UNA PLATAFORMA PARA LA GENERACIÓN DE ÁRBOLES DE INSTRUCCIONES PARA PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA

Carlos Contreras Bolton
Universidad de Santiago de Chile
carlos.contrerasb@usach.cl

Gustavo Gatica
Universidad Andrés Bello
ggatica@unab.cl

Victor Parada
victor.parada@usach.cl

RESUMEN

Algunos de los mejores resultados computacionales obtenidos para determinados problemas de optimización combinatoria, son consecuencia de la hibridación de diversas técnicas heurísticas y metaheurísticas. Aunque el diseño de este tipo de algoritmos es una tarea ardua y muchas veces compleja para el ser humano, poca atención ha recibido el estudio de realizarla con la ayuda de un computador, especificando, mediante un árbol de instrucciones, el mejor orden en el que deben ser ejecutadas las heurísticas elementales. En este artículo, presentamos una plataforma computacional basada en computación evolutiva, para generar árboles de instrucciones para un problema de optimización combinatoria de manera automática. Ilustramos su uso con el problema de coloración de grafos, presentando los resultados numéricos de un árbol de instrucciones generado automáticamente. El procedimiento presenta un 8,3% de error relativo promedio, mientras que la heurística con la cual nos comparamos presenta un 82,5%, considerando instancias típicas del benchmark DIMACS.

Palabras claves: Coloración de Vértices, Generación Automática de Algoritmos, Computación Evolutiva, Heurísticas.

Área Principal: Metaheurísticas.

ABSTRACT

Some of the best computational results obtained so far for certain combinatorial optimization problems result from the hybridization of various well known techniques such as heuristics and metaheuristics. Although the design of such algorithms is a difficult task and often complex for humans, little attention has been given to its study with the help of a computer allowing the specification of a tree of instructions, i.e. the best order in which must be executed a set of elementary heuristics. We present a computational platform based on evolutionary computation to generate trees of instructions composed by elementary heuristics in order to automatically solve a combinatorial optimization problem. We illustrate its use with the graph coloring problem, presenting the numerical results of a selected tree of instructions. The procedure has a 8.3% average relative error, while the heuristic with which we compare it presents 82.5%, considering a set of typical benchmark instances.

Palabras claves: Vertex Coloring, Automatic Generation of Algorithms, Evolutionary Computation, Heuristic.

Área Principal: Metaheuristics.

1. Introducción.

En un problema de optimización combinatoria se debe encontrar la mejor solución con base en una función de optimización. En particular, los problemas de la familia NP-Hard (Garey & Johnson, 1979; Papadimitriou & Steiglitz, 1998), han sido estudiados y abordados con diversas técnicas, tales como programación matemática, programación dinámica, optimización no lineal, métodos heurísticos, y metaheurísticos. Sin embargo, los algoritmos que producen mejor desempeño computacional son hibridaciones de tales técnicas (Pissinger, 2005; Titiloye & Crispin, 2011a, 2011b; Rego, Gamboa, Glover F & Osterman, 2011). Aunque el diseño de este tipo de algoritmos es una tarea ardua y muchas veces compleja para el ser humano, poca atención ha recibido el estudio de realizarla con la ayuda de un computador (Barra, Gatica, & Parada, 2009; Sepúlveda, 2011; Zepeda, Gatica, & Parada, 2011). La generación de al menos, un orden en el que las heurísticas deben ser combinadas para dar origen a los algoritmos para problemas de optimización, sería de gran ayuda para diseñar experimentos numéricos que conduzcan al mejor algoritmo. En este artículo, proponemos una plataforma computacional basada en computación evolutiva, que permite crear árboles de instrucciones que especifican el orden en el cual se deben ejecutar un conjunto de instrucciones elementales. La plataforma recibe, los datos del problema de optimización en estudio, un conjunto de componentes heurísticos elementales, la estructura de datos definida para el problema y una apropiada función de evaluación. Como salida, produce un árbol de instrucciones, comprensible por el humano y competitivo con los algoritmos eficientes que han sido desarrollados hasta el momento para el problema. Nosotros presentamos los resultados numéricos de un árbol de instrucciones generado de esta manera, para el bien conocido Problema de Coloreado de Grafos (Matula, Marble, & Isaacson, 1972), que consiste en encontrar el número cromático $\chi(G)$, correspondiente al mínimo número de colores necesarios para colorear un grafo no dirigido G.

En la siguiente sección de este artículo, se describe la plataforma y el procedimiento utilizado para la obtención del árbol de instrucciones para el PCG. En la última sección, son presentadas las conclusiones del estudio.

2. Diseño de la plataforma.

En esta sección se presenta la plataforma diseñada, la representación utilizada para el PCG, las estructuras de datos, la definición de la función de evaluación, la definición general de funciones y terminales utilizadas. Además, se presenta el hardware y software utilizado.

La plataforma

La arquitectura de la plataforma, se puede apreciar gráficamente en la Figura 1. En ésta, el módulo evolutivo (AG), interactúa con un módulo correspondiente a una función constructora (FC), capaz de decodificar un *string* binario y construir un correspondiente árbol de instrucciones, que a su vez, permite resolver el problema de optimización. El submódulo FCe, contiene el conjunto de heurísticas y estructuras de control definidas para el problema, mientras que el módulo FCc, contiene un conjunto de reglas gramaticales definidas para permitir la evaluación del conjunto de instancias (Contreras Bolton, Gatica, & Parada, 2011). La plataforma considera la utilización de biblioteca, tales como STL (Musser, Saini, & Stepanov, 1996) y *The Boost Graph Library* (Siek et al., 2002), que contienen operaciones para administrar eficientemente la estructura de datos definida para la ejecución del experimento. La función de evaluación, se evalúa mediante un conjunto de procesadores con memoria compartida, utilizándo *OpenMP* (Chapman et al., 2007). Adicionalmente, con el objetivo de visualizar los archivos de salida del proceso evolutivo, se emplean dos bibliotecas *Graphviz* (Gansner, 2009) y *Gnuplot* (Janert, 2009).

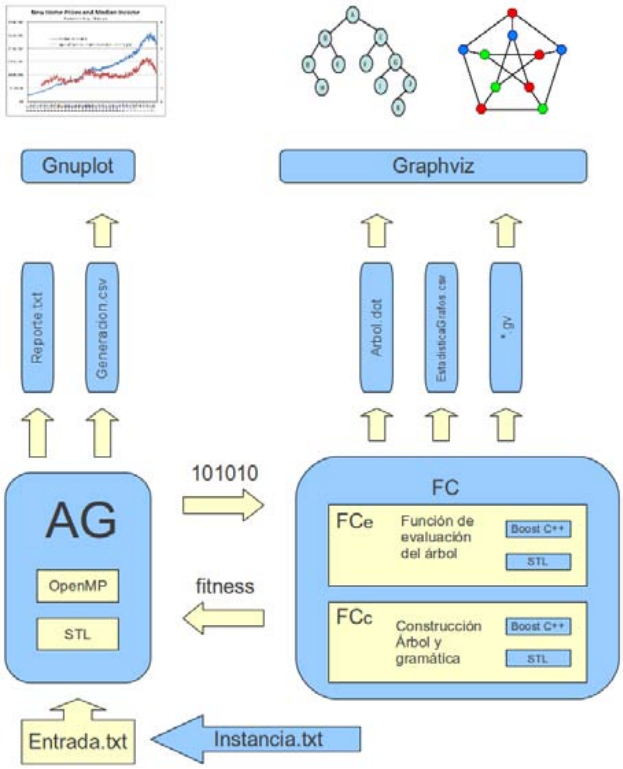

Figure 1: Arquitectura del Diseño

El proceso evolutivo

El proceso evolutivo lo realiza un AG poblacional, que opera en una modalidad genotipo-fenotipo (Falkenauer, 1998). En cada etapa, aplica operaciones de selección mediante torneo binario, cruzamiento de un punto y mutación (Affenzeller, Winkler, Wagner, & Beham, 2009). El genotipo es codificado mediante *strings* binarios, los cuales a su vez son decodificados en un árbol binario, como se puede apreciar en la Figura 2, la cual presenta un *string* de 20 bits, donde cada gen tiene un tamaño de 5 bits, para representar 32 componentes elementales. Cada árbol de instrucciones resuelve un conjunto de instancias del problema de optimización, y se asigna un valor de aptitud o *fitness* que depende del desempeño para resolver el conjunto. El valor del *fitness* es devuelto al AG, siguiendo su proceso evolutivo.

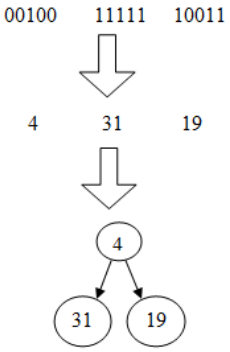

Figura 2. Decodificación de un *string* binario.

Representación de PCG para la Plataforma

Siguiendo las ideas provenientes de la Programación Genética (Koza, 2003), se considera un conjunto de funciones y terminales como los componentes elementales. Se utiliza una lista de adyacencia como estructura de datos para dar soporte a los árboles de instrucciones generados

artificialmente, y para su implementación se utiliza *Boost Graph Library*, que contiene un conjunto de funciones optimizadas para las operaciones recurrentes sobre grafos.

Como componentes elementales para el árbol de instrucciones a producir, se define un conjunto de 13 funciones y un conjunto de 13 terminales (Contreras Bolton et al., 2011; Sepúlveda, 2011). Se considera que la ejecución de una función requiere de uno o dos argumentos que también pueden ser funciones o terminales y se denotan como *función(argumento1,argumento2)*. Cada vértice del grafo está enumerado, al igual que los colores disponibles para colorear. Las funciones se subdividen en funciones de coloración y funciones genéricas. Las primeras, asignan un color al vértice en base a un criterio definido y requieren como argumento un terminal que retorne un vértice, en cambio, las segundas, reciben como argumento un valor booleano, que corresponde a la evaluación de un subárbol. Se utilizan nombres intuitivos que describen su funcionamiento, su construcción se realiza considerando partes de las heurísticas disponibles en la literatura.

Se define una función de *fitness* donde la coloración, corresponde a una medida de error relativo de la cantidad de vértices coloreados, con respecto a la cantidad total de vértices de cada problema.

Hardware Utilizado

La plataforma está diseñada para operar en un *cluster* de computadores, consistente en dos máquinas con procesadores *Intel Xeon* E5045 de 2 *GHz*, con 16 GB de memoria *RAM*, y otras 6 máquinas con procesadores *AMD Opteron* 2210 de 1.8 Ghz, con 2 GB de memoria *RAM*. Todos utilizan la distribución *GNU/Linux Ubuntu* 10.10.

3. Resultados y Discusión.

El árbol de instrucciones obtenido mediante el proceso evolutivo es numéricamente competitivo con heurística *Largest First*. El algoritmo generado al igual que las heurísticas existentes encuentra solución a todos los problemas. En la Tabla 1, se presentan los resultados numéricos del árbol de instrucciones producido, con las instancias LEI (Leighton, 1979); y son testeados con 10 problemas CAR (Caramia & Dell'Olmo, 2004) del *benchmark* DIMACS, disponible en http://mat.gsia.cmu.edu/COLOR/instances.html. Las primeras seis columnas corresponden a: nombre del problema, número de vértices, número de aristas, densidad, número cromático óptimo y número cromático mejor conocido; las siguientes tres columnas, corresponden a los resultados que proporciona la heurística *Largest First*, el número cromático obtenido por nuestro algoritmo y el tiempo de resolución. El símbolo "?" indica que se desconoce $\chi(G)$ para la instancia correspondiente y se utiliza letra negrita cuando se obtuvo el número cromático mejor conocido.

El árbol de instrucciones que se obtiene en el proceso evolutivo se presenta en la Figura 4; sus nodos corresponden a las funciones y terminales definidas y los arcos de cada función o terminal, definen la trazabilidad del procedimiento. Según lo anterior, corresponde a un procedimiento constructivo y goloso dado que colorea el grafo gradualmente y cada vez que aplica un color, trata de no aumentar el número de colores. Específicamente, en su etapa principal el árbol de instrucciones realiza la coloración completa de un grafo, esto sucede porque hay secciones del código que se ejecutan pero no logran su cometido, es decir, colorear o cumplir con una condición.

El procedimiento se puede explicar en cuatro etapas, enmarcadas en la Figura 4. La primera, consiste en colorear el vértice que tiene el mayor número de vértices adyacentes sin colorear, con el color más frecuente, luego, a los vértices que posean la mayor cantidad de vértices adyacentes los colorea con tres criterios de coloración, el color menor numerado, el menos frecuente y el más frecuente respectivamente, luego, realiza un intercambio de colores entre dos vértices, el primero es el vértice con el mayor número de vértices adyacentes sin colorear y el segundo, el vértice con el color mayor numerado. En la segunda etapa, se considera que mientras exista un vértice con el mayor número de vértices adyacentes sin colorear, se colorea con el color más frecuente a los

vértices que tengan la mayor cantidad de vértices adyacentes con distinto color asignado. En la tercera etapa, se considera que mientras exista un vértice que posee el mayor número de vértices adyacentes sin colorear, se colorea con el color menos frecuente a los vértices con menor numeración que tengan la menor cantidad de vértices adyacentes. La cuarta etapa es un refinamiento, ya que si existe un vértice con el color de mayor numeración entonces, éste se colorea con el color más frecuente.

Tabla 1. Resultados Numéricos.

Instancia	***N***	***m***	***d(G)***	**χ(G)**	***Best*(X)**	***Largest First***	**Árbol de Instrucciones**	**Tiempo (s)**
1-FullIns_3	30	100	0,230	?	4	8	**4**	0,001
1-FullIns_4	93	593	0,139	5	5	11	**5**	0,013
1-FullIns_5	282	3247	0,082	6	6	14	**6**	17,804
2-FullIns_3	52	201	0,152	5	5	10	**5**	0,004
2-FullIns_4	212	1621	0,072	6	6	14	7	0,072
2-FullIns_5	852	12201	0,034	7	7	18	**7**	1,930
3-FullIns_3	80	346	0,109	6	6	12	**6**	0,007
3-FullIns_4	405	3524	0,043	7	7	17	**7**	0,227
4-FullIns_3	114	541	0,084	7	7	14	**7**	0,014
4-FullIns_4	690	6650	0,028	8	8	20	**8**	0,780
5-FullIns_3	154	792	0,067	8	8	16	**8**	0,025
5-FullIns_4	1085	11395	0,019	?	9	23	**9**	1,683

4. Conclusiones.

En este artículo se presenta una plataforma computacional que utiliza un algoritmo evolutivo para crear árboles de instrucciones para problemas de optimización combinatoria, combinando componentes de las heurísticas elementales. La plataforma se basa en un algoritmo genético que opera bajo la modalidad genotipo-fenotipo. El genotipo identifica con un string binario la participación de los componentes elementales en la posterior construcción del árbol. En tanto, el fenotipo se representa mediante un árbol de instrucciones que indica el orden de ejecución de los componentes elementales. Con el fin de verificar el funcionamiento de la plataforma, nosotros generamos un árbol de instrucciones para el bien conocido problema de coloreamiento de grafos. Para evolucionar los árboles se utilizaron los 12 grupos de instancias presentes en el *benchmark* DIMACS y para testearlos se utiliza un conjunto de 12 instancias del grupo CAR del mismo *benchmark*. Se presenta un árbol de instrucciones que corresponde a un procedimiento constructivo y goloso, que es competitivo con la bien conocida heurística Largest First para el problema.

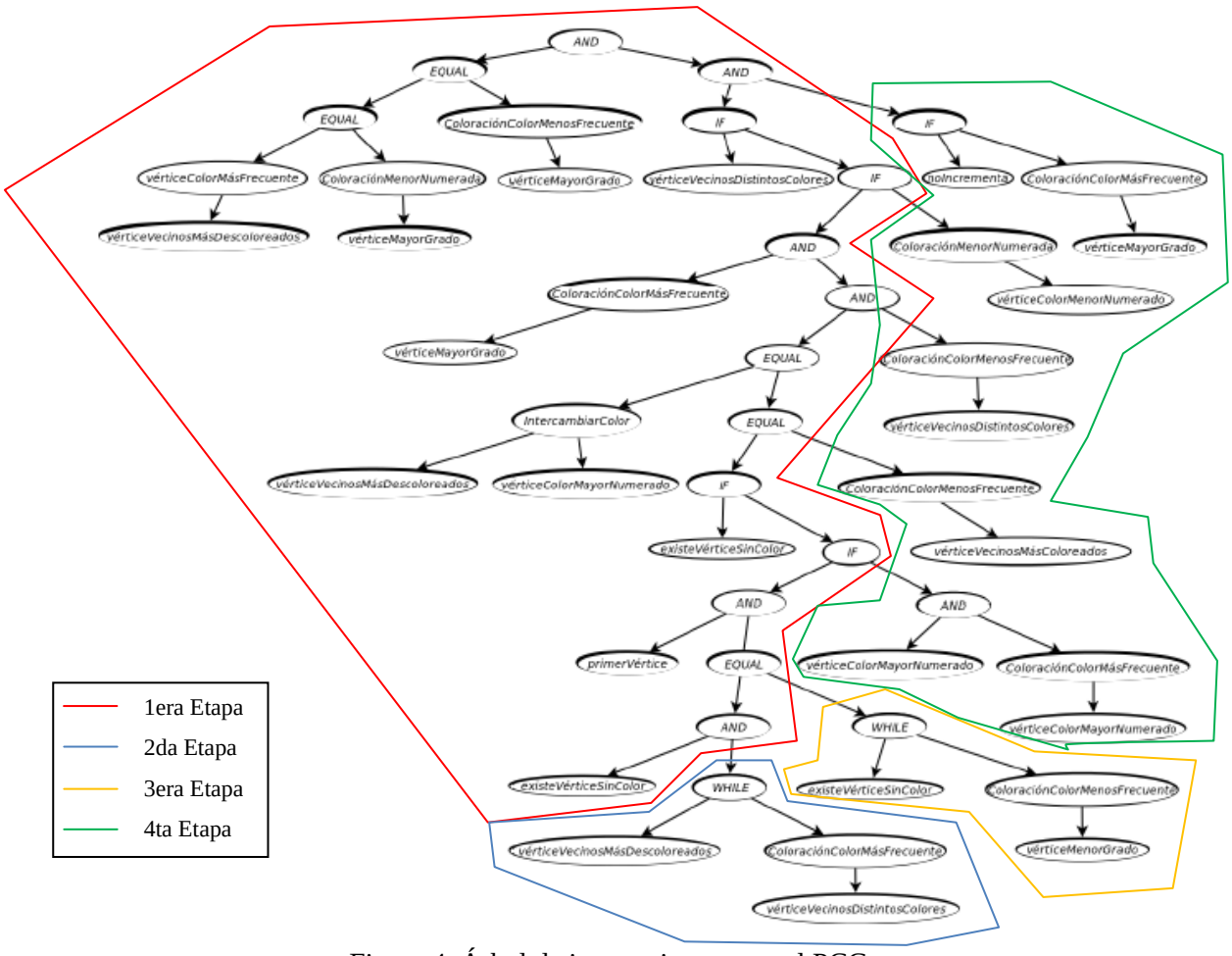

Figura 4. Árbol de instrucciones para el PCG.

Referencias.

Affenzeller, M., Winkler, S., Wagner, S., & Beham, A. (2009). *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications* (1.ª ed.). Chapman and Hall/CRC.

Barra, L., Gatica, G. & Parada, V. (2009) Evolucionando programas para el problema de la mochila mediante programación genética. En VIII Congreso Chileno de Investigación Operativa - OPTIMA 2009, Chillán, Chile.

Caramia, M., & Dell'Olmo, P. (2004). Bounding vertex coloring by truncated multistage branch and bound. *Networks*, *44*(4), 231–242.

Chapman, B., Jost, G. y Pas, R. (2007). *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press.

Contreras Bolton, C., Gatica, G., & Parada, V. (2011). Generación automática de algoritmos para el problema de coloración de vértices. En IX Congreso Chileno de Investigación Operativa, OPTIMA 2011, Pucón, Chile.

Falkenauer, E. (1998). *Genetic Algorithms and Grouping Problems*. New York, NY, USA: John Wiley & Sons, Inc.

Gansner, E. R. (2009). Drawing graphs with GraphViz. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, USA

Garey, M., & Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. New York: W. H. Freeman & Co.

Janert, P. (2009). Gnuplot in Action: Understanding Data with Graphs. Manning Publications Co., Greenwich, CT, USA.

Koza, J. R. (2003). *Genetic Programming IV: Routine Human-competitive Machine Intelligence*. Kluwer Academic Pub.

Leighton, F. T. (1979). A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, *84*(6), 489–503.

Matula, D. M., Marble, G., & Isaacson, J. D. (1972). Graph coloring algorithms. *Graph Theory and Computing*, Academic Press (R. C. Read., pp. 109–122). New York: Academic Press.

Musser, D. R., Saini, A., & Stepanov, A. (1996). *STL Tutorial and Reference Guide: C++ Programming With the Standard Template Library*. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, MA.

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Mineola, N.Y.: Dover Publications.

Pisinger, D. (2005). Where are the hard knapsack problems?. *Computers & Operations Research*, 32(9), 2271–2284.

Sepúlveda, M. (2011). Generación Automática de Algortimos para Problemas de Optimización Combinatoria. Tesis Doctorado en Ciencias de la Ingeniería. Universidad de Santiago de Chile, Chile.

Siek, J., Lee, L. Q., Lumsdaine, A., Lee, L. Q., Blackford, L. S., Demmel, J., Dongarra, J., et al. (2002). *The Boost Graph Library: User Guide and Reference Manual*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Rego C, Gamboa D, Glover F, Osterman C. (2011). Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3), 427-441

Titiloye, O., & Crispin, A. (2011a). Graph Coloring with a Distributed Hybrid Quantum Annealing Algorithm. En J. O'Shea, N. T. Nguyen, K. Crockett, R. J. Howlett, & L. C. Jain (Eds.), *Agent and Multi-Agent Systems: Technologies and Applications* (Vol. 6682, pp. 553–562). Berlin, Heidelberg: Springer Berlin Heidelberg.

Titiloye, O., & Crispin, A. (2011b). Quantum annealing of the graph coloring problem. *Discrete Optimization*, *8*(2), 376–384.

Zepeda, J. A., Parada, V., Gatica, G., & Sepúlveda, M. (2011). Automatic Generation of Algorithms for the Non Guillotine Cutting Problem. En Proc. of the VII ALIO/EURO – Workshop on Applied Combinatorial Optimization, Porto, Portugal (pp. 1-5).