del algoritmo que se desea mejorar, específicamente se desea generar un registro de trazas de ejecución que servirá de fuente de datos para el desarrollo de indicadores. Estos indicadores aportarán con información para el desarrollo de versiones posteriores del algoritmo.

La estrategia requiere un algoritmo heurístico desarrollado en algún lenguaje de programación, una especificación escrita en lenguaje natural describiendo la heurística para abordar el problema NP-C, en este caso el E-TSP y el seudocódigo en que se basó la programación del algoritmo. Las etapas son las siguientes:

1. **Análisis de la especificación de la heurística**
   De la especificación de la heurística se seleccionan oraciones ambiguas que admiten distintas interpretaciones y que generan dudas o que expresan alguna afirmación o negación.

2. **Introducir comentarios en el seudocódigo**
   Se introducen comentarios en el seudocódigo para facilitar la comprensión de la heurística. La importancia de esto es facilitar la detección de inconsistencias entre código y seudocódigo.

3. **Mejorar legibilidad del algoritmo heurístico**
   El desarrollo de un algoritmo heurístico es una tarea difícil y aunque este proceso haya sido apoyado por una metodología de desarrollo de *software*, además de incluir buenas prácticas para el desarrollo, posiblemente no se obtendrá un código fácil de leer e intervenir.

   En esta etapa no se interviene la heurística programada, sino que se focaliza en lograr procedimientos más cohesionados y menos acoplados entre sí, de tal forma que se facilite la integración de estos en un procedimiento principal que los invoque. Además se aplican normas respecto de la definición de variables y constantes, según sea el lenguaje de programación utilizado.

4. **Unificación de seudocódigo y código fuente**
   Se unifica seudocódigo y código fuente identificando correspondencia entre ellos.

5. **Incorporación de función generadora de datos**
   Se adiciona en el código fuente, llamadas a una función generadora de trazas de ejecución.

   Estas llamadas se insertan en el código, en donde se requiera verificar algún aspecto de la heurística. Por ejemplo, si la heurística establece que en todo momento es necesario considerar el casco convexo antes de conectar ciudades entre sí, entonces antes de invocar el procedimiento que conecta dos ciudades, se inserta el llamado a la función. De esta forma, se podrá conocer las veces que fue considerado el casco convexo con relación a las ciudades de la instancia.

   La función generadora de datos registra el momento en que se ejecutan las instrucciones y la secuencia de ejecución.

6. **Ejecución del algoritmo intervenido**
   Se ejecuta el algoritmo intervenido (intervenido por los pasos anteriores), con el objeto de obtener los datos que serán la fuente para los indicadores y se verifica que las soluciones encontradas por este algoritmo sean las mismas que su versión inicial, asegurando que no se haya intervenido accidentalmente la heurística.

7. **Construcción y análisis de indicadores**
   Se construyen los indicadores teniendo en cuenta los datos generados por la ejecución del algoritmo intervenido. La información

que arrojen los indicadores permitirá decidir la implementación de cambios en la heurística del algoritmo.

8. **Generación de una nueva versión del algoritmo** En base a la información entregada por los indicadores, se modifica el código fuente del algoritmo. A continuación se ejecuta la nueva versión y se comparan resultados anteriores con los nuevos resultados para un mismo conjunto de instancias.

En la próxima sección se ejemplifica la aplicación de la estrategia presentada.

APLICACIÓN DE LA ESTRATEGIA

En el primer paso se escogieron tres especificaciones utilizadas en la construcción del algoritmo heurístico, estas son: 1) Constantemente se busca enlazar las ciudades internas del problema con aquellas que componen el casco convexo, 2) se procura armar un circuito que contenga el grupo en su totalidad y 3) en la mayoría de los casos se selecciona la ciudad más cercana a la última ciudad agregada al circuito.

En el segundo paso se obtuvo un seudocódigo comentado en base a la heurística especificada. La Figura 3 muestra parte del seudocódigo correspondiente a la unión de ciudades al interior del casco convexo.

En el tercer paso se modularizó el código inicial. De esta tarea se obtuvo un total de 29 procedimientos y funciones además de un procedimiento que los reúne. Otros cambios no se efectuaron, pues el código inicial adoptaba las normas del lenguaje en que fue programado.

En el cuarto paso se unificó seudocódigo con código fuente, detectándose correspondencia entre estos. En el quinto paso se agregaron llamadas a la función generadora de datos. Después de efectuar estos pasos y ejecutar el algoritmo heurístico (Paso 6), un flujo de ejecución se generó automáticamente por el ambiente de pruebas. La Figura 4 permite apreciar el registro de los momentos en que se ejecuta una función, la cantidad de veces que fue ejecutada y el seudocódigo en detalle.

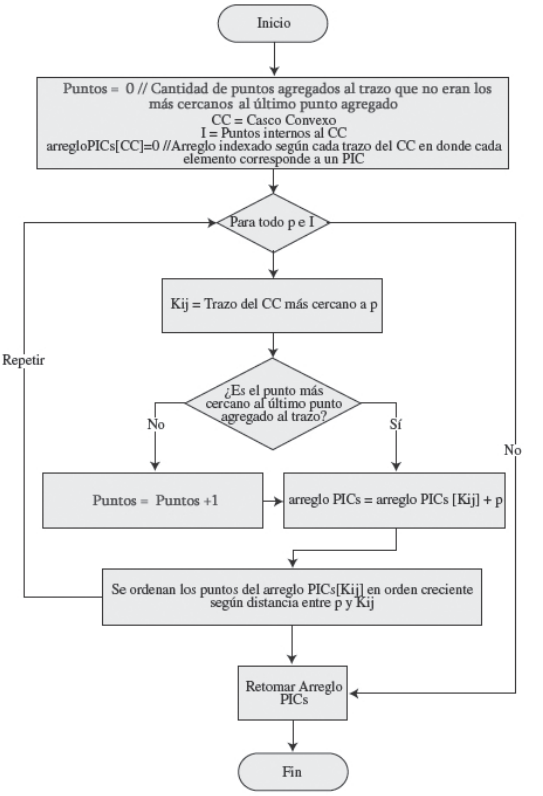

Figura 3. Ejemplo de seudocódigo con comentarios.

En el óvalo, al inicio de la Figura 4, se anota la ejecución en milisegundos y, por cada instrucción, las veces que se ejecutó.

En la etapa 7, las trazas generadas por la etapa anterior permitieron la ejecución de indicadores programados en el ambiente de pruebas. Las Figuras 5, 6 y 7 muestran la información generada por los indicadores. En cada figura se demarca en color azul el resultado para cada instancia utilizada y en color rojo el promedio obtenido para el conjunto de esas instancias. Algunas interrogantes respecto de las especificaciones seleccionadas en la primera etapa se analizan a continuación.

La primera interrogante fue: ¿En qué medida se busca enlazar ciudades internas del problema con las que pertenecen al casco convexo? La Figura 5 muestra que del total de instancias probadas, la cantidad promedio de tiempo que el algoritmo busca enlazar ciudades internas con el casco convexo es un 16%. La mayor cantidad de tiempo registrado corresponde a un 99,8%, la menor cantidad de

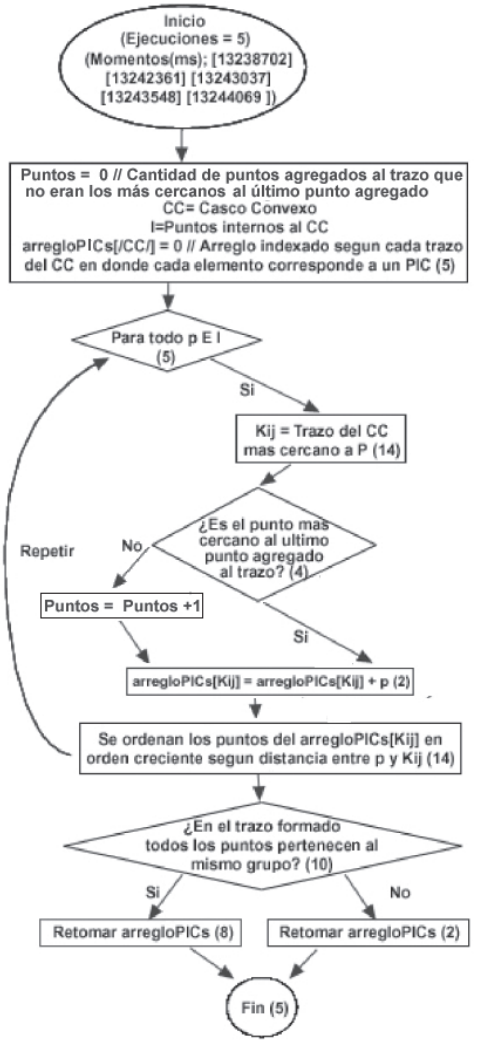

Figura 4. Flujo de ejecución de una función.

tiempo registrado corresponde a un 0,05% del total de tiempo de ejecución.

La diferencia en los porcentajes se debe a que el algoritmo busca enlazar trazos (ciudades ya enlazadas) internos al casco convexo y después unirlos a este, más que unir ciudades puntuales al casco convexo. Esta observación se encuentra bien implementada respecto de su especificación. La ecuación (1) representa el indicador utilizado, donde PI se refiere a las ciudades internas al casco convexo (CC).

$$X = \frac{\sum \text{Tiempo que se busca enlazar PI con CC} * 100}{\text{Tiempo de ejecución de la instancia}} \quad (1)$$
$$\text{Instancias probadas}$$

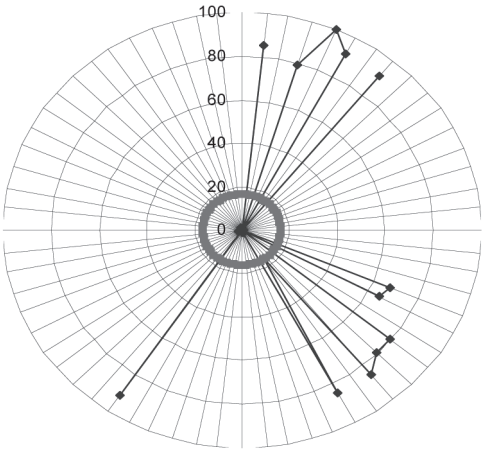

Figura 5. Indicador enlace ciudades internas con casco convexo. El valor de los ejes representa el porcentaje de tiempo respecto del tiempo total.

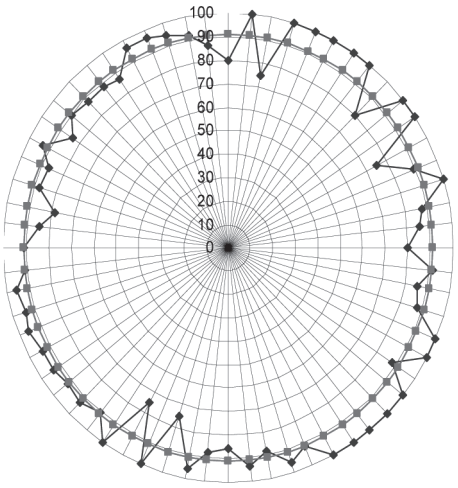

Figura 6. Indicador unión de ciudades cercanas. El valor de los ejes corresponde al porcentaje en que se enlazaron ciudades cercanas.

La segunda interrogante fue: ¿En qué medida se unen ciudades cercanas unas de otras? La Figura 6 muestra que en un 92% promedio las ciudades cercanas para cada instancia se enlazaron. En algunas instancias se logró un 100% y en otra un 75% siendo el porcentaje menor.

En general el algoritmo enlaza ciudades cercanas en cada instancia, por lo que esta observación es consistente con la especificación. La ecuación (2) representa el indicador utilizado.

$$X=\frac{\frac{\sum \text{Grupos enlazados entre sí} * 100}{\text{Cantidad de grupos}}}{\text{Instancias probadas}} \quad (2)$$

La tercera interrogante fue: ¿Existirán grupos de ciudades cercanas que ya fueron enlazadas, pero la ruta que las une es factible de mejorar? La Figura 7 muestra que del total de instancias, 28 no fueron analizadas y sobre aquellas que se aplicó un proceso de mejora, la cantidad de mejoras para cada una se aprecia insuficiente.

En la búsqueda de respuesta a esta situación se concluyó que cada vez que el algoritmo decide mejorar la ruta de un grupo enlazado, parte de cero, o sea, no guarda las mejores rutas conocidas que se pueden obtener de las ejecuciones del algoritmo.

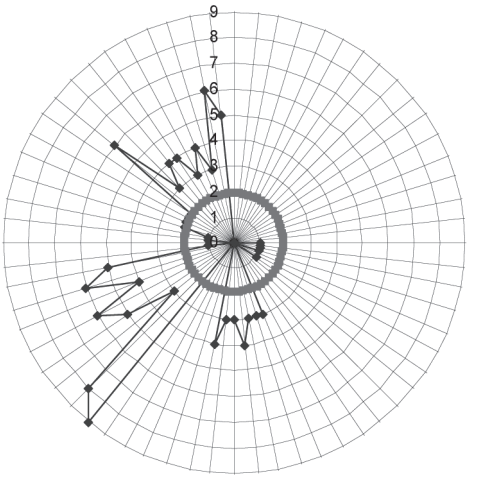

Figura 7. Indicador mejora de agrupaciones de ciudades. El valor de los ejes representa la cantidad de reagrupaciones realizadas en busca de mejoras para cada instancia.

Se estimó necesario modificar el algoritmo inicial para que guarde trazos mejorados a considerar en futuras ejecuciones. La ecuación (3) representa el indicador utilizado.

$$X=\frac{\sum \text{Reagrupaciones realizadas}}{\text{Instancias probadas}} \quad (3)$$

En la etapa 8 se procedió a modificar la versión original del algoritmo. Específicamente se implementó un procedimiento para guardar la mejor ruta conocida por cada instancia y agrupaciones iniciales, de tal forma que en siguientes ejecuciones, la mejora sea a partir de esa ruta, que probablemente el algoritmo sea capaz de mejorar.

La Tabla 1 presenta una comparativa entre los resultados (soluciones) logrados por la primera versión del algoritmo heurístico y la segunda versión del mismo, producto de la estrategia desarrollada en este trabajo. Se logró mejorar la calidad promedio de las soluciones, esto es, un 3,52% por parte de la segunda versión del algoritmo y un 6,9% por parte de la primera versión, ambas cifras respecto de óptimos conocidos, para 64 instancias.

En la Tabla 1 la columna "Instancias" se refiere al nombre con que es conocida la instancia, la columna "Óptimo" indica el óptimo conocido, la columna "V2" registra las soluciones logradas por la versión 2 del algoritmo heurístico y la columna "V1" las soluciones logradas por la versión 1 del algoritmo heurístico.

Tabla 1. Comparativa parcial de resultados.

Instancias	**Óptimo**	**V2**	**V1**
10puntos1	2211	2211	2211
10puntos1interiores	532	530	531
10puntos2	2477	2477	2477
10puntos2interiores	567	566	566
10puntos3	3353	3352	3352
10puntos3interiores	562	559	571
10puntos4	2620	2619	2620
10puntos4interiores	595	605	633
10puntos5	3285	3286	3286
10puntos5interiores	559	563	592
10puntos6interiores	528	528	529
15puntos1	3560	3561	3561
15puntos2	3531	3530	3531
15puntos3	2974	3011	3011
15puntos4	3882	3892	3947
15puntos5	2485	2486	2496
20puntos10interiores	698	740	800
20puntos12interiores	687	721	759
20puntos14interiores	664	678	703

Instancias	Óptimo	V2	V1
20puntos16interiores	590	591	614
20puntos4interiores	704	705	705
20puntos6interiores	703	758	816
20puntos8interiores	726	751	800
5puntos1	2036	2036	2036
5puntos2	1145	1145	1145
5puntos3	2203	2202	2202
5puntos4	2533	2533	2534
5puntos5	1627	1625	1625
berlin52	7542	7849	8248
bier127	118282	118607	125691
dantzig42	675	679	689
eil101	629	674	689
eil51	426	430	489
eil76	538	570	604
lin105	14379	15225	16045
pr107	44303	45330	45913
pr136	96772	104589	112209
pr76	108159	122986	128432
rat99	1211	1283	1318
st70	675	691	720

CONCLUSIONES

Se ha presentado una estrategia para la mejora de algoritmos heurísticos, específicamente algoritmos para el E-TSP. La construcción de indicadores que proveen retroalimentación para intervenir un algoritmo inicial es útil y factible, sobre todo si con ello se logran mejores soluciones.

Para concretar esta estrategia, fue necesario fusionar especificaciones de la heurística, seudocódigo y código fuente, además de adicionar en el código fuente llamados a una función generadora de datos, imprescindible para concretar la formulación de los indicadores.

La estrategia fue aplicada a un algoritmo heurístico, obteniéndose una versión mejorada del mismo, que logró mejores soluciones en promedio que su primera versión.

AGRADECIMIENTOS

Agradecemos el aporte financiero otorgado por el Departamento de Investigaciones Científicas y Tecnológicas (DICYT) y el Departamento de Ingeniería Informática de la Universidad de Santiago de Chile, USACH, para la materialización de este trabajo.

REFERENCIAS

[1] M. Garey, M. Johnson and D. Johnson. "Computers and intractability: a guide to the theory of NP-completeness". New York, USA. W.H. Freeman and Company, p. 338. 1979. ISBN: 9780716710455.

[2] A. Peña. "Juegos, Expertos e Intratabilidad Computacional". Tesis para optar al grado de Magíster en Informática. Departamento de Ingeniería Informática. Universidad Santiago de Chile, p. 164. Santiago, Chile. 2007.

[3] J. Cockbaine e I. Casas. "Un metamodelo OLAP para la evaluación de ambientes TEL". Revista de la Facultad de Ingeniería-Universidad de Tarapacá. Vol. 13 N° 1, pp. 9-20. Abril 2005. ISSN versión impresa: 0717-1072. ISSN versión en línea: 0718-1337. URL: http://www.scielo.cl/scielo.php?pid=S0718-13372005000100002&script=sci_arttext

[4] J. MacGregor and T. Ormerod. "Human performance on the traveling salesman problem". Perception & Psychophysics. Vol. 58, Issue 4, pp. 527-539. May, 1996. ISSN: 0031-5117.

[5] I. Rooij, A. Schactman, H. Kadlec and U. Stage. "Perceptual or analytical processing? Evidence from childrens and adults performance on the Euclidean Traveling Salesperson Problem". The Journal of Problem Solving. Vol. 1, Issue 1, pp. 44-73. 2006. ISSN: 1932-6246.

[6] D. Fortnow. "The status of the P versus NP problems". Communication of the ACM. Vol. 52, Issue 9, pp. 78-86. September, 2009. ISSN: 0001-0782. DOI: 10.10114/1562164.1562186.

[7] S. Cook. "The complexity of theorem proving procedures". Third Annual ACM Symposium of the Theory of Computing, pp. 151-158. 1971. DOI:101145/800157.805047

[8] Universität Heidelberg, Institut für Informatik. "TSPLIB". Fecha de actualización: Agosto

2008. Fecha de consulta: 13 de Junio de 2011. URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[9] Georgia Institute of Technology. "Concorde TSP Solver". Fecha de actualización: Diciembre 2011. Fecha de consulta: 9 de Junio de 2011. URL: http://www.tsp.gatech.edu/concorde.html

[10] DNA Evolutions. "JOpt.SDK". Fecha de actualización: 2012. Fecha de consulta: 3 de Mayo de 2011. URL: http://www.dna-evolutions.com/joptsdk.html

[11] The MathWorks, Inc. "MATLAB". Fecha de actualización: 2012. Fecha de consulta: 3 de Mayo de 2011. URL: http://www.mathworks.com/matlabcentral/fileexchange/4821-cross-entropy-tsp-solver

[12] S Skiena. "TSPSolvers". Fecha de actualización: Julio 2008. Fecha de consulta: 9 de Junio de 2011. URL: http://www.cs.sunysb.edu/%7 Ealgorith/implement/tsp/implement.shtml

[13] Q. Zhu and S. Chen. "A new ant evolution algorithm to resolve TSP problem". Presented at the International Conference on Machine Learning and Applications, pp. 62-66. 2007.

[14] E. Aarts, J. Korst and P. van Laarhoven. "A quantitative analysis of the simulated annealing algorithm: A case study for the traveling salesman problem". Journal of Statistical Physics. Vol. 50, Issue 1, pp. 187-206. 1988.

[15] G. Dantzig, R. Fulkerson and Johnson. "Solution of a large-scale traveling-salesman problem". Journal of the Operations Research Society of America. Vol. 2, Issue 4, pp. 393-410. 1954.

[16] E. Goldbarg, M. Goldbarg and J. Farias. "GRASP with path-relinking for the TSP". Metaheuristics: progress in complex systems optimization. 2007.

[17] J. Knox. "Tabu search performance on the symmetric traveling salesman problem". Computers & Operations Research. Vol. 21, Issue 8, pp. 867-876. 1994.

[18] M. Bader-El-Den, R. Poli and S. Fatima. "Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework". Memetic Computing. Vol. 1, Issue 3, pp. 205-219. 2009. DOI: 10.1007/s12293-009-0022-y.

[19] E. Burke, B. Mccollum, A. Meisels, S. Petrovic and R. Qu. "A graph-based hyper-heuristic for educational timetabling problems". European Journal of Operational Research. Vol. 176, Issue 1, pp. 177-192. 2007. DOI: 10.1016/j.ejor.2005.08.012.

[20] J. Koza. "Human-competitive results produced by genetic programming". Genetic Programming and Evolvable Machines. Vol. 11, Issue 3-4, pp. 251-284. 2010.

[21] S. Lin. "Computer solutions of the traveling-salesman problem". Bell System Tech. J. Vol. 44, pp. 2245-2269. 1965.

[22] F. Neumann. "Expected runtimes of evolutionary algorithms for the Eulerian cycle problem". Computers & Operations Research. Vol. 35, Issue 9, pp. 2750-2759. 2008.

[23] J. Potvin. "The traveling salesman problem: a neural network perspective". Centre de recherche sur les transports. Université de Montréal. Canadá. 1992.

[24] R. Rasmussen. "TSP in spreadsheets–A fast and flexible tool". Omega. Vol. 39, Issue 1, pp. 51-63. 2011.

[25] D. Wolpert and W. Macready. "No free lunch theorems for optimization". IEEE Transactions on Evolutionary Computation. Vol. 1, Issue 1, pp. 67-82. 1997.

[26] D. Applegate. "The traveling salesman problem: a computational study". Princeton Univ. Pr., New Jersey, USA. 2006.

[27] S. Lin and B. Kernighan. "An effective heuristic algorithm for the traveling-salesman problem". Operations research. Vol. 21, Issue 2, pp. 498-516. 1973.