A Spanning Tree-Based Encoding of the MAX CUT Problem for Evolutionary Search

Kisung Seo[1], Soohwan Hyun[1], and Yong-Hyuk Kim[2,*]

[1] Dept. of Electronic Engineering, Seokyeong University, Seoul, Korea
ksseo@skuniv.ac.kr, xjavalov@shhyun.com
[2] Dept. of Computer Science and Engineering, Kwangwoon University, Seoul, Korea
yhdfly@kw.ac.kr

Abstract. Most of previous genetic algorithms for solving graph problems have used vertex-based encoding. In this paper, we introduce spanning tree-based encoding instead of vertex-based encoding for the well-known MAX CUT problem. We propose a new genetic algorithm based on this new type of encoding. We conducted experiments on benchmark graphs and could obtain performance improvement on sparse graphs, which appear in real-world applications such as social networks and systems biology, when the proposed methods are compared with ones using vertex-based encoding.

Keywords: Basis change, encoding, representation, genetic algorithm, MAX CUT, spanning tree, graph.

1 Introduction

In genetic algorithms (GAs), different encodings lead completely different search on solution space, and as a result, encoding can affect performance largely. There have been many studies of emphasizing the importance of encoding in GAs. Kim *et al.* [1] improved the performance of genetic algorithms on various problems by rearranging the related gene positions to be closely located. This gene rearrangement can be seen as a simply type of transformed encoding. There have also been more generalized studies of encoding transformation making the relation between genes be the most independent by applying invertible linear transformation [2, 3]. These studies just showed the importance of encoding transformation, but they failed to show the concrete transformation methods. As an extension of these studies, there has been a trial to find better encoding using a meta-GA [4]. However it also failed to give a good guideline about how we transform encoding in a given problem.

Most studies about graph problems such as graph partitioning and MAX CUT have been vertex-centric when dealing with partitions and representing them [5-14]. Intuitive techniques based on vertices, which are easy to manage, have been mainly used to solve the graph problems. However, when dealing with partitions, there have been

* Corresponding author.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 510–518, 2012.

studies [15-19] using methods based on edge, which is a dual of vertex. In particular, Armbruster *et al.* [20] and Yoon *et al.* [21] used an edge representation for solving graph partitioning, which maps a solution to an edge set, not a vertex set. In their representation, each location of an encoding is assigned to 1 if its corresponding edge is on the cut and 0 otherwise. This representation is well adapted to their integer programming formulation, but it is very crucial but difficult to check whether or not a given encoding forms a valid graph partition.

In this paper, we propose a new genetic algorithm based on not vertex-based encoding but spanning tree-based encoding [22, 23] as a kind of edge-based encoding. Contrary to general edge-based encoding, spanning tree-based encoding represents only feasible partitions. As a target problem, we adopted the MAX CUT problem, which is well known as a representative NP-hard problem, and examined the performance of the proposed genetic algorithms experimentally. The proposed method is expected to well perform on sparse graphs. In particular, if we consider that graphs appearing in real-world applications such as social networks and systems biology are sparse, the proposed method has great potential in real-world graph problems.

Section 2 discusses the MAX CUT problem and its previous work. Section 3 describes the proposed spanning tree-based encoding scheme. Section 4 presents experimental results on various graph sets, and Section 5 concludes the paper.

2 MAX CUT

It is important in combinatorial optimization to partition the vertices into two disjoint subsets of nearly equal size such that the sum of edge weights with two edge endpoints in different sets (cut size) is maximized or minimized. Given an undirected graph $G = (V, E)$ with edge weights, the MAX CUT problem (Fig. 1) is that of finding a subset $S \subset V$ which maximizes the sum of edge weights in the cut $(S, V - S)$.

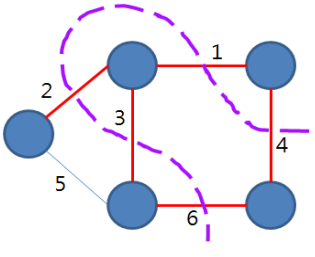

Fig. 1. Example of MAX CUT

Every graph has a finite number of cuts, so one can find the minimum or maximum weight cut in a graph by an exhaustive search that enumerates the sizes of all the cuts. This is not a practical approach for large graphs which arises in real-world applications since the number of cuts in a graph grows exponentially with the number of vertices. Although we can solve the min-cut problem without balance requirement in polynomial time using the maxflow-mincut algorithm [24], we have no such fortune

when it comes to the MAX CUT problem. There is no known way to solve the problem optimally other than by exhaustive enumeration. The MAX CUT problem is one of Karp's original NP-complete problems [25] and has been known to be NP-complete even if the problem is unweighted [26].

Since there is no algorithm that guarantees an optimal solution, a typical approach to solve such a problem is to find a ρ-approximation algorithm that delivers a solution at least ρ times the optimal value in polynomial time. Sahni and Gonzales [27] presented a 1/2-approximation algorithm for the MAX CUT problem. Their greedy approach iterates through the vertices and decides which placement (S or V - S) maximizes the cut of vertex v_i with respect to vertices v_1 to v_{i-1}. Since [27], many researchers have presented approximation algorithms for the MAX CUT problem [28-31], but little progress has been made. For more than twenty years a factor of 0.5 has been the best-known polynomial-time performance guarantee for the MAX CUT problem. An algorithm by Goemans and Williamson (GW) [32] guarantees a factor of 0.878 of the optimum. The significant improvements are due to the technique of positive semidefinite programming and randomized rounding. However, solving semidefinite programming is computationally expensive. Homer and Peinado [33] gave a parallelized version of GW. In [33], GW was improved by combining with simulated annealing (SA) [34]. Afterward, Kim et al. [35] successfully applied GAs to the MAX CUT problem. In practical, when the GA is combined with lock-gain-based local search [11], the hybrid GA could outperform GW (the best approximation algorithm) combined with SA. It is known the GAs have good performance when applied to the MAX CUT problem [35], we adopted the MAX CUT problem to test our new encoding scheme in this paper.

The MAX CUT problem has many applications in various fields, It has been observed that one of the phases (the layer assignment problem) in the design process for VLSI chips and printed circuit boards (PCB) can be reduced to the MAX CUT problem [36, 37]. One of the most famous applications of the problem comes from a classical application to statistical physics [36]. It is concerned with the exact determination of a minimal energy configuration of a spin glass under no exterior field and under a continuously varying exterior magnetic field. Poljak and Tuza [38] provided a comprehensive survey of the MAX CUT problem.

3 Encoding and Evaluation

Each solution is represented by a chromosome, which is a binary string. In this section, we consider two different types of binary encoding to represent solutions for the MAX CUT problem.

3.1 Vertex-Based Encoding

When we use vertex-based encoding in GAs, the number of genes in the chromosome equals n, which is the number of vertices in the graph. Each gene corresponds to a

vertex in the graph. A gene has value 0 if the corresponding vertex is in S, and has value 1 otherwise.

To evaluate the cut size of a solution, we should compute the number of cut edges, which is an edge whose end-vertices are in different sides. For each edge, to determine if it is a cut edge, we just check that the values of genes corresponding to its end-vertices are different, i.e., (0,1) or (1,0).

3.2 Spanning Tree-Based Encoding

If $\{V_1, V_2\}$ is a partition of V, the set $E(V_1, V_2)$ of all the edges of G crossing between V_1 and V_2 is called a *cut*. Cut space consisting of all the cuts is proven to be vector space [23]. It means that an arbitrary cut can be represented by a linear combination of basis elements of cut space.

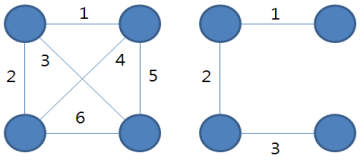

Fig. 2. Example of a graph (left) and its spanning tree (right)

In the case that the graph G is connected[1], we can derive a basis of cut space from a spanning tree of G. Finding basis of cut space based on spanning trees are known as nontrivial ones. General graph traversal algorithms such as depth-first search (DFS) and breadth-first search (BFS) can produce spanning trees of G. Let T be a spanning tree of G. For each edge e of the n - 1 edges in T, the graph T - e has exactly two components, and the set C_e of edges in G between the two components forms a cut. These n - 1 cuts are linearly independent and hence form a basis of cut space.

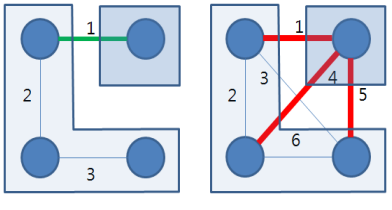

Fig. 3. Edge of spanning tree, sub-sets and cuts

When we use a spanning tree-based encoding in GAs, the number of genes in the chromosome equals n-1, the number of edges in T. Each gene corresponds to an edge in T. To evaluate the cut size of a solution, we should compute the number of cut edges. The set of cut edges is easily computed by summing C_e for each edge e in T whose gene value is 1. Then the cut size becomes the cardinality of the set of cut edges (see Fig. 3).

[1] In this paper, we assume that G is connected for convenience.

4 Simulation and Analysis

4.1 Experimental Environments

This section describes how we evaluated the proposed GA approach to develop spanning tree-based encoding for the MAX CUT problem. The GA parameters are shown in Table 1. The one-point crossover and random mutation were used for genetic recombination, and tournament selection is also adopted. The proposed algorithm was implemented using Open Beagle [39] with Boost Graph Library [40]. Three methods - Kruskal-like, DFS, and BFS - are used to find spanning trees. Every GA run is repeated 30 times for each case.

Table 1. GA Parameters

Parameters	*Values*
Max. # of generations	100
Population size	100
Crossover rate	0.9
Mutation rate	0.1
Tournament size	7

4.2 Experimental Results

The tabular results for various sets on a total of 31 graphs are provided in Table 2. The different classes of graphs that we tested our algorithms on are described below.

G*n.p*: a random graph on *n* vertices with edge probability *p*. E.g., G1000.01 is a 1,000-vertex graph with *p*=0.01.

U*n.d*: a geometric random graph on *n* vertices and expected degree *d*. E.g., U500.10 is a 500-vertex geometric graph with "expected degree" 10.

breg*n.b*: a regular random graph on *n* vertices in which each vertex has degree 3 and the optimal bisection size is *b* with high probability, i.e., probability approaches 1 as *n* approaches infinity.

cat.*n*: A caterpillar graph on *n* vertices, with each vertex having six legs. rcat.*n* is a caterpillar graph with *n* vertices, where each vertex on the spine has $\sqrt{n}$ legs.

grid*n.b*: A grid graph on *n* vertices and whose optimal minimum cut size is known to be *b*. w-grid*n.b* denotes the same grid but the boundaries are wrapped around.

Please see the reference [41] for details on how this class of graphs is generated.

Table 2. Comparison of encoding schemes for various graphs

Instances	Vertex / Edge ratio (%)	Vertex encoding		Edge encodings								
				Kruskal-like			DFS			BFS		
		Cut size	Std dev	Cut size	Std dev	Improvement(%)	Cut size	Std dev	Improvement(%)	Cut size	Std dev	Improvement(%)
G500.005	79.9%	375.0	3.7	391.3	3.0	4.35	**392.1**	**3.2**	**4.55**	388.9	3.8	3.71
G500.01	40.8%	700.1	4.5	706.1	4.6	0.87	**708.1**	**5.7**	**1.15**	707.0	4.4	1.00
G500.02	21.2%	**1303.1**	**7.2**	1293.3	7.4	-0.75	1294.7	7.9	-0.64	1295.7	7.5	-0.57
G500.04	9.7%	**2747.9**	**10.0**	2718.3	10.5	-1.08	2711.1	7.9	-1.34	2722.2	12.1	-0.94
G1000.005	40.0%	1373.4	8.4	**1383.6**	**5.4**	**0.75**	1383.5	6.9	0.74	1383.2	4.3	0.71
G1000.01	19.7%	**2711.0**	**8.5**	2699.2	7.7	-0.44	2695.9	9.7	-0.56	2702.0	7.8	-0.33
G1000.02	9.9%	**5307.4**	**10.9**	5271.3	13.6	-0.68	5265.6	15.0	-0.79	5280.5	15.2	-0.51
U500.05	38.9%	597.2	3.6	**606.3**	**3.8**	**1.52**	605.0	3.7	1.31	606.0	3.4	1.47
U500.10	20.3%	1276.1	4.8	1282.3	4.5	0.49	1279.3	4.0	0.25	**1283.8**	**6.5**	**0.61**
U500.20	11.0%	**2410.2**	**5.7**	2407.4	5.4	-0.11	2396.0	7.6	-0.59	2402.1	7.5	-0.34
U500.40	5.7%	**4575.6**	**7.1**	4563.2	8.2	-0.27	4547.1	6.6	-0.62	4557.5	7.2	-0.40
U1000.05	41.7%	1309.0	7.6	**1324.9**	**4.6**	**1.21**	1324.7	5.3	1.20	1323.1	5.0	1.08
U1000.20	10.7%	**4877.9**	**10.4**	4871.1	10.6	-0.14	4851.2	12.1	-0.55	4861.2	12.5	-0.34
U1000.40	5.5%	**9292.5**	**12.2**	9261.0	8.8	-0.34	9234.2	12.2	-0.63	9252.2	15.6	-0.43
breg500.12	66.5%	445.4	4.8	460.8	3.6	3.47	**461.8**	**3.3**	**3.67**	459.9	3.7	3.25
breg500.16	66.5%	444.5	3.4	**462.5**	**4.7**	**4.04**	462.3	3.8	4.01	460.0	3.5	3.49
breg500.20	66.5%	444.9	4.6	460.8	3.1	3.57	460.7	3.2	3.55	**461.2**	**4.3**	**3.66**
cat.352	100.0%	224.1	2.9	242.1	2.1	8.05	241.5	2.5	7.80	**242.7**	**2.8**	**8.33**
cat.702	100.0%	419.0	3.3	444.6	3.5	6.10	**446.2**	**3.9**	**6.49**	445.8	3.0	6.40
cat.1052	100.0%	606.8	5.0	**644.9**	**5.6**	**6.28**	**644.9**	**5.6**	**6.28**	**644.9**	**5.7**	**6.28**
cat.5252	100.0%	2804.5	10.0	2890.8	10.0	3.07	2885.3	10.7	2.88	**2891.2**	**10.2**	**3.09**
rcat.134	100.0%	99.4	1.5	**106.1**	**1.5**	**6.74**	**106.1**	**1.5**	**6.74**	**106.1**	**1.6**	**6.74**
rcat.554	100.0%	339.0	3.1	**360.2**	**2.5**	**6.26**	**360.2**	**2.5**	**6.26**	**360.2**	**2.6**	**6.26**
rcat.994	100.0%	577.9	5.2	**611.3**	**4.7**	**5.78**	**611.3**	**4.7**	**5.78**	**611.3**	**4.8**	**5.78**
rcat.5114	100.0%	2736.6	10.3	2817.4	10.0	2.95	**2817.6**	**9.7**	**2.96**	2816.1	9.5	2.91
grid100.10	55.2%	130.1	2.4	133.2	2.0	2.36	132.9	2.4	2.10	**133.7**	**1.8**	**2.77**
grid500.21	52.3%	556.9	4.3	569.4	4.3	2.24	**571.7**	**6.0**	**2.66**	570.2	5.5	2.38
grid1000.20	51.8%	1075.5	5.6	1094.6	7.2	1.77	**1096.3**	**5.1**	**1.93**	1095.9	7.7	1.89
w-grid100.20	49.5%	145.1	2.6	**146.8**	**1.9**	**1.22**	145.2	2.7	0.07	144.3	2.4	-0.55
w-grid500.42	49.9%	582.1	4.6	594.0	5.1	2.04	**594.5**	**5.4**	**2.12**	591.3	3.9	1.58
w-grid1000.40	50.0%	1113.8	9.0	**1133.2**	**7.0**	**1.74**	1130.2	5.1	1.47	1129.5	5.8	1.40

The ratio of vertices to edges in sparse graphs is greater than in dense graphs. For connected graphs, if n is the number of vertices, then the number of edges is between n-1 and $n(n-1)/2$. Therefore, the ratio of vertices to edges has a value between $2/(n-1)$ to $n/(n-1)$.

For the case of sparse graphs in which the ratio of vertices to edges is more than 38.9%, the performance of spanning tree-based encoding is superior to the results of vertex-based encoding. The improvement increases for greater ratios of vertices to edges, in general. Examples include breg*n*.*b*, grid*n*.*b*, cat.*n*, rcat.*n*, and w-grid*n*.*b*. The cat.*n* and rcat.*n* graph sets with 100% ratio of vertices to edges, show the average of approximately 6% improvement using spanning tree-based encoding.

On the other hand, the performance of vertex-based encoding is better than that of spanning tree-based encoding for dense graphs in which ratio of vertices to edges is less than 20%. Some G*n.p* and U*n.d* graph sets are examples of this category.

The efficiencies for graphs around with ratios of 20% are irregular. For example, the performance of spanning tree-based encoding is superior in U500.10, which has a 20.3% ratio, but the performance of vertex-based encoding is better in G500.02, which has a 21.2% ratio. Geometric random graphs are closer to real-world problems than random graphs, considering that the randomness of geometric random graphs is less than that of random graphs and the vertices connected with edges in geometric random graphs are locally clustered.

Therefore the superiority of spanning tree-based encoding will be expected for real-world problems which consist of sparse graphs having more than a 20% ratio of vertices to edges. In the paper, three methods - Kruskal-like, DFS, and BFS - are used to obtain spanning trees and the results differ slightly for each method, but they are not very different. The topic of what kind of algorithms for finding a spanning tree is efficient and how these algorithms influence the performance appears to be one of interests.

5 Conclusions

We proposed a new encoding method and investigated its performance comparing to a widely-used method for the MAX CUT problem. This study is the first trial of applying spanning tree-based encoding to optimization method for graph problems. To demonstrate the effectiveness of our proposed approach, experiments on three spanning tree-based encodings were conducted for benchmark graphs and could obtain performance improvement on sparse graphs.

We also found that the change of encoding method can make differences for optimization performance. In other words, edge-based encoding is advantageous for sparse graphs and vertex-based encoding is profitable for dense graphs.

The proposed approach can be applied to other graph partitioning problems, e.g., ratio-cut graph partitioning, which are similar to the MAX CUT problem. In particular, the proposed spanning tree-based encoding scheme has a merit on partitioning of sparse graphs, which appear widely in real-world applications. Further study will aim at the extension and refinement of the encoding schemes and their application to various graph sets.

Acknowledgments. This work was supported by National Research Foundation of Korea Grant funded by the Korea government (NRF-2011-0009958 and NRF-2009-0071419).

References

1. Kim, Y.-H., Kwon, Y.-K., Moon, B.-R.: Problem-independent Schema Synthesis for Genetic Algorithms. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 1112–1122. Springer, Heidelberg (2003)

2. Kim, Y.-H.: Linear transformation in pseudo-Boolean functions. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1117–1118 (2008)
3. Kim, Y.-H., Yoon, Y.: Effect of changing the basis in genetic algorithms using binary encoding. KSII Transactions on Internet and Information Systems 2(4), 184–193 (2008)
4. Kim, Y.-H., Yoon, Y.: Representation and recombination over nonsingular binary matrices. In: Proceedings of the World Summit on Genetic and Evolutionary Computation, pp. 855–858 (2009)
5. Alpert, C.J., Kahng, A.B.: Recent directions in netlist partitioning: a survey. Integration, the VLSI Journal 19(1-2), 1–81 (1995)
6. Bui, T.N., Jones, C.: Finding good approximate vertex and edge partitions is NP-hard. Information Processing Letters 42(3), 153–159 (1992)
7. Clark, L.H., Shahrokhi, F., Székely, L.A.: A linear time algorithm for graph partition problems. Information Processing Letters 42(1), 19–24 (1992)
8. Feige, U., Karpinski, M., Langberg, M.: A note on approximating Max-Bisection on regular graphs. Information Processing Letters 79(4), 181–188 (2001)
9. Fjällström, P.O.: Algorithms for graph partitioning: a survey. Linkoping Electronic Articles in Computer and Information Science 3 (1998)
10. Hagen, L., Kahng, A.B.: New spectral methods for ratio cut partitioning and clustering. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 11(9), 1074–1085 (1992)
11. Kim, Y.-H., Moon, B.-R.: Lock-gain based graph partitioning. Journal of Heuristics 10(1), 37–57 (2004)
12. Kučera, L.: Expected complexity of graph partitioning problems. Discrete Applied Mathematics 57(2-3), 193–212 (1995)
13. Powers, D.L.: Graph partitioning by eigenvectors. Linear Algebra and its Applications 101, 121–133 (1988)
14. Yan, J.-T., Hsiao, P.-Y.: A fuzzy clustering algorithm for graph bisection. Information Processing Letters 52(5), 259–263 (1994)
15. Antonio, S.M., Abraham, D., Juan, J.P., Raúl, C.: High-performance VNS for the max-cut problem using commodity graphics hardware. In: Proceedings of the 18th Mini Euro Conference on VNS (2005)
16. Cong, J., Labio, W.J., Shivakumar, N.: Multiway VLSI circuit partitioning based on dual net representation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15(4), 396–409 (1996)
17. Guattery, S., Miller, G.L.: On the quality of spectral separators. SIAM Journal on Matrix Analysis and Applications 19(3), 701–719 (1998)
18. Michel, J., Pellegrini, F., Roman, J.: Unstructured Graph Partitioning for Sparse Linear System Solving. In: Lüling, R., Bilardi, G., Ferreira, A., Rolim, J.D.P. (eds.) IRREGULAR 1997. LNCS, vol. 1253, pp. 273–286. Springer, Heidelberg (1997)
19. Venkatakrishnan, V.: Parallel computation of Ax and A^Tx. International Journal of High Speed Computing 6, 325–342 (1994)
20. Armbruster, M., Fügenschuh, M., Helmberg, C., Jetchev, N., Martin, A.: Hybrid Genetic Algorithm Within Branch-and-Cut for the Minimum Graph Bisection Problem. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 1–12. Springer, Heidelberg (2006)
21. Yoon, Y., Kim, Y.-H., Moon, B.-R.: A note on edge-based graph partitioning and its linear algebraic structure. Journal of Mathematical Modelling and Algorithms 10(3), 269–276 (2011)
22. Biggs, N.: Algebraic Graph Theory, 2nd edn. Cambridge University Press (1994)

23. Diestel, R.: Graph Theory, 3rd edn. Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg (2005)
24. Ford Jr., L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press (1962)
25. Karp, R.M.: Reducibility Among Combinatorial Problems, pp. 85–103. Plenum Press, New York (1972)
26. Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. Theoretical Computer Science 1(3), 237–267 (1976)
27. Sahni, S., Gonzalez, T.: P-complete approximation problems. Journal of the ACM 23(3), 555–565 (1976)
28. Vitányi, P.M.B.: How well can a graph be n-colored? Discrete Mathematics 34(1), 69–80 (1981)
29. Poljak, S., Tuza, Z.: A polynomial algorithm for constructing a large bipartite subgraph, with an application to a satisfiability problem. Canadian Journal of Mathematics 34, 519–524 (1982)
30. Haglin, D.J., Venkatesan, S.M.: Approximation and intractability results for the maximum cut problem and its variants. IEEE Trans. on Computer 40, 110–113 (1991)
31. Hofmeister, T., Lefmann, H.: A combinatorial design approach to maxcut. In: Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science (1995)
32. Goemans, M.X., Williamson, D.P.: Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. Journal of the Association for Computing Machinery 42(6), 1115–1145 (1995)
33. Homer, S., Peinado, M.: Design and Performance of Parallel and Distributed Approximation Algorithms for Maxcut. Journal of Parallel and Distributed Computing 46, 48–61 (1997)
34. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)
35. Kim, S.-H., Kim, Y.-H., Moon, B.-R.: A hybrid genetic algorithm for the MAX CUT problem. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 416–423 (2001)
36. Barahona, F., Grotschel, M., Junger, M., Reinelt, G.: An application of combinatorial optimization to statistical physics and circuit layout design. Operational Research 36, 493–513 (1984)
37. Pinter, R.Y.: Optimal layer assignment for interconnect. Journal of VLSI Computing Systems 1, 123–137 (1984)
38. Poljak, S., Tuza, Z.: Maximum cuts and largest bipartite subgraphs, vol. 20. American Mathematical Society (1993)
39. Boost Library, `http://boost.org`
40. Open Beagle, `http://beagle.gel.ulaval.ca`
41. Bui, T.N., Moon, B.R.: Genetic algorithm and graph partitioning. IEEE Transactions on Computers 45(7), 841–855 (1996)