Electronic Transactions on Numerical Analysis.
Volume 21, pp. 125-133, 2005.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

THE INFLUENCE OF RANDOM NUMBER GENERATORS ON GRAPH PARTITIONING ALGORITHMS*

ULRICH ELSNER†

Dedicated to Alan George on the occasion of his 60th birthday

Abstract. Many modern heuristic algorithms rely at least in some part on random numbers. Using graph partitioning algorithms as an example, we demonstrate the often underestimated influence of these random-number generators on the result of the heuristic algorithms. Even methods that rely on random numbers mostly as a tie-breaking tool can deliver very different results depending only on the starting value of the pseudo-random number generator.

Key words. graph partitioning, heuristic algorithms, pseudo random-number generator

AMS subject classifications. 05C85, 90C59, 94C15, 65Y99, 68R10

1. Introduction. When trying to solve a given problem on an parallel computer with distributed memory, the way the data is distributed to the different processors can have a huge impact on the performance. Obviously, one tries to distribute the data in such a way that the work each processor has to do is about equal. But all but the most easily parallizable problems also require communication between different processors. In many cases, both the amount of communication and the communication pattern itself (which processor needs to communicate with which) depend on the way the data is distributed.

The problem of how to optimize the distribution of data to the different processors can be expressed as a graph partitioning problem. Solving this problem and distributing the data accordingly can lead to a huge speedup compared to just using a simple equal-size distribution without regard to communication [6].

The "graph partitioning problem" as we consider it in this paper is the problem of dividing the vertices of a given graph in n partitions of equal size such that the number of edges between vertices in different partitions (the so-called cutsize) is minimized. Quite often one only considers $n = 2$ and then applies this bisection recursively. While this can be arbitrarily worse than partitioning directly in $n = 2^k$ partitions, for many common graph families it can be shown (cf. [21]) that this recursive bisection is not much worse than direct partitioning into n parts.

While data distribution in parallel computation is the application that has attracted the most attention in recent years, graph partitioning can also be used to solve problems in many different areas.

One of the first algorithms, the Kernighan–Lin algorithm, was developed to assign the components of electronic circuits to circuit boards such that the number of connections between boards is minimized [15]. Other applications include VLSI (Very Large Scale Integration) design [12, 17], CAD (Computer Aided Design) [17], decomposition [10, 19, 18, 20] or envelope reduction [1] of sparse matrices, hypertext browsing [2], information retrieval [9], geographic information services [13] and physical mapping of DNA [11].

Unfortunately, finding the optimal solution for the graph partitioning problem is known to be NP-hard for all non-trivial cases and the decision formulation is NP-complete [7]. Simply put, this means that there is no known algorithm that is much faster than trying all possible combinations and there is little hope that one will be found.

*Received May 27, 2004. Accepted for publication October 25, 2005. Recommended by A. Pothen.
†DB-Systems GmbH, Frankfurt (Main), Germany (`na.uelsner@na-net.ornl.gov`).

ETNA
Kent State University
etna@mcs.kent.edu

A little more exactly (but still ignoring some fundamentals of the theory of NP-completeness) it means that graph bisection falls in a huge class of problems all of which can be transformed into each other and for which there are no known algorithms that can solve the problem in polynomial time in the size of the input. And even worse, even finding a good (asymptotically optimal) approximate solution can be NP-hard ([3]). Given a graph $(\mathcal{V}, \mathcal{E})$ without special properties and $k = |\mathcal{V}| + |\mathcal{E}|$ it is suspected that there exists no algorithm for solving the graph bisection problem which runs in $\mathcal{O}(k^n)$ time for *any* given $n \in \mathbb{N}$.

So, instead of finding the optimal solution, we resort to *heuristics.* That is, we try to use algorithms which may not deliver the optimal solution every time but which will give a good solution at least most of the time. An overview and comparison of many different approaches can be found in [6]. While some of the discussions below assume an understanding of the algorithms involved, the results can easily be understood without any closer knowledge of the different approaches.

2. Graph partitioning and random number generators. Many of these algorithms make use of a (pseudo-)random number generator at some stage. In their coarsening phase, multilevel algorithms such as some of Chaco's [11] algorithms, Metis[14] or Jostle [22] may decide randomly (within limits) which vertex to consider next and may choose which edge to contract randomly amongst all or amongst some candidates. Kernighan-Lin [15] variants and "helpful set"-algorithms [5] randomly choose which vertices/sets to exchange or add. In Geometric (or Geometric Spectral) Bisection [4, 8] the hyperplane defining the great circle is chosen randomly.

But how does this influence the performance of the algorithms? If one just uses the different implementations one can find on the web as documented, there will be no recognizable impact. If one partitions a graph several times with the same parameters one will always get the same cutsize. So, from a normal user's point of view, the algorithms show no random behavior. Unfortunately, this is misleading. This non-random behavior happens because the standard random number generators in computers are not really random. They are only deterministic programs that try to produce numbers that seem more or less random [16]. But still, they are deterministic programs and given the same starting conditions (often called *seed*), they will always produce the same sequence of "random" numbers. As far as we could check, all the programs tested always start with the same initial conditions. This explains their seemingly deterministic behavior but it also hides the influence of randomness on the algorithms.

To examine these influences, we ran some algorithms with ten different initial seeds for the random number generator. The algorithms we examined were:

- Multilevel Bisection: Here the random number generator should influence the coarsening phase, the low-level partition and the refinement phase. We tested both Chaco [11] in default configuration and pmetis [14].
- Multilevel Spectral Bisection [4, 8] with Kernighan-Lin refinement. For this combination, the main influence is probably the Kernighan-Lin part. While the multilevel coarsening phase is influenced by random choice as well, the Rayleigh Quotient Iteration used to compute the Fiedler vector should cancel many of the influences because the end result is deterministic up to scaling. (The only exception should be if the invariant subspace in which the Fiedler vector lies has higher dimension.)
- Inertial Bisection with Kernighan-Lin refinement as implemented in Chaco [11]. Here the global method has no random component. All the influence of the random numbers will occur in the Kernighan-Lin phase.

The tests were done on 31 different graphs, many of them from "real life" applications, whose sizes range from 615 to over a million vertices and whose average degrees range from

ETNA
Kent State University
etna@mcs.kent.edu

less than 3 to over 32. A detailed description of the test data, technical details of the tests, such as the exact settings of tunable parameters, and much more detailed results can be found in [6]. Inertial Bisection requires coordinate information of the vertices which was not available for some graphs (4, 18-22). On these graphs only the first two algorithms were used.

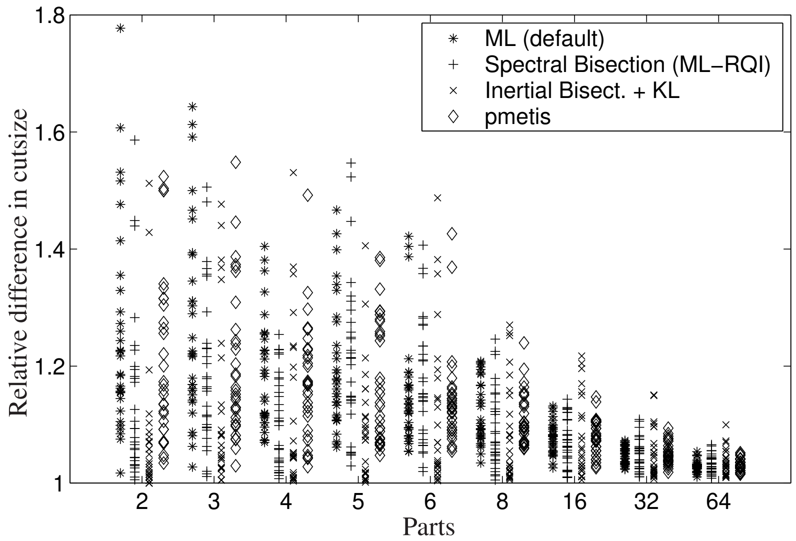

FIG. 2.1. *Random deviations*

3. Influence of starting seeds. Now we examine the results. First consider Figure 2.1. For each combination of algorithm, graph and number of parts we compute the ratio between the largest and smallest cutsize we achieved with different random number generator seeds. So, looking in the upper left corner of Figure 2.1, the '*' indicates that there is a graph whose worst cutsize when partitioned into two parts is about 80% worse than its best cutsize. (In fact, the graph is No. 16 and its best and worst cutsizes were 512 and 926, respectively, but these numbers cannot be seen in the picture.)

It should also be added that this plot does not show the worst offender, since plotting it would have compressed the y-scale so much that many details would have been lost. When trying to use Spectral Bisection to partition the graph No. 12 into three parts, the worst cutsize (2119) was about 5 times worse than the best(427). Chaco did issue a warning about possible misconvergence of the Rayleigh Quotient Iteration in this particular run, but this happened several other times (on the same and on other graphs) as well with much less drastic consequences.

Figure 2.1 is rather disheartening. For all algorithms examined we have large deviations in the cutsize for several graphs. Comparing this data with similar plots that highlight the difference between different algorithms, we see that the influence of the random number generator dwarfs many of the algorithmic differences. So one should always remember that a statement like "Algorithm A delivers a 5% better cutsize than Algorithm B" is quite meaningless if when we rerun the algorithms with different seeds the results differ by much more than 5%.

In Figure 3.1, we take a closer look on how the cutsizes vary. Because they showed the widest variations in Figure 2.1, we now only examine the partitioning into two parts (partitioning into other number of parts show a similar, albeit less pronounced behavior).

ETNA
Kent State University
etna@mcs.kent.edu

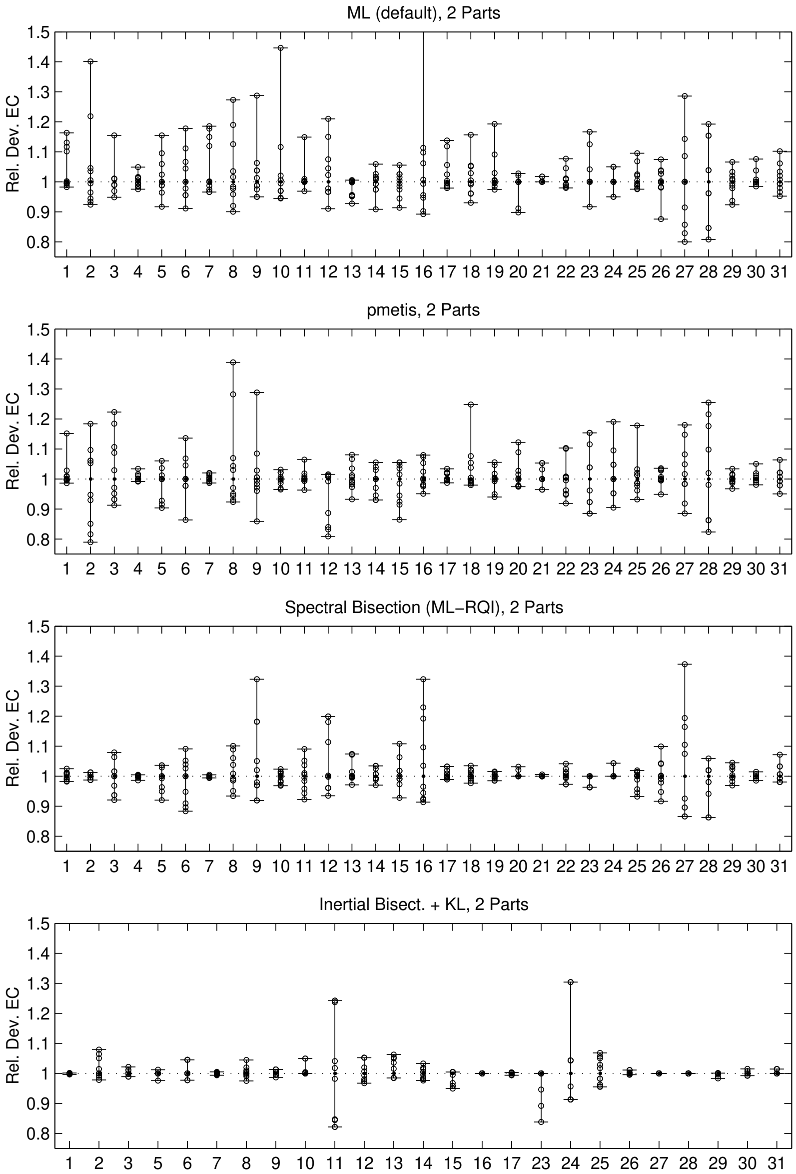

FIG. 3.1. *Random deviations for different graphs*

For increasing number of partitions, the fact that the variations decrease with the number of partitions seems to indicate that in the recursive application of the algorithms the variations are not independent but 'even out', such that a bad result in one step increases the chance of a good result in a succeeding step.

For each graph and each algorithm ,we plot the ratio of each single cutsize with the median of all ten cutsizes computed for this particular combination as well as, the error-bars showing the range of deviation from the median. As an example, looking at the leftmost column in the top subplot, we see that Chaco's Multilevel algorithm applied to graph No. 1 the lower error-bar is very close to the median, indicating that the smaller half of the cutsizes produced are almost the same while the larger half of the cutsizes vary by up to 15%. On the other hand, for graph No. 27 the cutsizes produced by varying the random number generator seed deviate by about 20% in both directions from the median. So graph No. 1 seems to have a "good" bisection that is found more than half of the time while No. 27 has no such "obvious" partition.

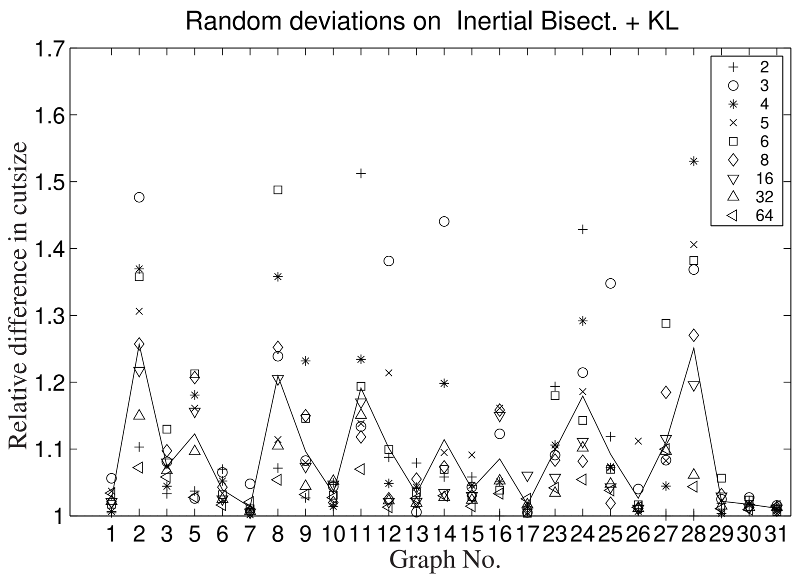

FIG. 3.2. *Random deviations for different graphs (2)*

Figure 3.1 shows several things. First, one notices that for many graphs, the distribution is unsymmetrical with the upper error-bar being much longer than the lower. This indicates that many graphs have a "good" bisection that is found rather often but not all the time. And looking closer at those graphs with very long error-bars, we see that often only one of ten tries is responsible for the large deviation while the other nine results cluster much closer. This is hopeful in so far as it implies that to avoid bad results, we often will need to try only a few different random number generator seeds to get a "good" result. On the other hand, some of the graphs (such as No. 27 and 28) in the case of the Multilevel algorithms and Spectral Bisection) have their result well spread out. So, some graphs seem to react well to certain algorithms while others do not. And looking at Figure 3.2 (which shows the maximal variation for different graphs and different numbers of partitions) we see that this difference in sensitivity to the random number generator seed for different graphs is not limited to two partitions. For some graphs the cutsize varies a lot with different seed, regardless of the number of partitions we seek, while other graphs deliver almost the same cutsize all the time. Figure 3.2 only looks at the results for Inertial Bisection but the same type of plots for Multilevel algorithms and for Spectral Bisection shows similar tendencies, i.e., some graphs display wide variation while for others most results fall into a (percent-wise) narrow range), although they are not quite as pronounced. The strong and weak reactions to the random seed can also appear at different graphs. This can also be observed in Figure 3.1. The distribution of long and short error-bars is quite different for different types of algorithms. The patterns for the two multilevel algorithms are somewhat similar but differ from the pattern for the Spectral Bisection and again from the pattern for Inertial Bisection.

To further study the influence of the random number generator, we chose some small and medium sized matrices that looked interesting in Figure 3.1 and bisected them using different methods, namely Chaco Multilevel, pmetis and Kernighan-Lin on a random starting partition. Each method was applied 10 000 times using different random number generator seeds. To test "pure" Kernighan-Lin as well, we asked Chaco to randomly divide the vertices into two equal-sized partitions and used this distribution as a starting point for an aggressive variant of Kernighan-Lin (more tries before giving up in both the inner and outer loop). The random-number generator used for these tests was the rand() function from the Linux operating system. Short experiments with other types of random number generators indicated no noticable influence.

The results of these experiments were both interesting and disheartening. For all tested graphs both multilevel algorithms displayed a huge spread. In all cases the worst cutsize was between 82% and 132% bigger than the best cutsize. It is thus quite possible that a usually "good" algorithm returns a partition with a cutsize twice as big as necessary. But luckily, a bad result like this happens only rarely. A look at some of the distributions (Figure 3.4 being a typical example) shows that the tail towards the "big end" is very thin.

TABLE 3.1
percentile worse than minimum cutsize

	Chaco ML				Metis			
	100%	99%	90%	50%	100%	99%	90%	50%
worst result	2.32	1.82	1.53	1.29	2.32	1.65	1.39	1.26
best result	1.92	1.34	1.10	1.03	1.82	1.27	1.16	1.09
ava.result	2.13	1.58	1.33	1.14	2.01	1.53	1.31	1.14

Only very few of the random seeds produce truly bad results. This is detailed in Table 3.1 which displays deviation from the best result for some percentiles. Looking for example at the values for Chaco, we see in the 100% column that for the "worst" random seed the cutsize was (on average over all graphs) 2.13 times as big as the best and even in the worst case the worst cutsize was 2.32 times as big as in the best. The 99% column indicates that the 9900th-best result (i. e., only 100 seed resulted in a worse cutsize) is on average 58% worse than the best and in the worst case 82% worse. It is interesting to note that the best and worst results are achieved on different graphs for the different algorithms.

Studying the data of the computations closer, we notice some surprising differences in the distribution of the cutsize. In many cases, plotting the number of occurrences of each cutsize will result in a distribution similar to that in Figure 3.3 with a huge, rather narrow peak near the optimum and then a very thin tail (note the given max. cutsize).

Some of the graphs result in a very different distribution that is nevertheless similar for both multilevel algorithm. For example, the results for both Chaco and Metis for graph No. 5 (see Figure 3.4 for Chaco's plot) displays two distinctive peaks.

But even this similarity is not always given. For example for graph No. 8, Chaco displays three distinctive high peaks while Metis only has one peak. Both methods have a rather fat tail. It seems that for this graph Metis is better at avoiding being trapped in a local minima.

For another graph (No. 16), the reverse is true. The Metis distribution looks quite familiar, with one main peak and two lower secondary peaks. But Chaco's distribution is extreme. Four singular peaks show up very distinctly, dwarfing the other results with each of these bars having a width of only one.

Is seems that there are several very well defined local minima that Chaco (which uses an implementation of the Kernighan-Lin algorithm as local optimizer) converges to while

ETNA
Kent State University
etna@mcs.kent.edu

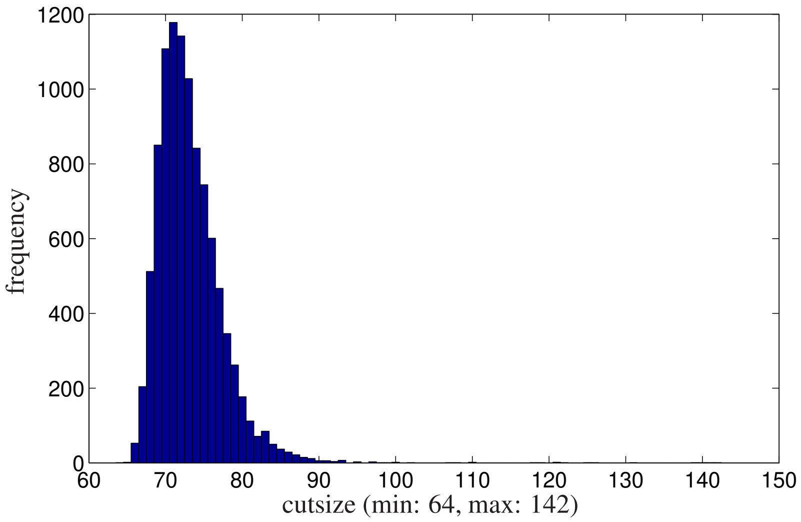

FIG. 3.3. *Chaco ML on graph No. 31*

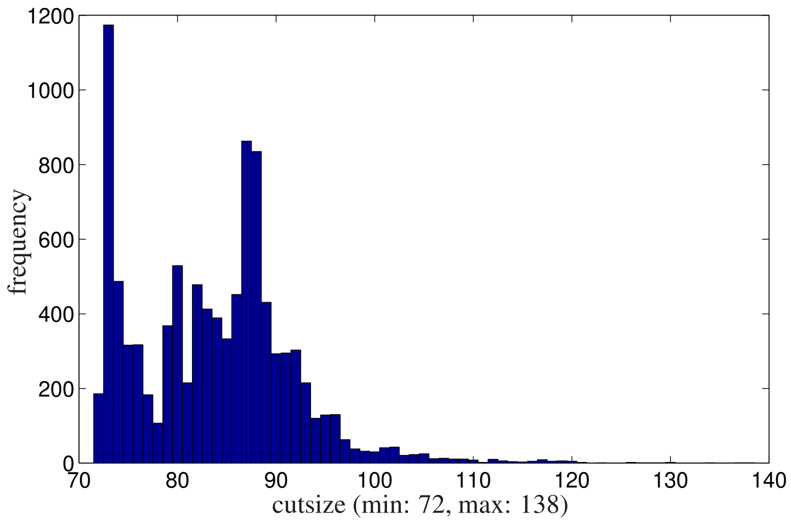

FIG. 3.4. *Chaco ML on graph No. 5*

the slightly simpler algorithm that Metis employs does not find these local minima. This supposition is strenghtend by looking at the data for Kernighan-Lin applied to random starting partitions. While for most graphs the results for this algorithm look like a very noisy, distorted bell-curve, sometimes with an additional peak near the beginning,the distribution for graph No. 16 also shows these distinctive peaks.)

But generally, Kernighan-Lin with a random starting partition shows two very different kinds of behavior. In all cases, the average cutsize is much worse compared to the multilevel algorithms but in most cases, its best results are just as good as the best cutsize delivered by the multilevel algorithms. In two cases though, (graphs Nos. 6 nd 31) the minimum of all 10000 tries are much worse (76% and 207%, resp.). For all these results, there were no discernable characteristics that indicated which graph would fall into which category.

In summary, there are several conclusions we can draw:

From a practical point of view, if one needs a "good" partition of a given graph then one should run one's chosen algorithm several times with a different starting value for the random number generator.

And to compare different graph-partitioning heuristics, instead of just comparing "one shot" results, one should regard the quality of the generated partition as a random variable dependent on the random-number seed. Depending on the application, one then needs to consider the mean deviation as well as the expected value. To be fair, one needs to take execution time into account as well since, given a fixed time-frame, a faster algorithm can be run more often, leading to a smaller mean deviation.

REFERENCES

[1] S. T. Barnard, A. Pothen, and H. Simon, *A spectral algorithm for envelope reduction of sparse matrices*, Numer. Linear Algebra Appl., 2 (1995), pp. 317–334.

[2] M. W. Berry, B. Hendrickson, and P. Raghavan, *Sparse matrix reordering schemes for browsing hypertext*, in The Mathematics of Numerical Analysis, J. Renegar, M. Shub, and S. Smale, eds., vol. 32 of Lectures in Applied Mathematics, AMS, 1996, pp. 99–123.

[3] T. N. Bui and C. Jones, *Finding good approximate vertex and edge partitions is NP-hard*, Inform. Process. Lett., 42 (1992), pp. 153–159.

[4] T. F. Chan, J. R. Gilbert, and S.-H. Teng, *Geometric spectral partitioning*, Tech. Rep. CSL-94-15, Parc Xerox, Jan. 1995.

[5] R. Diekmann, R. Lüling, B. Monien, and C. Spräner, *Combining helpful sets and parallel simulated annealing for the graph-partitioning problem*, Parallel Algorithms Appl., 8 (1996), pp. 61–84.

[6] U. Elsner, *Static and Dynamic Graph Partitioning*, PhD thesis, TU Chemnitz, July 2001. Logos Verlag, Berlin.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP–Completeness*, Freeman, 1979.

[8] J. R. Gilbert, G. L. Miller, and S.-H. Teng, *Geometric mesh partitioning: Implementation and experiments*, SIAM J. Sci. Comput., 19 (1998), pp. 2091–2110.

[9] D. R. Guillaume and F. Murtagh, *An application of xml and xlink using a graph-partitioning method and a density map for information retrieval and knowledge discovery*, in Proc Astronomical Data Analysis Software and Systems VIII, D. M. Mehringer, R. L. Plante, and D. A. Roberts, eds., vol. 172 of ASP Conference Series, 1999.

[10] A. Gupta, *Fast and effective algorithms for graph partitioning and sparse matrix ordering*, IBM Journal of Research and Development, 41 (1997), pp. 171–184.

[11] B. Hendrickson and R. Leland, *The Chaco User's Guide Version 2*, Sandia National Laboratories, Albuquerque NM, 1995.

[12] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, *Multilevel hypergraph partitioning: Applications in VLSI domain*, tech. rep., University of Minnesota, Department of Computer Science, 1997. short version in 34th Design Automation Conference.

[13] G. Karypis and V. Kumar, *Parallel multilevel k-way partitioning scheme for irregular graphs*, Tech. Rep. TR 95-036, University of Minnesota, Department of Computer Science, 1996.

[14] ———, *MeTis: Unstrctured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0*, Sept. 1998.

[15] B. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical Journal, 29 (1970), pp. 291–307.

[16] D. Knuth, *Seminumerical Algorithms*, vol. 2 of The Art of Computer Programming, Addison-Wesley, Reading, MA, USA, second ed., 1981.

[17] A. Pothen, *Graph partitioning algorithms with applications to scientific computing*, in Parallel Numerical Algorithms, D. E. Keyes, A. H. Sameh, and V. Venkatakrishnan, eds., Kluwer Academic Press, 1995.

[18] A. Pothen, E. Rothberg, H. Simon, and L. Wang, *Parallel sparse Cholesky factorization with spectral nested bisection ordering*, Tech. Rep. RNR-94-011, Numerical Aerospace Simulation Facility, NASA Ames Research Center, May 1994.

[19] A. Pothen, H. Simon, and L. Wang, *Spectral nested dissection*, Tech. Rep. RNR-92-003, NASA Ames Research Center, 1992.

[20] E. Rothberg and B. Hendrickson, *Sparse matrix ordering methods for interior point linear programming*, Tech. Rep. SAND96-0475J, Sandia National Laboratories, Jan. 1996.

[21] H. SIMON AND S.-H. TENG, *How good is recursive bisection*, SIAM J. Sci. Comput., 18 (1997), pp. 1436–1445.

[22] C. WALSHAW, M. CROSS, M. G. EVERETT, AND S. JOHNSON, *JOSTLE: Partitioning of unstructured meshes for massively parallel machines*, in Parallel Computational Fluid Dynamics: New Algorithms and Applications, N. Satofuka et al., eds., Amsterdam, 1994, Elsevier, pp. 273–280.