An Efficient Simulated Annealing Algorithm For Graph Bisectioning

Y.C. Zhao L. Tao K. Thulasiraman M.N.S. Swamy
Faculty of Engineering and Computer Science
Concordia University
Montreal, Canada H3G 1M8

Abstract

A new simulated annealing algorithm for solving the graph bisectioning problem is proposed. We run our simulated annealing algorithm, the Kernighan-Lin algorithm, and the Saab-Rao algorithms on the same set of random graphs with 50 to 500 nodes and compare their performances. Experiments show that our simulated annealing algorithm provides lower bisection cost than the Kernighan-Lin algorithm and the Saab-Rao algorithms for all of the graphs and our algorithm takes less running time than the other algorithms mentioned above for all of the graphs with more than 100 nodes. For the simulated annealing approach, we conclude that sequential neighborhood search outperforms random neighborhood search in solving the graph bisectioning problem.

Key words: Graph bisectioning, simulated annealing, combinatorial optimization.

1 Introduction

An undirected graph $G = (V, E)$ consists of a set V of nodes and a set E of edges, where an edge $e_{u,v} = \{u, v\} \in E$ is an unordered pair of nodes u and v in V and is said to be incident on u and v. A weighting function on the set E is a mapping $w : E \rightarrow R$ from the set E to the set of real numbers with $w(e_{u,v})$ denoting the weight of an element $e_{u,v} \in E$. A set E is said to be weighted if it has a weighting function defined on it. A graph $G = (V, E)$ is said to be an edge-weighted graph if the set E is a weighted set. In this paper , we only consider the undirected edge-weighted graphs.

We say that V_1 and V_2 partition the set V if V_1 and V_2 are both nonempty, $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$. We denote the partition by the unordered pair (V_1, V_2). An edge $e \in E$ is said to be cut by a partition (V_1, V_2) if its two ends belong to V_1 and V_2 respectively. The cost of a partition for a graph is the sum of the weights of all the edges cut by the partition, i.e., $\sum_{u \in V_1} \sum_{v \in V_2} w(e_{u,v})$. The Graph Bisectioning (GB) [8] is to find a partition (V_1, V_2) of $G = (V, E)$ such that $|V_1| = |V_2|$ and the cost of (V_1, V_2) is minimized. The GB problem has extensive applications in VLSI placement and routing problems[1, 5, 12]. It has been shown to be NP-complete[13]. Many heuristics have already been proposed for the GB problem. The Kernighan-Lin algorithm is the recognized champion among the classical approaches to this problem. The recent GB algorithms proposed by Saab and Rao [14] are faster than the Kernighan-Lin algorithm but cannot outperform the latter in bisection cost. Most of the algorithms mentioned above follow a descent paradigm. They start with an initial solution. A neighbor of this solution is then generated by some mechanism and the change in cost is calculated. If the cost is reduced , the current solution is replaced by the neighbor; otherwise the current solution is retained. The process is repeated until no further improvement can be found in the neighborhood of the current solution. The obvious disadvantage of the descent paradigm is the possibility of being trapped in a local minimum which may be far from the optimal solution.

Simulated Annealing (SA), which has received much attention over the last few years [6], is a randomized heuristic approach designed to give a good, though not necessarily optimal, solution within a reasonable amount of computing time. Metropolis et al. [11] proposed SA and applied it to the field of statistical physics as a means of determining the properties of metallic alloys at given temperatures. Kirkpatrick et al. [7] and Cerny [2] first demonstrated the potential of SA to solve the combinatorial optimization problems independently.

SA starts with a random initial solution. It attempts to avoid being trapped in a local optimum by sometimes allowing the temporal acceptance of inferior solutions. The acceptance or rejection of an inferior solution is probabilistically determined by a random num-

[1]This work has been partially supported by the Canada NSERC research grant OGP0041648.

ber r uniformly distributed on the interval [0, 1], a control parameter t (temperature), and the increase in the value of the objective function. SA has been successfully applied to many combinatorial optimization problems such as VLSI placement and routing [9], quadratic assignments [15, 3], and the zero-one knapsack problem [4].

In this paper, we use SA to solve the GB problem and demonstrate its superior performance to the Kernighan-Lin algorithm and the Saab-Rao algorithms in both the bisection cost and the computation speed. In Section 2, we propose a SA algorithm for solving the GB problem. In section 3, we compare the bisection cost and the computation time of our SA algorithm with the Kernighan-Lin algorithm and the Saab-Rao algorithms by applying them to the same set of random graphs. This paper concludes with some comments on SA.

2 SA algorithm for the GB problem

For the GB problem, the solution space S includes all the points corresponding to the feasible solutions. Given a particular bisection corresponding to point p in S, the exchange of a pair of nodes belonging to different partitions will let us move in S from p to a neighboring point. SA starts with an initial feasible solution, and moves step by step towards a solution giving hopefully the minimum (or close to the minimum) bisection cost. SA differs from the descent algorithms in that SA attempts to avoid being trapped in a local minimum by sometimes accepting a feasible solution that increases the value of the objective function f. We accept a move in S unconditionally if it decreases the value of f, and accept a move probabilistically if it increases the value of f.

The following key issues play an important role in our SA implementation.

1. *Initialization.* We can start the algorithm with either a random solution or a solution resulting from another algorithm. Although we can get a better initial solution in the latter case, we have to pay the extra computation time for the initialization algorithm. Since theoretically SA will eventually converge to the global optimal solution independently of the initial solution, we take the random initial solution to start our algorithm.

2. *Acceptance function.* The probability of accepting a move that causes an increase Δ in $f(s)$ is determined by the acceptance function. Like most researchers we use the Metropolis acceptance function [11] $e^{(-\Delta/t)}$, where t is a control parameter corresponding to the temperature in analogy with physical annealing. Because $\Delta > 0$ and $t > 0$, $e^{(-\Delta/t)}$ is always smaller than 1. We generate a random number r uniformly distributed on the interval [0, 1] and compare $e^{(-\Delta/t)}$ with r. If $e^{(-\Delta/t)} > r$, then the move is accepted; otherwise the move is rejected.

3. *Neighborhood search.* There are two ways for selecting the next feasible solution from the set of neighbors of the current solution: random and sequential. In most published research on SA, the next feasible solution is randomly selected from the neighbors of the current solution. According to Connolly [3], the random neighborhood search might miss some potential improvements by the random nature of the search. We implement both approaches for comparison. For the sequential neighborhood search, the attempted node-exchange is examined in the order (1, 2), (1, 3), ..., (1, n), (2, 3), ..., ($n-1$, n), (1, 2), ... so as to increase the chance of improvement.

4. *Cooling schedule.* It includes the selection of the starting temperature t_0, the rate at which the temperature is reduced, the number of iterations at each temperature, and the criterion for stopping the algorithm. The temperature is initially set to some large value for permitting almost all attempted moves and gradually lowered such that the acceptance probability of inferior moves is gradually reduced until it approaches zero. In our algorithm the temperature is controlled by the Lundy and Mees scheme [10] because it reduces the temperature more smoothly. By this scheme, starting from an initial temperature t_0, the temperature is reduced after each attempted node-exchange by the following recurrence expression:

$$t_{i+1} = \frac{t_i}{1 + \beta t_i}, \qquad i = 0, \cdots, m-1$$

where m is a given number of iterations for stopping the algorithm. Therefore,

$$\beta = \frac{t_0 - t_m}{m t_0 t_m}, \qquad t_0 \gg \beta.$$

We use $m = k(n^2/4)$, where $n^2/4$ is the size of the neighborhoods and k is a positive integer. We choose t_0 and t_m equal to the maximum

and the minimum increases of $f(s)$ for successive current solutions s in a fixed number (for example, $(1/100)m$) of node-exchanges between the two partitions respectively.

Our SA algorithm is given as follows.

Input:
Graph $G = (V, E)$ and the $n \times n$ cost matrix w.
Cost function $f(s) = \sum_{u \in V_1} \sum_{v \in V_2} w(e_{u,v})$ for any partition $s = (V_1, V_2)$.
Starting temperature t_0, final temperature t_m, and the number of iterations m.
$\beta = (t_0 - t_m)/(mt_0t_m)$.

Algorithm:
Generate a random feasible solution $s = (V_1, V_2)$ such that $|V_1| = |V_2|$.
Let $t = t_0$, cost $= f(s)$, best_s$= s$.
for all $v \in V_1$, enqueue(Q_1, v)[2].
for all $v \in V_2$, enqueue(Q_2, v).
while $t > t_m$ **do**
  Let $n_1 =$ dequeue(Q_1)[3], $n_2 =$ dequeue(Q_2).
  Let $s' = (V_1 - \{n_1\} \cup \{n_2\}, V_2 - \{n_2\} \cup \{n_1\})$.
  Let $\Delta = f(s') - f(s)$.
  Let r be a random number uniformly distributed on $[0, 1]$.
  if $\Delta \leq 0$ or $e^{-\Delta/t} > r$ **then**
    enqueue(Q_1, n_2), enqueue(Q_2, n_1).
    if f(best_s) $>$ cost **then** best_s $= s'$.
    Let cost $= f(s')$, $s = s'$.
  Let $t = \frac{t}{1+\beta t}$.
end while
Output: best_s.

3 Performance comparison

We run our SA algorithm, the Kernighan-Lin algorithm, and the Saab-Rao algorithms on MIPS M120/5 for random graphs with node numbers ranging from 50 to 500. The random graphs have the following properties:

1. Connected.
2. The degree of each node ranges from 2 to $n/2$, where n is the number of nodes in the graph.
3. The weight of each edge ranges from 2 to 200.

Table 1 and Table 2 summarize the performances of these algorithms in cost and CPU time respectively.

n	Average Cost				
	K.-L. alg.	Saab-Rao alg.		SA alg.	
	(iter. #)	3.1	3.2+4.1	seq.	rand.
50	977(15)	924	975	923	927
100	4039(5)	3983	4120	3978	3980
150	9061(5)	8995	9383	8992	8995
200	16417(6)	16373	16946	16369	16373
250	24830(7)	24815	25728	24805	24815
300	36499(6)	36484	37290	36483	36482
350	49713(7)	49659	50495	49858	49662
400	63770(7)	63710	65616	63709	63707
500	98350(1)	98026	99590	98024	98026

Table 1: Comparisons for bisection cost

n	Average CPU time (sec.)				
	K.-L. alg.	Saab-Rao alg.		SA alg.	
	(iter. #)	3.1	3.2+4.1	seq.	rand.
50	3(15)	0	0	2	2
100	12(5)	3	0	6	8
150	52(5)	13	1	9	10
200	131(6)	34	2	25	29
250	265(7)	69	3	38	39
300	465(6)	114	5	57	76
350	747(7)	190	7	138	165
400	1104(7)	294	9	171	181
500	334(1)	663	15	250	352

Table 2: Comparisons for computation time

For the Saab-Rao algorithm, there are two parts: one is for the algorithm 3.1, another is for the algorithm 3.2+4.1 in which algorithm 3.2 is used to obtain an initial bisection for algorithm 4.1. For the SA algorithm, the two parts are for two versions of implementation different only in the neighborhood search scheme: sequential and random. The cost and the CPU time are the averages over ten graphs with the same number of nodes.

It can be seen from Table 1 that sequential SA outperforms random SA on bisection cost in all but two graphs. For the graphs with 300 and 400 nodes, the costs for sequential SA are greater than those for random SA for about 1.5 units which is not too significant. From Table 2, we see that sequential SA always takes less CPU time than random SA for all graphs. In sum-

[2]Procedure enqueue(Q, v) enters node v into queue Q.
[3]Function dequeue(Q) deletes and returns the first node in queue Q.

mary, sequential SA seems better than random SA in solving the GB problem.

Although the Saab-Rao algorithm 3.2+4.1 is the fastest algorithm, its solution quality is very poor. Table 1 shows that sequential SA generates bisections with lower costs than the Kernighan-Lin algorithm and the Saab-Rao algorithm 3.1 . For the graphs with 500 nodes, our sequential SA algorithm takes only about 1/1.4 and 1/2.6 CPU time of the Kernighan-Lin algorithm and the Saab-Rao algorithm 3.1 respectively.

4 Conclusion

In this paper, we use SA to solve the GB problem and compare the performance of our SA algorithm with those of the Kernighan-Lin algorithm and the Saab-Rao algorithms. We show that for all graphs with node numbers ranging from 50 to 500 , sequential SA algorithm outperforms all the other algorithms mentioned above in solution quality. For the graphs with more than 100 nodes, our sequential SA algorithm always takes less CPU time than the Kernighan-Lin algorithm and the Saab-Rao algorithm 3.1. For the SA algorithm, we compare the random neighborhood search and the sequential neighborhood search under the same cooling schedule for the same set of graphs with node numbers ranging from 50 to 500. The results show that the sequential neighborhood search outperforms the random neighborhood search in computation time for all the graphs and in solution quality for most graphs. We conclude that sequential SA algorithm is a very powerful tool for solving large-scale GB problem.

References

[1] S. Bhatt and F. Leighton. "A framework for solving VLSI graph problem." J. Comput. Syst. Sci., Vol. 28, No. 2, pp. 300-343, Apr., 1984.

[2] V. Cerny. "Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm." Journal of Optimization Theory and Applications, Vol. 45, pp. 41-45, 1985.

[3] D. T. Connolly. "An improved annealing scheme for the QAP." European Journal of Operational Research, Vol.46, pp. 93-100, 1990.

[4] A. Drexal. "A simulated annealing approach to the multiconstraint zero-one knapsack problem." Computing, Vol.40, pp.1-8, 1988.

[5] A. Dunlop and B. Kernighan. "A procedure for placement of standard-cell VLSI circuits." IEEE Trans. Computer-Aided Design, Vol. CAD-4, pp. 92-98, Jan., 1985.

[6] R. W. Eglese. "Simulated annealing: A tool for operational research." European Journal of Operational Research, Vol. 46, pp. 271-281, 1990.

[7] S. Kirkpatrick, C. D. Gelate Jr., and M. P. Vecchi. "Optimization by simulated annealing." Science, Vol. 220, pp. 671-680, 1983.

[8] B.W. Kernighan and S. Lin. "An efficient heuristic procedure for partitioning graphs." The Bell Technical Journal, Vol. 49, pp. 291-307, Feb., 1970.

[9] P. J. M. Van Laarhoven and E. H. L. Arts. "Simulated annealing : theory and applications." D. Reidel Publishing Company, Chapter 7, 1987.

[10] M. Lundy and A. Mees. "Convergence of an annealing algorithm." Mathematical Programming, Vol. 34, pp. 111-124, 1986.

[11] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. "Equation of state calculations by fast computing machines." Journal of Chemical Physics, Vol. 21, pp. 1087-1092, 1953.

[12] D. La Potin and S. Directer. "Mason: A global floorplanning approach for VLSI design." IEEE Trans. Computer-Aided Design, Vol. CAD-5, pp. 477-489, Oct., 1986.

[13] Y. Perl, M. Snir. "Circuit partitioning with size and connection constrains." Networks, Vol. 13, No. 3, pp. 365-375, 1983.

[14] Y. G. Saab and V. B. Rao. "Fast effective heuristics for the graph bisectioning problem." IEEE Trans. Computer-Aided Design, Vol. CAD-9, pp. 91-98, Jan., 1990.

[15] M.R. Wilhelm and T. L. Ward. "Solving quadratic assignment problem by simulated annealing." IIE Transactions, Vol. 19, pp. 107-119, 1987.