GRAPH PARTITIONING USING TABU SEARCH

Andrew LIM* Yeow-Meng CHEE†

Abstract

In this paper, we present another approach to the balanced minimum cut graph partitioning problem. This approach is based on the meta-heuristic known as tabu search. Our experimental results compare favorably with two of the most effecitive methods available in terms of quality of solutions and computational times. Our experience and experiments suggest that this technique can be applied effectively to many NP-hard combinatorial optimization problems.

1 Introduction

Let $G = (V,E)$ be a undirected graph with a cost $c(u,v)$ associated with each edge $(u,v) \in E$. We consider only graphs with $|V|$ even. The *balanced minimum cut graph partitioning problem* (BGPP) is to partition the vertices of G into two subsets of equal sizes such that the cut set has minimum cost, i.e. the sum of the cost of all those edges with end points in different subsets is minimum. It is clear that if we just want a cut set with minimum cost without regard to the size of the subsets in the partition, we can apply any polynomial time max-flow algorithm to obtain the optimal solution efficiently. However, if we restrict the sizes of the subsets, the problem becomes much harder: BGPP is NP-hard [GJ79]. BGPP has several useful applications in the areas of VLSI layout, systems, and computer networks.

In 1970, Kernighan and Lin [KL70] gave a heuristic for obtaining good solutions. After that, Fiduccia and Mattheyses [FM82] improved the speed of the Kernighan-Lin heuristic by a faster implementation using efficient data structures. We shall refer to this algorithm as the *KLFM algorithm.* Unfortunately, we observed that the performance of the KLFM algorithm is very erratic. Similar findings are reported in [Kri84, NOP87]. Another approach for obtaining good solutions is the *simulated annealing* approach [KGV83]. Although the annealing approach gives superior solutions, it is very time consuming. In addition, it is not clear to us how to choose effective parameters for the annealing algorithm without substantial simulation with test data. Lam and Delosme [LD88] gave an efficient implementation of an annealing-like algorithm for BGPP making use of a Kernighan-Lin-like heuristic in choosing moves. Such an adaptation gave a good speedup at the expense of solution quality in some test cases, whereas in other test cases noticeable improvement may be obtained.

In this paper, we present a viable technique called *tabu search* that is capable of obtaining solutions of competitive quality in high speed when compared to simulated annealing and the KLFM algorithm.

2 Tabu Search

Recently, a general technique, called *tabu search*, was proposed by Glover [Glo86, GG89, Glo89] for finding good solutions to combinatorial optimization problems. This technique is conceptually simple and elegant. It has also proven itself to be very useful in providing good solutions for large instances of many NP-hard problems in a reasonable amount of time [FHW89, HW89, WH89]. For the general framework of tabu search, refer to figure 1.

Input
An instance of the problem to be solved
Definitions
X : Set of feasible solutions
f : Objective function
$N(x)$: Neighborhood of $x \in X$
T : Tabu List(s)
A : Aspiration function
max : Maximum number of iterations between improvement
Initialization
Set $i = 0$;
Generate an initial solution $x_i \in X$;
Initialize tabu list(s) T and aspiration function A;
Set $best = x_i$, $bestcost = f(best)$ and $besti = i$;
Body
while (i - $besti < max$) {
$i = i + 1$;
locate the best x_i in $N(x_{i-1})$ where x_i does not satisfy tabu conditons or if aspiration function overrules tabu conditions;
if ($f(x_i) < bestcost$) {
$best = x_i$; $bestcost = f(best)$; $besti = i$;
}
update tabu list(s) T;
update aspiration function A;
}
Output
best and *bestcost*

Figure 1: General framework of Tabu Search

*Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A.
†Information Technology Institute, National Computer Board, 71 Science Park Drive, S0511, Republic of Singapore.

CH 3006-4/91/0000 – 1164 $1.00 © IEEE

Tabu search may be regarded as a "meta-heuristic" superimposed on another heuristic. The higher level heuristic organizes and directs the subordinate one. Although tabu seach and simulated annealing share the same property of being general iterative improvement techniques, the former does not resort to pure randomization to conquer intractability nor does it take the conservative approach that a proper rate of descent will lead us to a good local optimum, hopefully close to the the global one. Instead, it takes a more aggressive approach. Tabu search proceeds on the assumption that there is no value in choosing an inferior solution unless it is absolutely necessary, as in the case of getting out of a local optimum. At each iteration of the search, it selects the best neighborhood solution. This is unlike hill-climbing as it might make a down-hill move. Thus, the algorithm never runs out of choices for the next move. However, this approach may result in cycling, trapping the algorithm at locally optimal solutions. So two structures called *tabu lists* and *aspiration functions* are introduced. These two sturctures keep information about past moves in order to constrain and diversify the search for good solutions.

Tabu lists may be structured in many ways depending on the problem in question. The most simplified form of a tabu list is a linear list which stores the k most recent moves. The main purpose of the list is to constrain the direction of search. It prevents the algorithm from going back to a state which was reached previously. Such an action hopefully prevents us from being trapped in any local optimum. *Tabu conditions* are satisfied if the current move tries to undo a move previously made which is still in the tabu list. It is clear that if the search is unconstrained, upon leaving a local optimum, there is a very high probability that we may return to the same local optimum. In addition to tabu lists, we also have aspiration functions. An aspiration function has the ability to overrule tabu conditions. This serves as a mechanism to diversify the search and encourage the exploration of new regions in the search space.

3 Implementation of the Algorithm

The input to our balanced minimum cut graph partitioning algorithm is a graph $G = (V,E)$, and a cost $c(u,v)$ associated with each edge $(u,v) \in E$. We define the set of feasible solutions X to be

$$X = \{ (S_1,S_2) \mid S_1 \cup S_2 = V, |S_1| = |S_2| \},$$

and the objective function $f(x)$, $x = (S_1,S_2) \in X$, to be

$$f(x) = \sum_{u \in S_1,\, v \in S_2} c(u, v).$$

This is the simplest and most intuitive; Lam and Delosme [LD88], and Sechen and Chen [SC88] uses a different objective function which seems to perform very well also. The purpose is to minimize f(x).

In our implementation, we generate the initial solution randomly. We are currently investigating if clustering algorithms will help in improving the quality of solutions.

In our algorithm, we define $N(x)$ of $x \in X$ to be the set of configurations that can be reached from x via a single pair-exchange between members of the two different subsets of x. Naturally, the best neighbor of x corresponds to the member in $N(x)$ that gives us the highest decrement in cost. $N(x)$ can also be regarded as the neighborhood configuration of x. The following neighborhood search function was used in our experiments.

Method
We keep the subset S_1 sorted according to the linear order $\ll$: $u \ll v$ if and only if the decrement in cost when vertex u is moved from S_1 to S_2 is no smaller than the decrement in cost when vertex v is moved from S_1 to S_2. S_2 is sorted similarly. To make a move, we look for the pair of exchange that produces the greatest decrement in cost from the first K (a constant) elements of both S_1 and S_2. The time required to look for the exchange is $O(1)$ since K is a constant. Maintaining the sorted order of the two subsets S_1 and S_2 can be done in $O(n\log|V|)$ time, where n is the number of vertices adjacent to the two vertices that were chosen to be exchanged. In the worst case, $n = |V|$. So this method takes at most $O(|V|\log|V|)$ time.

There are several ways of structuring the tabu list to prevent cycling and to constrain the search direction. We took the simplest approach, we used a circular tabu list. It was observed from our test runs that when the length of the tabu list is between 7 and 13, good results were obtained.

Each member of the list stores an exchange that took place. For example, if the current exchange pair is (u,v), $u \in S_1$ and $v \in S_2$, we add (v,u) to the tabu list to replace the oldest member unless (v,u) is already in the tabu list (this can happen if a tabu condition is overruled by the aspiration function). The tabu list prevents a move in the near future to undo what was done before. It serves as some sort of short term memory to constrain the search. Of course, such a prevention is by no means absolute, as exchanges like (v,w) and (w,u) can still undo our previous exchange. However, such situations are unlikely to happen.

Another way to structure the tabu list may be just to keep a particular vertex that has just been involved in an exchange for at least K iterations of the algorithm before it can be exchanged again. However, we found that our structure of the list quite effective as indicated by our experimental results.

An aspiration function has the ability to overrule tabu conditions if a particular exchange is attractive even though it satisfies tabu conditions. This action diversifies the search. The aspiration function $A(y)$ we choose is a simple one. It is one less than the cost of the best configuration that has been reached from a configuration of cost y. Intuitively, it means that the tabu conditions are overruled if the new configuration has a lower cost than any other configuration reached from a configuration with current cost.

The updating of the aspiration function is given below:

```
if (A(f(x_{i-1})) > f(x_i)) {
        A(f(x_{i-1})) = f(x_i) - 1;
} else if (A(f(x_i)) > f(x_{i-1})) {
        A(f(x_i)) = f(x_{i-1}) - 1;
}
```

4 Experimental Results

4.1 Test Data

We conducted our experiments on two types of randomly

generated graphs. The first type is the *standard random graph* $R_{n,m}$, where n is the number of vertices and m is the number of edges. The m edges are randomly generated for the n vertices. The second type of graphs are the *geometric graphs*. Geometric graphs have clustering structures (see Figure 2). They may be closer to real applications. The geometric graph $G_{n,d}$ can be generated by algorithm G. We note that the expected degree of each vertex is approximately $nd^2\pi$. All edges have a cost of 1 in our test cases.

Input : n and d

step 1
Generate n points in a unit square.
Each point represents a vertex.

step 2
For any two points within Euclidean distance d, put an edge between the two points.

Algorithm G: To generate a geometric graph $G_{n,d}$

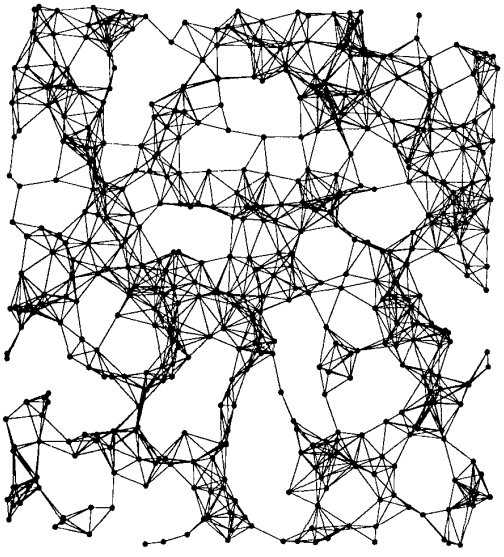

Figure 2: This is a geometric graph G with 500 vertices and expected degree 10

4.2 Our Competitors

We implemented both the simulated annealing algorithm and the KLFM algorithm. In the simulated annealing algorithm, the starting temperature is set at the value such that the moves attempted are accepted with a probability of 0.7. This value is determined in a preprocessing phase. The annealing schedule used is $T_k = \alpha T_{k-1}$, where α varies between 0.87 and 0.98. The algorithm is terminated when the temperature $T_k \leq 0.1$, or when the acceptance percentage drops below 0.1%. We do not claim our implementation of the annealing algorithm to be the best. Nevertheless, we have experimented with its parameters for a while and believe that they are reasonably good. We also approximated the computation of $\exp(-\Delta/T)$ by $1 - \Delta/T$ since $\exp(-\Delta/T)$ is computed many times. Considerable savings in computation times (as much as 30%) are made without noticeable degradation of the quality of solutions.

4.3 Test Runs

Since different starting configurations may result in different final solutions, we need to run our algorithm, the simulated annealing algorithm and the KLFM algorithm many times, each time with a different initial configuration. Currently, due to resource constraints, we are unable to run the simulated annealing algorithm as many times as we like in all the test cases. We will report our experiments once they are ready. We believe, however, that the annealing algorithm, unlike the KLFM algorithm, is less susceptible to the effects of different starting configurations.

In all our experiments, the KLFM algorithm and our algorithm were run 100 times for each graph. The annealing algorithm is run only once. Results are summarized in Tables 1 and 2. In the tables, k, Best, Ave, σ, and Time, are respectively, the expected degree of a vertex of the graph, the cost of the best solution, the average cost of solutions generated, the standard deviation, and the average run time for one test run over all runs. All times reported are in seconds.

All programs were written in Pascal and ran on a SUN SPARCstation 1 computer.

R-graphs		*Annealing*		*KLFM*				Our Algorithm			
n	k	Result	Time	Best	Ave	σ	Time	Best	Ave	σ	Time
250	10	341	2131	355	381	14	.25	349	370	11	.28
500	10	693	3978	726	763	16	0.9	724	754	14	1.0
1000	10	1385	7234	1484	1536	27	4.3	1475	1528	18	4.2

Table 1: Results on Random Graphs

G-graphs		*Annealing*		*KLFM*				Our Algorithm			
n	k	Result	Time	Best	Ave	σ	Time	Best	Ave	σ	Time
250	5	12	306	15	27.6	7.5	.24	12	25.6	7.2	.25
500	5	70	596	36	66.8	13.2	.95	37	64.4	13.2	1.0
1000	5	143	1017	75	133	23.7	3.8	53	111	22.8	4.1
250	10	28	711	42	87.2	24.2	.25	31	75.0	19.5	.27
500	10	130	1213	83	173	36.2	1.0	69	150	28.3	1.1
1000	10	246	1997	137	302	56.5	3.9	120	250	48.1	4.3
250	20	102	1899	102	205	47.3	.26	102	169	41	.30
500	20	220	2956	196	372	83.2	1.1	143	295	67	1.1
1000	20	560	5117	408	747	137	4.6	274	527	113	4.5

Table 2: Results on Geometric Graphs

5 Closing Remarks

Our experimental results indicate that our algorithm consistently outperformed the KLFM algorithm. Improvements in the quality of solutions can be as high as 33%. The speed of our algorithm is also comparable. Improvements on random graphs are small, but improvements on geometric graphs are very significant. Our algorithm is also less erratic.

When compared with the simulated annealing algorithm, our algorithm did not perform as well in terms of quality of solutions on random graphs, even though in most cases the results are close. However, our algorithm outperformed the annealing algorithm in almost all the test cases on geometric graphs. Our algorithm is also faster by two to three orders of magnitude.

Our results on BGPP gave very positive indications that tabu search may be used in providing good solutions to many NP-hard combinatorial optimization problems. Others have experienced similar findings in solving the node coloring, travelling salesman, and flow-shop scheduling problems. Since many of the computer-aided design (CAD) problems in VLSI, like placement, routing, and PLA folding, can be modelled as combinatorial optimization problems, more studies are needed to see in what context tabu search would work well on these problems. Investigations need to be conducted on the generation of neighborhood solutions, structuring of the tabu lists, definition of aspiration functions, updating of aspiration functions, and convergence properties of tabu search.

Studies comparing various general optimization techniques like simulated annealing, genetic algorithms, neural networks, and tabu search, can also be undertaken to compare the merits and applicability of these different techniques.

References

[FHW89] C. Friden, A. Hertz and D. de Werra (1989), STABULUS: a technique for finding stable sets in large graphs with tabu search, *Computing* **42**, 35-44.

[FM82] C.M. Fiduccia and R.M. Mattheyses (1982), A linear-time heuristic for improving network partitions, *Proceedings of the 19th Design Automation Conference* (ACM Press) 175-1982.

[GJ79] M.R. Garey and D.S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, CA).

[Glo86] F. Glover (1986), Future paths for integer programming and links to artificial intelligence, *Computer and Operations Research* **13**, 533-549.

[Glo89] F. Glover (1989), Tabu search - part I, *ORSA Journal on Computing* **1**, 190-206.

[GG89] F. Glover and H.J. Greenberg (1989), New approaches for heuristic search: a bilateral linkage with artificial intelligence, *European Journal of Operational Research* **39**, 119-130.

[HW89] A. Hertz and D. de Werra (1989), Using tabu search techniques for graph coloring, *Computing* **39**, 345-351.

[KGV83] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi (1983), Optimization by simulated annealing, *Science* **220**, 671-680.

[KL70] B.W. Kernighan and S. Lin (1970), An efficient heuristic procedure for partitioning graphs, *Bell System Technical Journal* **49**, 291-307.

[Kri84] B. Krishnanmurthy (1984), An improved min-cut algorithm for partitioning of VLSI networks, *IEEE Transaction on Computers* **33**, 438-446.

[LD88] J. Lam and J.M. Delosme (1988), Simulated annealing: a fast heuristic for some generic layout problems, *Proceedings of the International Conference on Computer-Aided Design* (ACM Press) 510-513.

[NOP87] T.K. Ng, J. Oldfield and V. Pitchumami (1987), Improvement of a mincut partition algorithm, *Proceedings of the International Conference on Computer-Aided Design* (ACM Press) 470-473.

[SC88] C. Sechen and D. Chen (1988), An improved objective function for mincut circuit partitioning, *Proceedings of the International Conference on Computer-Aided Design* (ACM Press) 502-505.

[WH89] M. Widmer and A. Hertz (1989), A new heuristic method for the flow shop sequencing problem, *European Journal of Operational Research* **41**, 186-193.