Partitioning of Unstructured Meshes for Load Balancing *

Olivier C. Martin
Division de Physique Théorique†, Institut de Physique Nucléaire,
Orsay CEDEX 91406 France
martino@ipno.in2p3.fr
and
Steve W. Otto
Department of Computer Science and Engineering
Oregon Graduate Institute of Science & Technology
20000 NW Walker Rd, PO Box 91000
Portland, Oregon, USA 97291-1000
otto@cse.ogi.edu

Abstract

Many large-scale engineering and scientific calculations involve repeated updating of variables on an unstructured mesh. To do these types of computations on distributed memory parallel computers, it is necessary to partition the mesh among the processors so that the load balance is maximized and inter-processor communication time is minimized. This can be approximated by the problem of partitioning a graph so as to obtain a minimum cut, a well-studied combinatorial optimization problem. Graph partitioning is NP complete, so for real world applications, one resorts to heuristics, i.e., algorithms that give good but not necessarily optimum solutions. These algorithms include recursive spectral bisection, local search methods such as Kernighan-Lin, and more general purpose methods such as simulated annealing. We show that a general procedure enables us to combine simulating annealing with Kernighan-Lin. The resulting algorithm is both very fast and extremely effective.

1 Introduction

Consider an unoriented graph G=(V,E), i.e., a collection of vertices V_i, $i = 1, ...N$, and edges $E_{i,j}$ ($E_{i,j}$ joins vertices V_i and V_j). The graph partitioning problem (GPP) consists of finding a partition of V into k subsets of specified sizes so that the number of "cut" edges is minimized. An edge $E_{i,j}$ is cut if V_i and V_j belong to different subsets. The GPP has many practical applications. It occurs in program text segmentation [1], and is a major ingredient in the problem of cell placement for VLSI [2, 3]. The application of interest for this paper is the partitioning of unstructured meshes used in scientific and engineering problems. The computations performed on

*Appeared in *Concurrency: Practice and Experience* vol 7(4) pp303-314 (1995)
†Unité de Recherche des Universités Paris XI et Paris VI associée au C.N.R.S.

these meshes demand vast amounts of computer power, so that an efficient implementation using parallel computation is a great advantage. Parallel implementations on distributed-memory computers require the distribution of the mesh amongst the processors. This leads to a graph partitioning problem where G=(V,E) is the graph associated with a differential operator defined on the mesh [4, 5, 6, 7, 8, 9].

Model the parallel computation as consisting of updates to variables located at the vertices of G, with data dependences between the variables given by the edges, E, of G. We have in mind an iterative solver of a PDE, such as simple relaxation or conjugate gradient, for which the dominant computational cost has the structure as described. Mapping the computation onto a distributed-memory parallel computer leads to a k-way graph partitioning problem, where k is the number of processors. Load balancing is achieved by appropriately specifying the sizes of each of the k subsets. If the processors are of equal speed, we simply choose equal size subsets. Communication overhead is made small by minimizing the number of cut edges. This is a simplified version of the mapping problem. Strictly speaking, one should minimize the maximum (over the k processors) of the combined communication and computation times. However, in practice, one uses the GPP to represent the mapping problem because it is conceptually and computationally more tractable, and because it is thought to give adequate solutions in most cases of interest.

In what follows, we quickly summarize a number of solution methods for the GPP, and stress particularly the the Kernighan-Lin local search heuristic [10, 11]. After this, we explain our method of combining local search methods, such as Kernighan-Lin, with simulated annealing. This methodology, which we call chained local optimization (C-L-O), is a very general one. It can be applied to many optimization problems and is quite effective. The paper goes on to compare C-L-O against other effective heuristics [12, 8], for both synthetically generated graphs and for graphs from real-world unstructured meshes. Finally, we describe the implementation of the C-L-O algorithm on a parallel network of workstations running PVM [13, 14].

2 Graph Partitioning Heuristics

Since the GPP is NP-complete, it comes as no surprise that exact methods are slow. An integer linear programming formulation of the GPP has recently been given by Barahona and Casari [15]. Since real applications have very large meshes, in practice it is necessary to take a heuristic approach. Two important, general-purpose heuristics are simulated annealing [16], and a variable depth, local search originally due to Kernighan and Lin [10, 11], which we will call Kernighan-Lin (K-L). Methods specific to the mapping of unstructured meshes include recursive coordinate bisection [17], compaction or contraction methods [18], and recursive spectral bisection [6, 8]. Leland and Hendrickson [9], and also Williams [19], compare these methods, and Mansour, Savage, and Wloka give parallel implementations [20, 21].

For the partitioning of generic (random) graphs, the "best" heuristics are simulated annealing and K-L. However, for unstructured meshes, K-L is substantially better than simulating annealing, and is also much faster [12]. Nevertheless, it is necessary to enhance K-L for it to be competitive with special purpose methods such as recursive spectral bisection. K-L is used within our algorithm, so for completeness, we now give a description of K-L and of some enhancements for unstructured meshes.

The Kernighan-Lin Local Search

It is easiest to describe K-L for $k = 2$ and equal sized partitions, so we restrict the explanation to that case. To deal with k-way partitions, one successively applies the algorithm described below to each pair of subsets chosen among the k subsets, until no improvement is found. The method can also be readily extended to the case of unequal sized partitions [10].

Let A be a subset of V, of size $N/2$, and B its complement. Define a 1-exchange to be an exchange of one element of A with an element of B. Suppose one repeatedly applies 1-exchanges that decrease the cut size until no more such 1-exchanges can be found. The configuration is then termed to be 1-optimal, or 1-opt for short. An iterative procedure that strictly reduces the cut size at each step is an example of a local search method.

It turns out that 1-opt is a mediocre algorithm, and that going to higher n-opt (i.e., looking at all possible n-exchanges) is very costly and does not lead to much improvement. Kernighan-Lin (K-L) is a variable, n-exchange algorithm that is much more effective than either 1-opt or 2-opt while being quite fast. "Variable" n means that some n-exchanges for n large are done, but not necessarily all of them. K-L is essentially a greedy, tabu, 1-exchange sweep through all the members of sets A and B: at each step, one exchanges the most favorable (or least unfavorable) pair of elements. During the sweep, if one element has already been exchanged, it can no longer be considered (it is "tabu") for further exchange during that sweep. Throughout the sweep, one monitors how the cut size changes. If the cut size does not decrease anywhere in the sweep, the partition is defined to be K-L-optimal. If it does decrease, one takes the partition with the lowest cut found during the sweep and uses that as the starting point for another sweep. The cut size is a decreasing function of sweep number, and in general one reaches a locally optimal partition in just a few sweeps.

For sparse graphs, K-L is fast, requiring $O(Nln(N))$ operations per sweep. As shown by Johnson et al., it is much faster than simulated annealing, and also gives smaller cut sizes [12]. However, K-L gives erratic results from run to run. In particular, for unstructured meshes, it is beaten by the recursive spectral bisection and coordinate bisection methods. Thus, for such graphs, it is necessary to run K-L many times from different random starts or to find ways to enhance K-L.

Enhancements to Kernighan-Lin for Unstructured Meshes

There are two commonly used approaches for improving K-L. The first, called compaction or contraction [18], consists of contracting the graph by merging nearby vertices, partitioning the smaller graph via K-L, undoing the merging procedure, and reapplying K-L. This approach, if used on multiple levels in a hierarchical manner, is well suited to unstructured meshes. The second approach consists of using something besides a random starting partition for the K-L. A simple, yet effective, starting partition can be obtained by coordinate bisection [17]. Since the coordinate-bisection of two-dimensional meshes uses a dividing line with a random orientation, the algorithm is named L-K-L for "Line K-L." L-K-L gives as good results as a hierarchical compaction approach but is simpler and is more effective than simulated annealing or K-L from random starts [12]. In view of this, we restrict ourselves to presenting comparisons of our algorithm, C-L-O, to L-K-L only.

3 Chained Local Optimization

Martin, Otto, and Felten [22] introduced a new meta-heuristic for optimization by combining local search methods with simulated annealing. The important realization is that simulated annealing needlessly explores all configurations. For most optimization problems, there are local search methods that quickly give good approximate solutions. By a simple generalization, we force simulated annealing to sample only locally optimal configurations. The resulting algorithm is termed "Chained Local Optimization" (C-L-O). It is a general purpose algorithm that improves upon both simulated annealing and local search methods (it necessarily beats local search, since it incorporates local search in the inner-most loop of the algorithm). We did [22, 23] an in depth study of C-L-O for the traveling salesperson problem, and found that it surpassed by a wide margin Lin-Kernighan, the best heuristic for that combinatorial optimization problem since 1973. More generally, as discussed by Martin and Otto [24], C-L-O should perform well on a wide class of problems which includes the GPP. For the purpose of this paper, important features of C-L-O include the following.

- It is general purpose, so it can be applied to general graphs. On the contrary, the compaction and L-K-L methods only work well on graphs with spatial structure.
- It out-performs L-K-L.
- It out-performs mesh-mapping-specific methods.
- The method incorporates the good aspects of both simulated annealing and K-L.

C-L-O for the GPP proceeds as follows. Suppose the partition is currently locally optimal (e.g., K-L-opt). This is labeled *Start* in Fig 1. Now apply a "kick" (an n-exchange with n not too small) to this partition so as to significantly change the character of *Start*. After the kick, we reach the configuration labeled *Intermediate* in the figure. Standard simulated annealing would impose the accept / reject procedure to *Intermediate*. Instead, we notice that it is much better to first improve *Intermediate* by a local search and apply the accept / reject test only afterwards. The local search takes us from *Intermediate* to the partition labeled *Trial* in Fig 1. Now apply the accept / reject test. If *Trial* is accepted, we have managed to find an interesting large change to *Start*. If *Trial* is rejected, we return to *Start*. The iteration, or chaining, of this process is the C-L-O method. Since the partition often changes dramatically in going from *Start* to *Trial*, the method behaves as a simulated annealing algorithm with very large steps from one configuration to the next.

C-L-O is much more effective than simulated annealing — as we've emphasized, the accept / reject step is only applied *after* the partition is returned to a local minimum. Many of the barriers (the "ridges") of the cost landscape are jumped over in one step by the C-L-O algorithm. Effectively, these barriers are smoothed or eliminated from the landscape. Simulated annealing, by contrast, must climb over each of these ridges in a series of steps, passing the accept / reject test many times, so that trapping is much more likely. Though C-L-O has the character of simulated annealing, for example one has a parameter that plays a similar role to the "temperature" of simulated annealing, C-L-O is outside the class of simulated annealing algorithms. A symmetry property known as detailed balance is violated by C-L-O and this means that it does not correspond to the true "annealing" of some "physical" system [22].

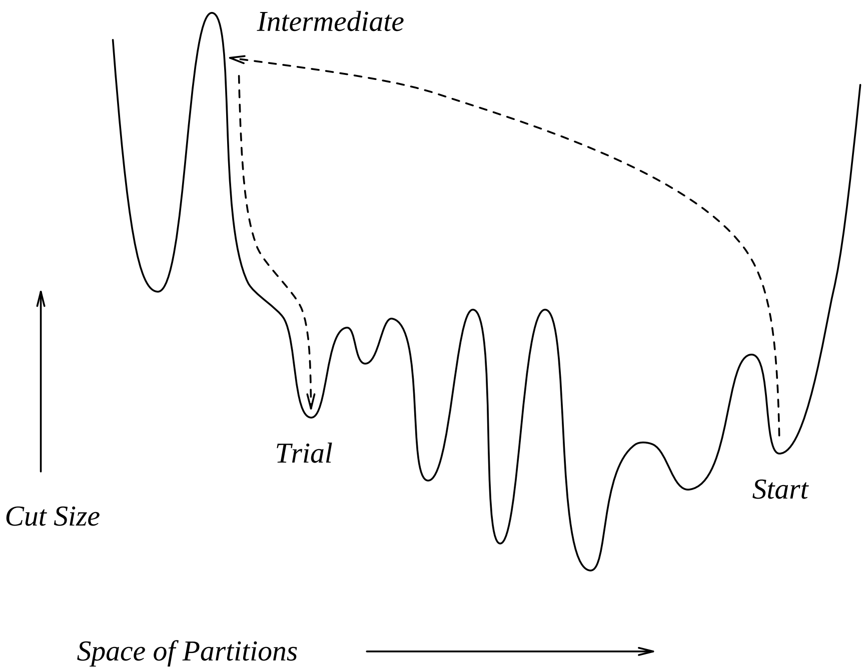

Figure 1: Schematic representation of the objective function and of the partition modification procedure used in chained local optimization.

To implement C-L-O for an arbitrary combinatorial optimization problem, one requires two things: a good local search heuristic, and a choice for the kick adapted to the optimization problem. In the case of the GPP, the first requirement is met by the Kernighan-Lin local search. To find an appropriate kick, notice that K-L generates partitions with many "islands", i.e., the subsets A and B usually end up being highly fragmented. It is this bad behavior that renders K-L uncompetitive against mesh-mapping-specific methods for mesh graphs. The fragmentation suggests a kick which exchanges vertices between islands and motivates the following procedure for generating a kick. First, in each subset A and B, randomly choose a vertex that belongs to a cut edge. These two vertices will be seeds. Let X and Y be the set of vertices in A and B that will be exchanged by the kick. X and Y are generated by growing a cluster around each seed: one adds to each cluster vertices that belong to the "other" subset but that are connected to the current cluster. The size of X and Y is chosen randomly ahead of time, but if one cluster can no longer grow (as happens when the seed is inside an island), then the cluster growth is stopped and one takes that as the kick. As shown in the next sections, the overall procedure gives rise to dramatically better partitions for unstructured meshes, but it also works extremely well for more general graphs.

4 Performance on "Geometric" Graphs

A good graph partitioning algorithm for one type of graph may not be good for another. In particular, the compaction and line algorithms discussed at the end of section 2 are good only for a special class of graphs. This makes it clear that the choice of algorithm should be motivated by the application. The graphs obtained from mesh-mapping problems are generally sparse and have a built-in spatial structure. In Section 5, we shall consider graphs associated with unstructured meshes, but we also wish to benchmark our algorithm on a more homogeneous

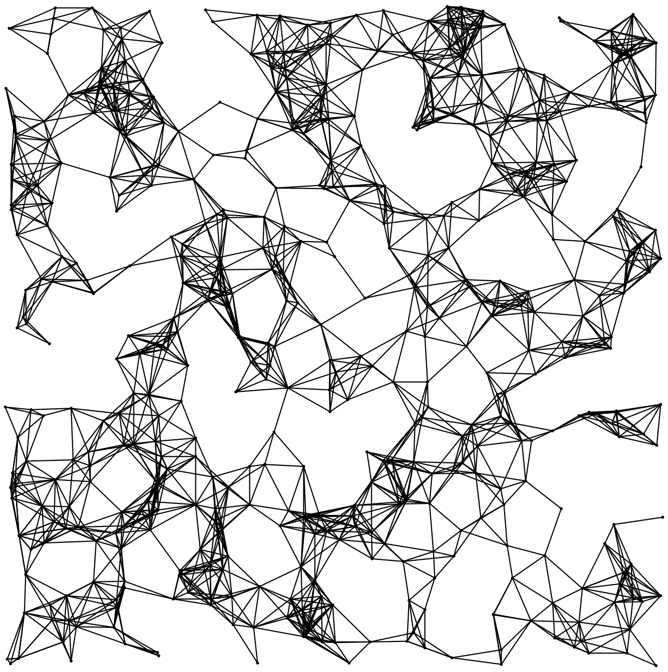

Figure 2: A geometric graph with $N = 500$ vertices and $d = 10$.

ensemble of graphs that can be generated randomly. Choose the graphs of this ensemble to be sparse and have spatial structure. For these graphs, hereafter called geometric graphs, the vertices are laid at random inside the unit square; two vertices are connected if and only if they are at a distance less than R (see figure 2). As R increases, the connectivity as measured by d, the average degree of a vertex, increases. Neglecting edge effects, one has, on average,

$$d = \pi R^2 N \ . \tag{1}$$

Johnson et al. did a thorough comparison of several algorithms and concluded that for such geometric graphs, K-L from random starts was better than simulated annealing, but that L-K-L was better still [12].

We first qualitatively compare the performance of C-L-O with K-L from random starts. Figure 3 contains the results of a run on a geometric graph of $N = 1000$ and average degree $d = 6$. The histogram gives the distribution of cut sizes encountered for 1000 K-L's from random start and those for one run of C-L-O for 1020 steps, the first 20 being omitted from the histogram. The C-L-O algorithm was run with a temperature of 2.0. Clearly, the C-L-O algorithm is exploring far better solutions than K-L from random start.

The reason for the poor results of K-L can be understood by looking at typical partitions: they are almost always fragmented as mentioned in section 3. Better results would be obtained by simply partitioning the vertices according to their coordinates, i.e., by using coordinate bisection [17]. For geometric graphs, this bisection can be obtained by choosing a random direction in space and partitioning the graph by a line parallel to this direction; this corresponds to the line algorithm discussed in section 2. Clearly that procedure gives rise to cut sizes that scale as $\sqrt{N}$ for geometric graphs. Its performance can be calculated analytically: for instance, for a vertical

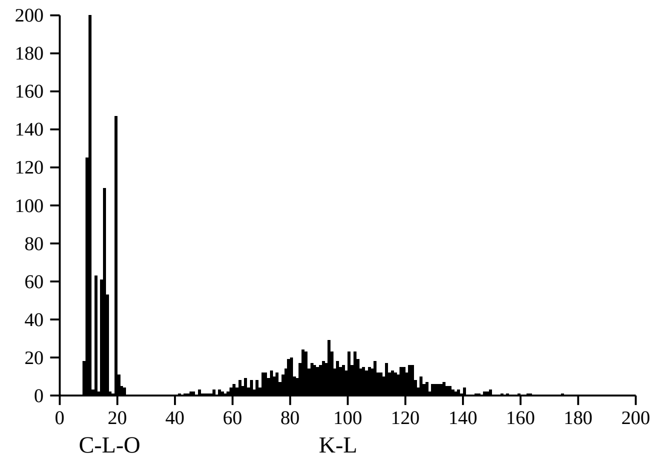

Figure 3: Histogram of solution values for a geometric graph with $d = 6$, $N = 1000$. The x-axis corresponds to cut size and the y-axis corresponds to the number of times each cut size was found. The data is for 1000 K-Ls from random starts, and for 1000 steps of a single C-L-O run.i The highest bin of the C-L-O is off-scale by a factor of two.

or horizontal line, the line algorithm gives

$$\langle \text{cut size} \rangle = \frac{2}{3}\sqrt{N}(\frac{d}{\pi})^{3/2}. \tag{2}$$

On the contrary, because K-L from random starts leads to fragmented partitions, it gives cut sizes that scale as N. It might be argued that by running K-L many times one could get much better results. However, since geometric graphs are extended in space, the central limit theorem suggests that the distribution of K-L cut sizes will tend towards a Gaussian of width $\sqrt{N}$ centered on its mean (proportional to N). Thus as $N \to \infty$, it becomes hopeless to use K-L to get cut sizes on the order of $\sqrt{N}$. More generally, the argument can be used to show that as $N \to \infty$, the performance of an algorithm is characterized by the *average* cut size it leads to; it is thus more efficient to improve an algorithm than to simply use it in multiple restarts as N becomes large.

We now report on the performance of two algorithms, L-K-L (the Kernighan-Lin algorithm with a line start), and C-L-O. For C-L-O, it is necessary to specify what to do with the temperature. For simplicity, we consider runs where T has been set to 0 (zero temperature quenches); we have also done runs where the temperature was fixed or followed an exponential annealing schedule, but the results were not significantly different from the $T = 0$ runs. We have chosen $T = 0$ because it has the advantage of corresponding to a parameter-free schedule.

What is the dependence of the min cut size on both N and d? For "small" values of d, the min cut is generally very small and nearly N independent, because the connectivity is very low. Indeed if d is sufficiently small, the graph becomes disconnected, and it is often possible to find a zero cut partition. As d increases up to about 5, min cut sizes are on the order of 1,2, or 3 as would be the case if one had a tree graph. Finally, as d increases further, the min cut crosses over to a $\sqrt{N}$ scaling law. The graphs obtained from unstructured mesh problems belong to this latter regime. We present results for $d = 6$ and for $d = 10$: we chose $d = 6$ because it is the average degree of two-dimensional unstructured meshes (c.f. the instances investigated in the next section), and $d = 10$ because Johnson et al. gave results for this case.

We begin with the case $d = 6$. Before using the $\sqrt{N}$ scaling law to compare performance, we give explicit results as a function of N for illustrative purposes. Five instances of random, geometric graphs were generated for several choices of the number of vertices: $N = 100, 250, 500,$ and 1000. For each instance, we ran L-K-L 2000 times, and we ran C-L-O 20 times, each run consisting of 100 kick/K-L steps. From the 2000 L-K-L data points, we followed the method described in [12] to derive the distribution of the best cut found in 100 independent trials. The mean was then compared with the corresponding mean of the best found in each of the 20 C-L-O runs. Both algorithms (one run of 100 steps of C-L-O and 100 L-K-Ls) use about the same amount of CPU time. The results are presented in Table 1, along with the average (single run) L-K-L performances. We do not know the *exact* minimum, but for reference, we have also given the best cut ever found by the two algorithms. For $N = 100$, the best cut ever found was always obtained by each algorithm. $N = 100$ corresponds to "easy" problems, so we have omitted those data from the table. For $N = 250$, 2000 L-K-Ls was not enough to find the best ever for 3 of the 5 graphs, and for the larger values of N, L-K-L was never able to find the best ever. C-L-O, on the other hand, finds (for 100 steps) the best ever multiple times among the 20 runs for $N = 250, 500$ and 1000.

For large N, the performance of the algorithms can be characterized by the factor C in the formula $\langle \text{cut size} \rangle = C\sqrt{N}$. Using additional data for $d = 6$, we find the C corresponding to 100 L-K-Ls to be $C_{100-L-K-L} = 0.381$ and $C_{C-L-O} = 0.356$ for the $T = 0$ quenches. In practice, our quenches were run for 100 kicks, so the quoted result is higher than the value for infinitely long runs. As it stands, 100-L-K-L leads to cut sizes about 7% larger than C-L-O.

The same methodology was used to study graphs with $d = 10$. Again, C-L-O beats L-K-L for a given amount of computer time. We find $C_{100-L-K-L} = 1.54$, and $C_{C-L-O} = 1.50$. We observe that as the graphs become more dense, the advantage of C-L-O sets in at progressively larger values of N. This makes sense, because as d increases for geometric graphs, the optimum cut becomes straight and the L-K-L algorithm has an easier time finding it.

In summary, L-K-L is superior to K-L for geometric graphs and, in particular, gives the correct scaling in $\sqrt{N}$. However, it is surpassed by C-L-O for both sparse and dense geometric graphs, even though we have not fine-tuned the kick or the temperature in C-L-O. One should also keep in mind that the C-L-O approach is not limited to graphs with spatial structure, and indeed leads to good results for random graphs. The line initialization is not possible for such graphs, nor is compaction of much use.

5 Performance on unstructured meshes

Barnard and Simon [8] studied recursive spectral bisection on several unstructured meshes that arise in mesh-mapping problems. This section benchmarks L-K-L and C-L-O on some of these same problems, as well as a three-dimensional graph studied in [9]. The main differences with the ensemble of geometric graphs used in the previous section are that, except for the "Mpart" mesh, the graphs are planar and have an average degree close to 6. The lack of variance in the degree of vertices makes these problems easier to solve both for L-K-L and C-L-O.

The five meshes have the names: Spiral (1200), Parc (1240), Hammond (4720), Barth5

	Cut Size for Five, $N = 250$ Graphs				
Algorithm					
L-K-L	11.6	13.7	9.6	10.8	13.8
100-L-K-L	4.4	6.0	3.2	4.0	9.5
C-L-O	4.0	6.0	2.3	4.0	8.9
Best Found	4	5	2	4	7
	Cut Size for Five, $N = 500$ Graphs				
Algorithm					
L-K-L	21.0	16.4	15.4	22.2	18.4
100-L-K-L	12.0	7.0	5.1	10.2	10.6
C-L-O	11.8	4.8	5.0	10.0	7.1
Best Found	9	3	4	8	5
	Cut Size for Five, $N = 1000$ Graphs				
Algorithm					
L-K-L	26.8	25.1	29.5	23.8	26.6
100-L-K-L	13.1	10.0	14.2	10.2	13.7
C-L-O	12.4	7.6	14.6	7.4	13.2
Best Found	8	5	11	5	10

Table 1: Average performance on 15 random geometric graphs of $d = 6$. There are five graphs each for $N = 250, 500$, and 1000. For algorithm L-K-L, the value in the table shows the average over 2,000 runs of L-K-L from random starts. For algorithm 100-L-K-L, L-K-L was run 100 times from random starts and the best value was taken. The result shown in the table is the average value of that best, when this procedure is done many times. For algorithm C-L-O, the value in the table is the average over 20 runs of C-L-O, each of length 100 steps. The temperature of the C-L-O runs was set to zero. The values under Best Found are the min over all the previous procedures — this was always found among the 20 runs of C-L-O.

(15606), and Mpart (6673), where the number of vertices of each mesh is indicated in parentheses. The first four are two-dimensional meshes, the last a three-dimensional mesh. In comparing various algorithms, we need not consider simulated annealing since it has been shown that K-L performs better than S-A on such sparse graphs [12]. We consider the five graphs in turn.

Spiral has the geometry of a spiral, so the use of the line algorithm (i.e., coordinate bisection) leads to a fragmented partition. One thus might expect L-K-L to perform poorly, but the fact is that the number of vertices is sufficiently small for K-L (and thus L-K-L) to give good results. The average cut-size for L-K-L is given by $\langle \text{cut size} \rangle_{L-K-L} = 15.4$, and it finds the best ever (of cut size 9) 47% of the time. The repeated use of L-K-L improves this result of course, so that with 100 trials, one is virtually certain to hit the best ever: $\langle \text{cut size} \rangle_{100-L-K-L} = 9.0$. For C-L-O, we did 20 runs of 100 steps as in section 4, and found also $\langle \text{cut size} \rangle_{C-L-O} = 9.0$. (The best ever is found in all the C-L-O runs.) These results lead us to believe that the best ever is the optimum.

The second mesh, Parc, is a triangulation of variable density. Again the two heuristics solve this instance rather easily. We find $\langle \text{cut size} \rangle_{L-K-L} = 34.1$, $\langle \text{cut size} \rangle_{100-L-K-L} = 21.1$, and $\langle \text{cut size} \rangle_{C-L-O} = 21.0$ with the same run parameters as before. L-K-L hits the best ever cut (21) with probability 0.036, and again C-L-O finds this best ever in all 20 runs.

The next two meshes correspond to airfoil problems. The smallest problem, Hammond, is still rather easy to solve. All C-L-O runs found the best cut ever, of size 90. L-K-L found this cut with probability 0.28, and give an *average* cut size of 116.4. With 100 L-K-L's, one is virtually certain to find the best cut. For this mesh, Barnard and Simon report a cut size of 106 using spectral bisection.

For Barth5, we find $\langle \text{cut size} \rangle_{L-K-L} = 197.1$, $\langle \text{cut size} \rangle_{100-L-K-L} = 139.2$, and $\langle \text{cut size} \rangle_{C-L-O} = 139.25$ for runs of 100 steps. The best cut ever was 139: L-K-L found it with probability 0.016, and C-L-O found it in 15 of the 20 runs. For this instance, spectral bisection gave a cut size of 176 [8].

For Mpart [9], the best C-L-O runs found a partition with cut size 731. For comparison, Chaco [25] gets partitions of size 729 ("Multilevel-KL") and 724 ("Spectral-KL") on this graph (private communication).

It is fair to conclude that these types of meshes are solved rather easily, either by L-K-L using multiple tries, or by C-L-O. Not surprisingly, the larger meshes become more difficult. Note that both algorithms incorporate K-L, a general-purpose graph partitioning method. If we compare them to the mapping problem-specific methods (coordinate or spectral bisection), it is clear that those methods are not as good. The key is to use K-L as a post-processor. Then, almost any method will become competitive. We illustrated this for coordinate bisection when transformed into L-K-L. Similarly, Barnard and Simon, as well as Leland and Hendrickson, have reported improvements to spectral bisection when K-L is used as a post-processor [8, 9]. However, such methods are limited by their deterministic behavior, and could almost certainly benefit from some randomization.

6 Parallel C-L-O

Most local search methods for the GPP do not parallelize well, mainly because the constraint of maintaining a feasible solution is not readily implemented in parallel. Thus, we only consider implementations where a given processor has a complete configuration in local memory. We work in the framework of a distributed-memory architecture and have implemented the codes on a network of workstations under PVM [13, 14].

The simplest way to parallelize chained local optimization is to have each processor run independent C-L-O chains. This is equivalent to running multiple random starts on a single processor. If we have P processors, at any given time we have a population of at least P configurations. However, independent runs are not best because one should be able to use the mutual information available in the current population. Thus, we have implemented branching and pruning among the configurations on the different processors. This is called Darwinian selection for genetic algorithms and diffusion Monte Carlo in physics. In a branching step, the best configurations are duplicated in the population while in pruning, the worst ones are eliminated. Branching and pruning events occur relatively rarely (as measured in cpu time) so very little time is spent on communication, leading to an efficient parallel algorithm. In our implementation, we run for a certain time interval, find the configuration with the best cut, and then apply a winner-take-all selection strategy. Note that two processors may contain copies of the same configuration, but they go through distinct random number sequences and so they perform independent searches. The GPP code runs both on uniprocessor systems and on heterogeneous computing environments using PVM. All machines are used to near maximum capacity, and parallel speed-up (at least for tens of workstations) is near-linear. This is simply because communications are done rarely.

7 Discussion and conclusion

Many algorithms have been proposed for partitioning unstructured meshes for the mapping problem. The standard, general-purpose algorithms, simulated annealing and local search (including K-L), do rather poorly. This has stimulated the development of special purpose methods such as coordinate or spectral bisection. In this paper, we showed how a general-purpose approach to combinatorial optimization could be successfully adapted to graph partitioning. As shown in sections 4 and 5, the C-L-O algorithm beats special-purpose methods. It also beats the hybrid method, L-K-L, on geometric graphs, while being as good on unstructured meshes. This performance was achieved without any parameter fine-tuning (the temperature was set to zero, and no effort was made to improve the kick). Finally, C-L-O has several appealing properties: (i) it works for general graphs; (ii) it is easily generalized to k-way partitions without resorting to recursive bisection; (iii) and, it can handle unequal partition sizes (important for mapping onto heterogeneous processors). We are currently studying the performance of C-L-O for full k-way graph partitioning and these results will appear in a future paper.

8 Acknowledgements

We thank Robert Leland, Bruce Hendrickson, and Horst Simon for useful discussions. This work was supported in part by NATO travel grant CRG 920831.

References

[1] B.W. Kernighan. *Some Graph Partitioning Problems Related to Program Segmentation.* PhD thesis, Princeton University, Princeton, NJ, 1969.

[2] K. Shahookar and P. Mazumder. VLSI cell placement techniques. *ACM Computing Surveys*, 23(2):143–220, June 1991.

[3] M. Hanan and J. M. Kertzberg. A review of the placement and quadratic assignment problems. *SIAM Review*, 14(2):324, 1972.

[4] C. Farhat. On the mapping of massively parallel processors onto finite element graphs. *Computers and Structures*, 32(2):347–353, 1989.

[5] J. Flower, S. Otto, and M. Salama. A preprocessor for finite element problems. In *Symposium on Parallel Computations and their Impact on Mechanics.* American Society of Mechanical Engineers, 1987. ASME Winter Meeting, Dec. 14-16, 1987, Boston, Mass.

[6] A. Pothen, H. Simon, and K. P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *"SIAM J. Math. Anal. Appl.*, 11(3):430–452, 1990.

[7] V. Venkatakrishnan, H. Simon, and T. Barth. A MIMD implementation of a parallel Euler solver for unstructured grids. *J. Supercomputing*, 6(2):117–127, 1992.

[8] S. Barnard and H. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, April 1994.

[9] R. Leland and B. Hendrickson. An empirical study of static load balancing algorithms. In *Scalable High Performance Computing Conference*, pages 682–685. IEEE Computer Society Press, 1994.

[10] B. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49:291, 1970.

[11] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Workshop*, page 175, 1982.

[12] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part I (graph partitioning). *"Operations Res.*, 37:865–892, 1989.

[13] A. L. Beguelin, J. J. Dongarra, A. Geist, R. J. Manchek, and V. S. Sunderam. Heterogeneous network computing. In *Sixth SIAM Conference on Parallel Processing*, 1993.

[14] Jack Dongarra, Al Geist, Robert Manchek, and Vaidy Sunderam. Integrated PVM framework supports heterogeneous network computing. *Computers in Physics*, April 1993.

[15] F. Barahona and A. Casari. On the magnetisation of the ground states in two dimensional Ising spin glasses. *Comput. Phys. Commun.*, 49:417–421, 1988.

[16] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[17] M. Berger and S. Bokhari. A partitioning strategy for non-uniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36(5):570, 1987.

[18] T. Bui, C. Heigham, C. Jones, and T. Leighton. Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms. In *26th ACM/IEEE Design Automation Conference*, page 775, 1989.

[19] R. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and Experience*, 3(5):457–481, October 1991.

[20] N. Mansour. *Physical Optimization Algorithms for Mapping Data to Distributed-Memory Multiprocessors*. PhD thesis, Syracuse University, 1992.

[21] J. E. Savage and M. G. Wloka. On parallelizing graph-partitioning heuristics. In *Proceedings of the ICALP'90*, page 476, 1990.

[22] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, June 1991.

[23] O. Martin, S. W. Otto, and E. W. Felten. Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 11:219–224, 1992.

[24] O. C. Martin and S. W. Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[25] B. Hendrickson and R. Leland. The Chaco user's guide, version 1.0. Technical Report SAND 93-2339, Sandia National Laboratories, Albuquerque, NM, October 1993.