Application of Fusion-Fission to the multi-way graph partitioning problem

Charles-Edmond Bichot

Laboratoire d'Optimisation Globale, École Nationale de l'Aviation Civile/Direction des Services de la Navigation Aérienne, 7 av. Edouard Belin, 31055 Toulouse, France,
bichot@recherche.enac.fr,
WWW home page: http://www.recherche.enac.fr/~bichot

Abstract. This paper presents an application of the Fusion-Fission method to the multi-way graph partitioning problem. The Fusion-Fission method was first designed to solve the normalized cut partitioning problem. Its application to the multi-way graph partitioning problem is very recent, thus the Fusion-Fission algorithm has not yet been optimized. Then, the Fusion-Fission algorithm is slower than the state-of-the-art graph partitioning packages. The Fusion-Fission algorithm is compared with JOSTLE, METIS, CHACO and PARTY for four partition's cardinal numbers: 2, 4, 8 and 16, and three partition balance tolerances: 1.00, 1.01 and 1.03. Results show that up to two thirds of time are the partitions returned by Fusion-Fission of greater quality than those returned by state-of-the-art graph partitioning packages.

1 Introduction

For ten years, the state-of-the-art method to solve the multi-way graph partitioning problem is the multilevel method. The multilevel method often used a graph growing algorithm for the partitioning task and a Kernighan-Lin type refinement algorithm. This method has been introduced in [HL95b,KK95,AHK97]. It is a very efficient process which is very fast too. It consists in reducing the number of vertices of the graph, which is sometimes very high (more than 100,000 vertices), by coarsening them. Then, a partition of the coarsenest graph (less than 100 vertices) is built, generally with a graph growing algorithm [KK98a]. After that, the vertices of the partition are successively un-coarsened and the partition refined with a Kernighan-Lin algorithm [KL70,FM82] or a helpful set algorithm [DMP95].

Graph partitioning has many applications. The most famous of them are parallel computing, VLSI design and engineering computation. Thus, graph partitioning is an important combinatorial optimization problem. Because of its great number of applications, there are different graph partitioning problems. The aim of this paper is to study the most classical of them, the multi-way graph partitioning problem, also called k-way graph-partitioning problem [KK98c]. The other graph partitioning problems, such as the Normalized-Cut partitioning problem

[SM00,DGK04] or the Ratio-Cut partitioning problems [DGK07], are not presented in this paper.

This paper presents an application of the Fusion-Fission method to the multi-way graph partitioning problem. The Fusion-Fission method was first created to solve the Normalized-Cut graph partitioning problem [Bic06,Bic07]. Because the multi-way graph partitioning problem and the normalized-cut graph partitioning problems are strongly related, it is interesting to evaluate the efficiency of the same method to both problems, as it has been done for the multilevel method [DGK07].

The work presented in this paper is the first adaptation of Fusion-Fission to the multi-way graph partitioning problem. Thus, all components of Fusion-Fission presented in [Bic07] do no appear in this preliminary adaptation. However, partitions found by this adaptation are quite good regarding partitions returned by state-of-the-art graph partitioning packages.

2 Graph partitioning

The multi-way graph partitioning problem consists in finding a partition of the vertices of a graph into parts of the same size, while minimizing the number of edges between parts. It is well-known that the multi-way graph partitioning problem is NP-complete.

The difficulty of the task is to keep the sizes of the parts equal while minimizing the edge-cut. The value which represents the difference between the sizes of the parts is named the balance of the partition. Because a small difference between the sizes of the parts may lead to a lower edge-cut [ST97], lots of results are presented with partitions not perfectly balanced.

Definition 1 (Partition of the vertices of a graph). *Let $G = (V, E)$ be an undirected graph, with V its set of vertices and E its set of edges. A partition of the graph into k parts is a set $P_k = \{V_1, \ldots, V_k\}$ of sub-sets of V such that:*

- *No element of P_k is empty.*
- *The union of the elements of P_k is equal to V.*
- *The intersection of any two elements of P_k is empty.*

The number of parts k of the partition P_k is named the cardinal number of the partition.

Assume that the graph G is weighted. For each vertex $v_i \in V$, let $w(v_i)$ be its weight. For each edge $(v_i, v_j) \in E$, let $w(v_i, v_j)$ be its weight. Then, the weight of a set of vertices $V' \subseteq V$ is the sum of the weight of the vertices of V': $w(V') = \sum_{v \in V'} w(v)$.

Definition 2 (Balance of a partition). *Let $P_k = \{V_1, \ldots, V_k\}$ be a partition of a graph $G = (V, E)$ into k parts. The average weight of a part is: $W_{average} =$*

$\frac{w(V)}{k}$. *The balance of a partition is defined as the maximum weight of all parts divided by the average weight of a part:*

$$balance(P_k) = \frac{\max_{V_i \in P_k} w(V_i)}{W_{average}} = \frac{k}{w(V)} \max_{V_i \in P_k} w(V_i) \ .$$

The objective function to minimize is the *cut* function. It is defined as the sum of the weight of the edges between the parts. More formally, let V_1 and V_2 be two elements of P_k:

$$cut(V_1, V_2) = \sum_{u \in V_1, v \in V_2} w(u, v) \ .$$

Then, the *cut* objective function is defined as:

$$cut(P_k) = \sum_{V_i, V_j \in P_k, i<j} cut(V_i, V_j) \ .$$

The partition which has the lowest *cut* value is the solution of the multi-way graph partitioning problem. However, because of the size of the graph to partition (several thousands of vertices), and because of the combinatorial nature of the problem, the partition with the lowest *cut* value can not be found. Thus, combinatorial optimization methods are used to solve this problem.

3 The Fusion-Fission adaptation to multi-way graph partitioning

Because principles of Fusion-Fission are described in [Bic07], this paper presents only succinctly this method. Fusion-Fission principles are based on nuclear force between nucleons. This force is responsible for binding of protons and neutrons into atomic nuclei. In the nature, the fifty-six particles of an iron nucleus are more tightly bound together than in any other element. Thus, the Fusion-Fission optimization process consists in splitting and merging atoms to create atoms of maximum binding energy. Nucleons of big atoms are merged into atoms with few nucleons.

An analogy with graph partitioning is easy. Let the nucleons be the vertices of the graph and the atoms the parts of the partition. The binding energy between two nucleons is the edge weight between the corresponding vertices. According to the Fusion-Fission process, parts of the partition are successively merged and split. Then, the cardinal number of the partition changes during the process. Resulting atoms of the Fusion-Fission process should be atoms of the same size. Which means that the final partition is perfectly balanced.

To be as close as possible to the process described before, the Fusion-Fission application to multi-way graph partitioning is an iteration process which works as follows: at each step of the process, a new partition $P_{l'}^{t+1}$ is created based on the preceding partition P_l^t. The fission process consists in splitting each part of the partition $P_{l'}^{t+1}$ into l parts. Because of its efficiency, the multilevel method has been chosen for the splitting. The fusion process consists in merging the $l' * l$ parts into a partition P' of l' parts. The merging can be viewed as graph partitioning problem where the vertices of the graph are the $l' * l$ parts. Thus, a multilevel method has been chosen for the merging too. Then, the partition P' is refined using a Kernighan-Lin type algorithm (KL). The resulting partition is the partition $P_{l'}^{t+1}$. The initial partition, P_k^0, is provided by the multilevel method.

The algorithm 1 presents the Fusion-Fission application to multi-way graph partitioning. The number of part of the new partition, l', changes at each iteration. We decided to force it to follow a binomial distribution centered in k. Then, a list of numbers which follow this binomial distribution is constructed at the beginning of the Fusion-Fission algorithm. Then, each iteration starts by selecting a new number of part l' in this list.

The multilevel method and the Kernighan-Lin type algorithm (KL in the algorithm 1) used are those of the pMETIS software and are both described in [KK98a]. The pMETIS software does not refer to the parallel implementation of METIS, but pMETIS is the name given of the recursive bisection software implemented in the serial METIS package.

The particularity of the Fusion-Fission algorithm is to find several partitions of different cardinal numbers. Moreover, for each partition found during the algorithm's iteration, refinement is a four-step process. The partition is first refined for a balance of 1.00, then for a balance of 1.01, and after, for balances of 1.03 and 1.05. This four-step refinement process greatly decrease the computation time of the algorithm. Since the algorithm code is not optimized as much as the multilevel softwares, its computation time is less relevant than partition quality.

4 Comparison with state-of-the-art graph partitioning packages

4.1 Benchmarks graphs

The performance of the Fusion-Fission adaptation to multi-way graph partitioning is evaluated on a wide range of tests graphs arising in different application domains. These tests graphs have been chosen among classical benchmarks in the literature of graph partitioning. Some of these benchmarks have been tested in some recent papers [BGOM03,SWC04,KcR04,DGK07]. These graphs are both vertex and edge unweighted. The characteristics of these graphs are described in table 1.

All of these benchmarks graphs can be downloaded at the University of Greenwich graph partitioning archive (May 2007):
http://staffweb.cms.gre.ac.uk/∼c.walshaw/partition/

All the experiments in this paper were performed on an Intel Pentium IV 3.0 GHz processor with 1 Go of memory, running a GNU/Linux Debian operating system.

Algorithm 1 Fusion-Fission

procedure FusionFission($G = (V, E)$, k, n, $pMETIS$, KL)
- $l \leftarrow k$
- $P \leftarrow pMETIS(G, k)$
- $P_k^0 \leftarrow P = \{P_1, \ldots, P_k\}$
- **for** $t = 1$ **to** n **do**
  - choose a new number of parts l'
  - $P_l^t = \{P_1, \ldots, P_l\}$
  - **for** $j = 1$ **to** l **do**
    - $V'_{l'} \leftarrow pMETIS(P_j, l')$
    - $V' \leftarrow V' \cup V'_{l'}$
  - **end for**
  - make a graph G' based on the set of parts V'
  - $P_{l'}^{t+1} \leftarrow pMETIS(G', l')$
  - $P' \leftarrow KL(P)$
  - **if** $l' = k$ **and** $cut(P') < cut(P)$ **then**
    - $P \leftarrow P'$
  - **end if**
- **end for**
- **return** P

end procedure

Table 1. Benchmark graphs characteristics.

Graph name	Size $\lvert V\rvert$	Size $\lvert E\rvert$	Degree min	Degree max	Degree avg	Description (source)
add20	2395	7462	1	123	6.23	20-bit adder (Motorola)
data	2851	15093	3	17	10.59	
3elt	4720	13722	3	9	5.81	2D nodal graph (NASA/RIACS)
uk	4824	6837	1	3	2.83	2D dual graph
add32	4960	9462	1	31	3.82	32-bit adder (Motorola)
bcsstk33	8738	291583	19	140	66.74	3D stiffness matrix (Boeing)
whitaker3	9800	28989	3	8	5.92	2D nodal graph (NASA/RIACS)
crack	10240	30380	3	9	5.93	2D nodal graph
wing_nodal	10937	75488	5	28	13.80	3D nodal graph
fe_4elt2	11143	32818	3	12	5.89	
vibrobox	12328	165250	8	120	26.81	Sparse matrix
bcsstk29	13992	302748	4	70	43.27	3D stiffness matrix (Boeing)
4elt	15606	45878	3	10	5.88	2D nodal graph (NASA/RIACS)
fe_sphere	16386	49152	4	6	6.00	
cti	16840	48232	3	6	5.73	3D semi-structured matrix
memplus	17758	54196	1	573	6.10	Memory circuit (Motorola)
cs4	22499	43858	2	4	3.90	3D dual graph
bcsstk30	28924	1007284	3	218	69.65	3D stiffness matrix (Boeing)
bcsstk31	35588	572914	1	188	32.20	3D stiffness matrix (Boeing)
bcsstk32	44609	985046	1	215	44.16	3D stiffness matrix (Boeing)
t60k	60005	89440	2	3	2.98	2D dual graph
wing	62032	121544	2	4	3.92	3D dual graph
brack2	62631	366559	3	32	11.71	3D nodal graph (NASA/RIACS)

4.2 Some graph partitioning packages

The quality of the partitions produced by the Fusion-Fission algorithm is compared with those generated on the same computer by several public domain graph partitioning softwares:

- The CHACO software [HL95a]. This software includes multilevel and spectral algorithms. Because it is more efficient than the spectral algorithm, only the multilevel algorithm of CHACO, described in [HL95b], is compared with Fusion-Fission.
- The JOSTLE software [Wal02]. It is based on a multilevel multi-way partitioning algorithm [WC00].
- The METIS package [KK98b]. This package provides both the pMETIS and the kMETIS softwares. kMETIS is a direct multi-way partitioning algorithm [KK98c]. pMETIS uses a recursive bisection algorithm [KK98a].
- The PARTY software [PD98]. This software is is based on a multilevel algorithm and a helpful-sets refinement algorithm [DMP95].

From all of this softwares, two have a balance parameter : JOSTLE and CHACO (with KL_IMBALANCE). The two others found partitions with a variable balance.

4.3 Comparisons between graph partitioning softwares

To be compared with the other algorithms, the Fusion-Fission algorithm has been limited to 2,000 iterations. Then, its runtime is between one minute and one hour. This computation time is quite long regarding those of graph partitioning packages which is often less than a second. There are some explanations to this deficiency. The Fusion-Fission algorithm has not been optimized. It makes four refinement steps instead of one (see section 3). However, the Fusion-Fission algorithm is not slow in comparison with metaheuristics applied to graph partitioning [BGOM03,SWC04] which have a computation time of several hours to several days.

Tables 2 and 3 present some comparisons between the public graph partitioning packages presented in section 4.2 and the Fusion-Fission algorithm. Four cardinals numbers have been chosen: $k = 2$, 4, 8 and 16. CHACO naturally finds partition perfectly balanced. Its results are compared with those of JOSTLE and Fusion-Fission for $balance = 1.00$. pMETIS (labeled pM. in tables 2 and 3) finds partitions for a balance number of 1.01, thus it is compared with JOSTLE and Fusion-Fission for this imbalance. kMETIS and PARTY are compared with JOSTLE and Fusion-Fission for $balance = 1.03$. When an algorithm do not find a partition for the given balance, the result is marked not available (N/A in tables 2 and 3).

In Tables 2 and 3, lines heading "Best" summarize the number of times the algorithms found the best partition quality over the 23 graphs of this benchmark, regarding results of the other algorithms for the same balance. Results show that Fusion-Fission outperforms the other algorithms in all cases except for $k = 8$ and

$k = 16$ with a balance of 1.00. In the two last cases, the Fusion-Fission algorithm does as well as the JOSTLE software. The Fusion-Fission algorithm has not been constrained to find perfectly balanced partitions even if it tries to do so. Thus, in a few cases it does not find perfectly balanced partitions. The Fusion-Fission algorithm is particularly good for the two smallest cardinals numbers, $k = 2$ and $k = 4$. It can be noticed that the JOSTLE software does almost as well as the other softwares, except when it is compared with pMETIS for $k = 16$ and $balance = 1.03$.

5 Conclusion

A new multi-way graph partitioning method has been presented in this paper. This method named Fusion-Fission is based on a previous work we made to solve the normalized cut graph partitioning problem [Bic06,Bic07]. The adaptation of Fusion-Fission to the multi-way graph partitioning problem uses the pMETIS multilevel algorithm and its Kernighan-Lin refinement algorithm.

This method has been compared with four state-of-the-art graph partitioning packages: JOSTLE, METIS, CHACO and PARTY. Classical benchmarks have been used. The partitions searched are of cardinal numbers 2, 4, 8 and 16, with a balance of 1.00, 1.01 and 1.03. Results show that up to two thirds of time are the partitions returned by Fusion-Fission of greater quality than those returned with state-of-the-art graph partitioning packages.

Since Fusion-Fission takes much longer time than state-of-the-art graph partitioning packages, it may bee difficult to used it for parallel matrix applications. However, it can be advantageously be used for fields where run-time is less of a concern, as VLSI layout or air traffic management problems.

References

[AHK97] Charles J. Alpert, Jen-Hsin Huang, and Andrew B. Kahng. Multilevel circuit partitioning. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 530–533, 1997.

[BGOM03] R. Baños, C. Gil, J. Ortega, and F.G. Montoya. Multilevel heuristic algorithm for graph partitioning. In *Proceedings of the European Workshop on Evolutionary Computation in Combinatorial Optimization*, pages 143–153, 2003.

[Bic06] Charles-Edmond Bichot. A metaheuristic based on fusion and fission for partitioning problems. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium*, 2006.

[Bic07] Charles-Edmond Bichot. A new method, the fusion fission, for the relaxed k-way graph partitioning problem, and comparisons with some multilevel algorithms. *Journal of Mathematical Modeling and Algorithms (JMMA)*, 6(3):319–344, 2007.

[DGK04] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kullis. Kernel k-means, spectral clustering, and normalized cuts. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 551–556, 2004.

Table 2. Comparisons between algorithms for cardinals numbers $k = 2$ and $k = 4$.

	balance = 1.00			*balance* = 1.01			*balance* = 1.03			
Graph	JOSTLE	CHACO	FF	JOSTLE	pM.	FF	JOSTLE	kM.	PARTY	FF
$k = 2$										
add20	734	740	**699**	729	725	**677**	721	788	750	**666**
data	241	279	**204**	241	218	**203**	241	244	233	**196**
3elt	95	92	**90**	95	108	**90**	95	129	136	**88**
uk	33	34	**21**	28	23	**21**	25	41	29	**19**
add32	12	28	**11**	**10**	21	**10**	**10**	50	23	**10**
bcsstk33	12621	10224	**10175**	12616	10205	**10175**	12409	14655	12071	**10069**
whitaker3	136	132	**127**	136	135	**126**	135	152	132	**126**
crack	207	209	**186**	196	187	**186**	199	278	222	**186**
wingnodal	**1739**	1828	1790	**1741**	1820	1748	**1724**	2054	1782	1735
fe4elt2	**130**	**130**	**130**	**130**	**130**	**130**	**130**	143	**130**	**130**
vibrobox	11436	**10346**	11866	**11424**	12427	11552	11511	18245	11975	**11440**
bcsstk29	2898	2917	**2843**	2898	2843	**2818**	2997	3904	N/A	**2818**
4elt	151	179	**143**	151	154	**139**	157	258	159	**138**
fesphere	466	422	**386**	466	440	**386**	468	568	386	**384**
cti	347	410	**334**	342	334	**318**	342	688	366	**318**
memplus	6141	6861	**5816**	6096	6337	**5816**	6047	6519	7604	**5574**
cs4	455	421	**414**	435	**414**	**414**	**406**	613	418	414
bcsstk30	6456	6447	**6407**	6599	6458	**6345**	6599	8297	6696	**6275**
bcsstk31	4044	4020	**2805**	4060	3638	**2718**	4191	4950	N/A	**2698**
bcsstk32	6764	5507	**4782**	7027	5672	**4747**	6888	5527	5520	**4747**
t60k	108	101	**100**	107	100	**83**	98	103	93	**78**
wing	956	952	**950**	**908**	950	**908**	**896**	1562	927	908
brack2	754	752	**738**	751	738	**714**	715	845	947	**691**
Best	2	2	21	5	2	21	5	0	1	20
$k = 4$										
add20	1238	1357	**1292**	1229	1292	**1211**	1255	1387	1281	**1202**
data	448	**433**	459	447	480	**430**	425	505	511	**420**
3elt	**212**	219	**212**	**210**	231	**210**	**201**	265	243	208
uk	69	63	**61**	67	67	**55**	67	85	62	**51**
add32	45	79	**36**	40	42	**33**	41	107	62	**33**
bcsstk33	**22130**	26191	23066	**22293**	23131	22652	**21590**	25493	22445	21853
whitaker3	417	**398**	406	403	406	**397**	406	575	399	**396**
crack	442	479	**382**	431	382	**378**	413	589	476	**371**
wingnodal	4073	3992	**3720**	4048	4000	**3659**	4048	4832	N/A	**3659**
fe4elt2	396	**356**	359	375	359	**351**	368	1780	437	**351**
vibrobox	21761	21087	**20282**	22156	21471	**19940**	21844	36206	N/A	**19825**
bcsstk29	9833	8831	**8826**	9122	8826	**8692**	9122	10851	N/A	**8523**
4elt	498	405	**378**	485	406	**351**	434	425	364	**342**
fesphere	**825**	868	844	**818**	872	825	**806**	1103	819	818
cti	1355	**1016**	1049	1357	1113	**1029**	1329	2294	1089	**976**
memplus	10696	11532	**10596**	10550	10559	**10436**	10470	10640	11406	**10182**
cs4	1194	**1132**	1154	1177	1154	**1102**	1162	1599	N/A	**1089**
bcsstk30	25825	**17013**	17443	25865	17685	**16816**	25438	24151	N/A	**16767**
bcsstk31	10190	10184	**8201**	10066	8770	**7812**	10134	15279	N/A	**7812**
bcsstk32	14890	14946	**12205**	14887	12205	**11340**	14887	16215	13333	**9924**
t60k	**240**	290	255	**229**	255	255	242	279	272	**227**
wing	**1922**	2161	1937	**1840**	2086	1937	**1824**	3454	N/A	1900
brack2	**3222**	3356	3705	3144	3250	**3109**	2999	4129	N/A	**2935**
Best	6	6	12	5	0	19	4	0	0	19

Table 3. Comparisons between algorithms for cardinals numbers $k = 8$ and $k = 16$.

	balance = 1.00			*balance* = 1.01			*balance* = 1.03			
Graph	JOSTLE	CHACO	FF	JOSTLE	pMETIS	FF	JOSTLE	kMETIS	PARTY	FF
					$k = 8$					
add20	**1853**	1881	1907	**1894**	1907	1907	**1836**	2130	2018	1907
data	798	**763**	842	800	842	**758**	756	N/A	791	**714**
3elt	462	393	**388**	433	388	**364**	418	527	432	**356**
uk	114	130	**101**	104	**101**	**101**	106	168	148	**101**
add32	**120**	139	N/A	105	**81**	**81**	106	351	N/A	**72**
bcsstk33	**36106**	41951	40070	36269	40070	**35579**	35961	44681	39071	**34919**
whitaker3	716	**712**	719	710	719	**692**	706	1047	759	**687**
crack	809	806	**773**	779	773	**721**	751	1047	808	**721**
wingnodal	**6070**	6152	**6070**	**6033**	6070	6070	**5965**	8335	6284	5976
fe4elt2	713	**651**	654	688	654	**646**	681	801	707	**641**
vibrobox	30103	33410	**28696**	31032	28177	**26162**	30247	43334	N/A	**25796**
bcsstk29	17391	**16887**	17534	16742	16555	**15181**	17234	24525	N/A	**15043**
4elt	674	701	**635**	612	635	**604**	656	827	722	**583**
fesphere	1351	1337	**1330**	**1280**	1330	1302	**1274**	1643	1277	1294
cti	2257	**1886**	N/A	2158	2110	**2076**	2086	3888	2482	**2005**
memplus	**12866**	13956	13110	**12684**	13110	13110	**12540**	N/A	13119	13110
cs4	**1703**	1808	1746	**1673**	1746	1746	**1588**	2733	1721	1746
bcsstk30	39271	**35647**	N/A	38746	**36357**	**36357**	38228	41052	48539	**35668**
bcsstk31	**15360**	17553	16012	17094	16012	**14754**	19849	20647	N/A	**14754**
bcsstk32	29281	25810	**23601**	26655	**23601**	**23601**	25343	39817	N/A	**23601**
t60k	**556**	593	561	**532**	561	561	**530**	1309	581	561
wing	**3028**	3221	3205	**2918**	3205	3205	**2911**	5748	N/A	3205
brack2	8007	8061	**7844**	8037	**7844**	**7844**	**7757**	10171	N/A	7844
Best	9	6	9	7	5	16	8	0	0	15
					$k = 16$					
add20	2555	2269	**2504**	2532	**2504**	**2504**	2565	N/A	2510	**2504**
data	1299	**1279**	1309	1299	1370	**1278**	1263	4857	1475	**1224**
3elt	645	**641**	665	621	665	**607**	603	969	754	**598**
uk	**211**	**211**	N/A	190	**189**	**189**	**180**	384	220	189
add32	269	217	**128**	239	**128**	**128**	180	N/A	N/A	**128**
bcsstk33	59884	61800	**59791**	61505	59791	**58694**	**57553**	123044	61630	58183
whitaker3	**1172**	1241	1237	**1138**	1237	1180	**1147**	1436	1277	1165
crack	**1245**	1277	1255	1212	1255	**1197**	1191	2296	1263	**1187**
wingnodal	**9083**	9327	9290	9091	9290	**8962**	8947	10097	9006	**8890**
fe4elt2	1194	**1095**	1152	1146	1152	**1083**	1140	1500	1236	**1076**
vibrobox	**35447**	42634	37441	**36233**	37441	36398	**34521**	N/A	N/A	35809
bcsstk29	28294	**26239**	N/A	28062	28151	**26422**	28338	147143	N/A	**25417**
4elt	1081	1099	**1056**	**1034**	1056	1048	**1012**	5077	1282	1015
fesphere	**1918**	2061	2030	**1759**	2030	1952	**1741**	2495	N/A	1943
cti	3402	**3122**	3181	3345	**3181**	**3181**	3262	4760	N/A	**3181**
memplus	**14510**	15654	N/A	15085	**14942**	**14942**	**13958**	15804	14831	14942
cs4	**2518**	2550	2538	**2519**	2538	2538	**2477**	3614	2488	2538
bcsstk30	87824	**79046**	N/A	87472	**77293**	**77293**	81764	93834	N/A	**76791**
bcsstk31	27897	29364	**27180**	27954	**27180**	**27180**	27388	189562	N/A	**27180**
bcsstk32	46954	46266	**43371**	48162	**43371**	**43371**	48395	50660	N/A	**43371**
t60k	**977**	1027	N/A	**969**	998	998	**984**	1347	1097	998
wing	4761	4806	**4666**	**4623**	4666	4666	4681	7712	N/A	**4666**
brack2	13318	13117	**12655**	13337	**12655**	**12655**	13164	150514	N/A	**12655**
Best	9	7	8	7	9	16	9	0	0	14

[DGK07] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007. To appear.

[DMP95] R. Diekmann, B. Monien, and R. Preis. Using helpful sets to improve graph bisections. In *Proceedings of the DIMACS Workshop on Interconnection Networks and Mapping and Scheduling Parallel Computations*, pages 57–73, 1995.

[FM82] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[HL95a] Bruce Hendrickson and Robert Leland. *The Chaco User's Guide*. Sandia National Laboratories, 2.0 edition, 1995.

[HL95b] Bruce Hendrickson and Robert W. Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of Supercomputing*, 1995.

[KcR04] Peter Korošec, Jurij Šilc, and Borut Robič. Solving the mesh-partitioning problem with an ant-colony algorithm. *Parallel Computing*, 30(5-6):785–801, 2004.

[KK95] George Karypis and Vipin Kumar. Analysis of multilevel graph partitioning. In *Proceedings of Supercomputing*, 1995.

[KK98a] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of Scientific Computing*, 20(1):359–392, 1998.

[KK98b] George Karypis and Vipin Kumar. *Metis : A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*. University of Minnesota, 4.0 edition, sep 1998.

[KK98c] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[KL70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

[PD98] Robert Preis and Ralf Diekmann. *The Party Partitioning Library, User Guide*. University of Paderborn, 1.99 edition, oct 1998.

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[ST97] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

[SWC04] Alan J. Soper, Chris Walshaw, and M. Cross. A combined evolutionary search and multilevel optimisation approach to graph-partitioning. *Journal of Global Optimization*, 29:225–241, 2004.

[Wal02] Chris Walshaw. *The serial JOSTLE library user guide*. University of Greenwich, 3.0 edition, jul 2002.

[WC00] Chris Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22:63–80, 2000.