Advanced Engineering Forum *Online: 2011-09-09*
ISSN: 2234-991X, Vol. 1, pp 257-261
doi:10.4028/www.scientific.net/AEF.1.257

Local Refinement for Graph Partitioning Based on Level Structure

Yao Lu[a], Wang Zhenghua[b], Cao Wei[c] and Li Zongzhe[d]

School of Computer, National University of Defense Technology, Changsha, 410073, China

[a]shaoeq@163.com, [b]zhhwang@21cn.com, [c]caowei193@gmail.com, [d]lzz144@163.com

Keywords: Graph partition; Level structure; Local refinement

Abstract. Graph partitioning is a fundamental problem in several scientific and engineering applications. In this paper, we propose a heuristic local refinement algorithm for graph partitioning, which seeks to improve the quality of separators by reducing the width of the level structure. The experiments reported in this paper show that the use of our local refinement algorithm results in a considerable improvement in the quality of partitions over conventional graph partitioning scheme based on level structure.

1. Introduction

Graph partitioning is an important problem that has extensive applications in many areas, including VLSI design, parallel task scheduling, image segmentation, sparse matrix factorization, etc.

Given an undirected graph $G=(V, E)$, where V is the set of vertices, with $|V|=n$, and E the set of edges that determines the connectivity among the vertices. Without loss of generality, let the graph G be connected. A node separator is a node subset $S_v(S_v \subset V)$ whose removal divides V-S_v into k disconnected parts $V_1, V_2, \ldots, V_k$, so that $V_i \cap V_j = \varnothing$ for all $i \neq j$. An edge separator is an edge subset $S_E(S_E \subset E)$ whose removal divides V into k disconnected parts $V_1, V_2, \ldots, V_k$, so that $V_i \cap V_j = \varnothing$ for all $i \neq j$.

The graph partitioning problem is to find a minimal node or edge separator which divides the graph into k roughly equal parts. Specially when $k=2$, it becomes the well-known graph bisection problem which is only considered in this paper without loss of generality. Unfortunately, graph partitioning problem, even the bisection problem, is NP-complete[1]. So, many algorithms have been developed to find reasonably good separators.

It is difficult to quantify good separators. Generally speaking, they are small subsets of nodes or edges whose removal divides the graph approximately in half. For different classes of problems, the size of what is regarded as good separators varies. Lipton and Tarjan[2] show that any planar graph can be divided into components of no more than $2n/3$ vertices by removing a set of no more than $\sqrt{8n}$ vertices. Moreover, such a separator can be found in O(n) time.

As graph partitioning problem is NP-complete, different heuristic strategies have been proposed to solve the problem, classified into combinatorial approaches[3,4], based on geometric representations[5,6], multilevel algorithms[7,8], spectral bisection[9], level structures[10], simulated annealing[11], tabu search[12], and genetic algorithms[13].

In this paper, we propose a heuristic local refinement algorithm, based on successively optimizing level structure, for the graph partition problem.

The outline of this paper is as follows: In section 2, we present a partitioning scheme based on level structures. Section 3 introduces a new local refinement algorithm to improve the quality of separators. In section 4, we provide some comparative experimental results, which show considerable improvement in the quality of partitions.

2. Partitioning Scheme based on level structure

Let $G=(V, E)$ be a connected graph, where V is the set of vertices with $|V|=n$, and E the set of edges with $|E|=m$. Given a node $v \in V$, the level structure $L_v(G)$(or $L(G)$ for short) rooted at v is defined to be the subset sequence

$$L_0, L_1, L_2, \ldots, L_h$$

where $L_0=\{v\}$, and for level $i=1,...,h$,

$L_i=Adj_G(L_0 \cup ... \cup L_{i-1})$.

Note that $L_0 \cup L_1 \cup L_2 \cup ... \cup L_h=V$ if G is a connected graph. The essential properties of $L(G)$ are that all nodes adjacent to nodes in L_0 are in either L_0 or L_1; all nodes adjacent to nodes in L_h are in either L_h or L_{h-1}; for $0<i<h$, all nodes adjacent to nodes in L_i are in either L_{i-1}, L_i or L_{i+1}. The number of levels h is the length of a longest path from the node v_0 to other nodes in the graph, which is called length of the level structure. Correspondingly, $w_i(L)=|L_i|$(the number of nodes in L_i) is called the width of L_i, and $w(L)=max\{w_i(L)\}(i=0,...,h)$ is the width of the level structure $L(G)$.

In the level structure, any level $L_i(0<i<h)$ is a node separator and any $E_i(1 \le i \le h)$ is an edge separator by defining $E_i = \{(v_p,v_q)|v_p \in L_{i-1}, v_q \in L_i, \text{ and } (v_p,v_q) \in E\}$. In order to divide the graph into roughly equal parts, the separator is required choosing appropriately. We can choose the level L_j as a node separator such that

$j = max\{i: |L_0| + ... + |L_{i-1}| < n/2\}$.

Or choose E_j as a edge separator such that

$j = max\{i: |L_0| + ... + |L_{i-1}| \le n/2\}$.

The size of separator is also of great importance. Intuitively, we hope the level structure is as long and narrow as possible, which is, however, a NP-complete problem. Generally speaking, level structures of small width are usually among those of maximal length. Increasing the number of levels always decreases the average number of vertices in each level, and tends to reduce the width of the level structures as well. Ideally, generating level structures rooted at endpoints of a diameter is the best. However, there is no known efficient algorithm which can always finds such vertices. A heuristic scheme is given by Gibbs[14] to obtain the endpoints of a pseudo-diameter, which are a pair of vertices that are at nearly maximal distance apart. The pair of vertices are called pseudoperipheral nodes. After finding a pseudo-diameter, two long level structures L_u and L_v are constructed, which rooted at the pseudoperipheral nodes u and v respectively. These two level structures can be combined into a new level structure whose width is usually less than that of either of the original ones. According to the above mentioned procedure, we can construct a long and narrow level structure. The following problem is whether we can further reduce the width of the level structures. In next section, a local refinement algorithm is proposed to reduce the width of the level structures, which can further improve the quality of partitioning.

3. A New Local Refinement Algorithm

As mentioned above, reducing the width of level structures can further improve the quality of separators. In this section, we present a local refinement algorithm to narrow the level structures and thus to reduce the size of separators.

The basic idea of our algorithm is: find the levels that have the most nodes and move the nodes in these levels successively into other levels. There are two basic operations performed by the algorithm: lifting a node and dropping a node.

```
LIFT(v, h)
1  if h≤0 then
2     return
3  level[v] ← level[v] – h
4  for each vertex u∈ Adj_G(v) do
5     if level[u]-level[v]>1 then
6        LIFT(u, level[u]-level[v]-1)
```

Fig.1 LIFT(v, h) operation.

According to the definition in section 2, given a node $v \in L_k$, the basic operation $LIFT(v, h)$ moves v into L_{k-h}. Before v is lifted, all nodes in $Adj_G(v)$ can exist only in three levels: L_{k-1}, L_k and L_{k+1}. While after lifted, in order to ensure the integrality and validity of the level structure, some nodes in $Adj_G(v)$

need being lifted accordingly. Specifically, operation *LIFT(u, max{k-1+α, 0})* should be performed to each $u \in Adj_G(v)$ in $L_{k+\alpha}(\alpha=-1, 0, 1)$. Then, operation should be performed to $Adj_G(Adj_G(v))$. This pattern continues until the new level structure is built completely. Obviously, the operation *LIFT(v, h)* is a recursive process, which is described in Fig.1(*level[v]* is level number of node *v*).

Given a node $v \in L_k$, the operation *DROP(v, h)* moves *v* into L_{k+h}, which can be defined in the similar way. Fig.2 shows our algorithm by making use of the operations *LIFT* and *DROP*.

```
Local Refinement Algorithm:
1  calculate the length l and width d of the current level structure
2  find level Lk which has the most nodes in all levels
3  for each vertex v∈Lk do
4     for i ← 1 to k do
5        LIFT(v, i)
6        if the width is reduced and the length is not shortened then
7           set it to the current level structure
8           go to 1
9     for i ← 1 to l-k-1 do
10       DROP(v, i)
11       if the width is reduced and the length is not shortened then
12          set it to the current level structure
13          go to 1
```

Fig.2 Local refinement algorithm.

In step 2, there are perhaps more than one levels which have the most nodes. Optimization can be performed level by level for this case. Also, we tend to choose the vertex first whose degree is minimal in step 3. A practical approach is ordering all the nodes in L_k by degrees and operating them from minimum degree to maximum degree.

4. Experimental Results and Conclusions

In this section, we present experimental results on the quality of partitioning generated by our algorithm. The experiments are based on the set of real-world graphs from Matrix Market[15]. The main characteristics of the graphs that we have used in our experiments are shown in Table 1.

Table 1 Characteristics of the graphs that we use in the experiment.

Matrix/Graph	|V|	|E|	Avg. Degree	Description
gr_30_30	900	3422	7.60	Finite-difference Laplacians
nos3	960	7442	15.50	Lanczos with partial reorthogonalization
can_1072	1072	5686	10.61	Structures problems in aircraft design
bcsstk09	1083	8677	16.02	BCS Structural Engineering
bus_1138	1138	1458	2.56	Power system networks
jagmesh8	1141	3162	5.54	Graded L-shape patterns
dwt_1242	1242	4592	7.39	Structural engineering
bcspwr09	1723	2394	2.78	Power network patterns
s1rmq4m1	5489	137811	50.21	Finite element analysis of cylindrical shells

Table 2 compares the quality of partitioning with and without local refinement. *W(L)* is the width of the level structure *L*. |*NS*| and |*ES*| represent the sizes of node separator and edge separator respectively while N_1/N_2 and M_1/M_2 represent the ratio of the two parts of bisection accordingly. It is shown clearly that our algorithm is effective in narrowing the level structures from about 10% to 40%, thus reducing the size of separators to a certain extent. Correspondently, more balanced partitions can be achieved due to the reduction of separators.

Table 2 Comparison of the partitioning quality with and without local refinement.

Matrix/Graph	Initial algorithm					Improved algorithm				
	W(L)	|NS|	N_1/N_2	|ES|	M_1/M_2	W(L)	|NS|	N_1/N_2	|ES|	M_1/M_2
gr_30_30	59	43	1.06	123	1.04	46	31	1.01	89	1.08
nos3	78	62	1.00	334	1.13	56	48	1.01	256	1.10
can_1072	228	185	1.39	359	1.89	136	136	1.21	297	1.09
bcsstk09	111	81	1.02	486	1.14	99	72	1.08	419	1.06
bus_1138	120	120	1.21	135	1.47	80	74	1.10	84	1.04
jagmesh8	49	46	1.05	84	1.14	33	26	1.02	45	1.07
dwt_1242	123	123	1.05	301	1.27	80	60	1.01	139	1.12
bcspwr09	120	117	1.02	126	1.17	81	81	1.05	92	1.05
s1rmq4m1	348	252	1.08	4302	1.18	255	246	1.05	4194	1.15

But the algorithm also has some disadvantages and limitations. Firstly, it is only a heuristic which cannot ensure a minimum separator. Further, it is not appropriate to partition some undirected graphs by level structure, such as disconnect graphs, etc. In this case, it could not achieve good quality of separators, even if the above local refinement algorithm is used.

Acknowledgements

We thank the anonymous reviewers for their constructive comments and suggestions. This work was partially supported by the National Grand Fundamental Research 973 Program of China under Grant No.G2009CB723803.

References

[1] M.R. Garey, D.S. Johnson, L. Stockmeyer, Some simplified NP-complete graph problems, Theoretical Computer Science. 1 (1976) 237-267.

[2] R.J. Lipton, R.E. Tarjan, A separator theorem for planar graphs, SIAM J. Appl. Math. 36 (1979) 177-189.

[3] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphics, The Bell Sys. Tech. Journal. (1970) 291-307.

[4] C. Fiduccia, R. Mattheyses, A linear time heuristic for improving network partitions, In Proc. 19th IEEE Design Automation Conference. (1982) 175-181.

[5] H.D. Simon, S. Teng, How good is recursive bisection, SIAM J. Scientific Computing. 18 (1997) 1436-1445.

[6] J. Gilbert, G. Miller, S. Teng, Geometric mesh partitioning: implementation and experiments, In Proc. 9th Int. Parallel Processing Symposium. (1995) 418-427.

[7] G. Karypis, V. Kumar, Multilevel k-way partitioning scheme for irregular graphs, Journal of Parallel and Distributed Computing. 48 (1998) 96-129.

[8] J. Cong, M. Smith, A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design, In Proc. ACM/IEEE Design Automation Conference. (1993) 755-760.

[9] A. Pothen, H.D. Simon, K.P. Liu, Partitioning sparse matrices with eigenvectors of graphs, SIAM J. on Matrix Analysis and Applications. 11 (1990) 430-452.

[10] H.D. Simon, Partitioning of unstructured problems for parallel processing, Computing Systems in Engineering. 2 (1991) 135-148.

[11] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning, Oper. Res. 37 (1989) 865-892.

[12] E. Rolland, H. Pirkul, F. Glover, Tabu search for graph partitioning, Ann. Oper. Res. 63 (1996) 209-232.

[13] T.N. Bui, B. Moon, Genetic algorithm and graph partitioning, IEEE Transactions on Computers. 45 (1996) 841-855.

[14] N.E. Gibbs, W.G.Jr. Poole, P.K. Stockmeyer, An algorithm for reducing the bandwidth and profile of a sparse matrix, SIAM J. Numer. Anal. 13 (1976) 236-250.

[15] Matrix Market on http://math.nist.gov/MatrixMarket