A Generational Scheme for Partitioning Graphs

Alan Soper
Computing & Mathematical Sciences,
University of Greenwich,
Old Royal Naval College, Greenwich,
London, SE10 9LS, UK.
A.J.Soper@gre.ac.uk

Chris Walshaw
Computing & Mathematical Sciences,
University of Greenwich,
Old Royal Naval College, Greenwich,
London, SE10 9LS, UK.
C.Walshaw@gre.ac.uk

Abstract

Graph partitioning divides a graph into several pieces by cutting edges. Very effective heuristic partitioning algorithms have been developed which run in real-time, but it is unknown how good the partitions are since the problem is, in general, NP-complete. This paper reports an evolutionary search algorithm for finding benchmark partitions. Distinctive features are the transmission and modification of whole subdomains (the partitioned units) that act as genes, and the use of a multilevel heuristic algorithm to effect the crossover and mutations. Its effectiveness is demonstrated by improvements on previously established benchmarks.

1 INTRODUCTION

The graph partitioning problem can be stated as: partition the vertices of a graph into a given number of sets so that each set is of (approximately) equal size and so that the number of edges cut by the partition is minimised. The need for graph partitioning arises naturally in many applications such as distributing a finite element mesh across the nodes of a parallel computer in order to minimise communication overhead. It is well known that this problem is NP-complete (i.e. it is unlikely that an optimal solution can be found in polynomial time), so in recent years much attention has been focused on developing suitable heuristics, and a range of powerful methods have been devised, e.g. [8].

Here we report on a technique, combining an evolutionary search algorithm together with a multilevel graph partitioner, which has enabled us to find partitions considerably better than those that can be found by any of the public domain graph partitioning packages such as JOSTLE, METIS, etc. We do not claim this evolutionary technique as a possible substitute for the aforementioned packages; the very long run times preclude such a possibility for the typical applications in which they are used. However we do consider it of interest to find the best possible partitions for benchmarking purposes and for certain applications such as circuit partitioning, where the quality of the partition is paramount, the computational resources required may be completely justified by the very high quality partitions that the technique is able to find.

The main focus of this paper is to describe a strategy for combining evolutionary search techniques with a standard graph partitioning method. In Section 2 we outline the multilevel graph partitioning method used and establish notation & definitions. In Section 3 we then describe the genetic framework by defining the crossover and mutation operators and discuss how they are combined with the multilevel partitioner. Related work is also discussed here. We have conducted many experiments to test the technique and in Section 4 present some of the results including tests on unstructured meshes (§4.1). We also compare our results against a recent benchmark of Kang & Moon, [10]. Some of these graphs have similar structure to meshes, but some less structured examples are included.

The principal innovation described in this paper is the construction of crossover and mutation operators with an heuristic bias suitable for partitioning certain types of graphs which include meshes. These operators rely on the use of a multilevel graph partitioner, which is used to partition carefully chosen subgraphs of the original graph.

2 MULTILEVEL GRAPH PARTITIONING

Let $G = G(V, E)$ be an undirected graph of vertices V, with edges E. Given that the graph needs to be distributed to P processors, define a partition π to be a mapping of V into P disjoint subdomains S_p such that $\bigcup_P S_p = V$. The partition π induces a *subdomain graph* on G which we shall

refer to as $G_\pi = G_\pi(S, L)$; there is an edge or *link* (S_p, S_q) in L if there are vertices $v_1, v_2 \in V$ with $(v_1, v_2) \in E$ and $v_1 \in S_p$ and $v_2 \in S_q$. We denote the set of inter-subdomain or cut edges (i.e. edges cut by the partition) by E_c. Vertices which have an edge in E_c (i.e. those which are adjacent to vertices in another subdomain) are referred to as *border* vertices. Finally, note that we use the words subdomain and processor more or less interchangeably: the mesh is partitioned into P subdomains; each subdomain S_p is assigned to a processor p and each processor p owns a subdomain S_p.

In the context of partitioning a mesh for a parallel application, the definition of the graph partitioning problem is to find a partition which evenly balances the load or vertex weight in each subdomain whilst minimising the communications cost. To evenly balance the load, the optimal subdomain weight is given by $\overline{S} := \lceil |V|/P \rceil$[1] and the *imbalance* is then defined as the maximum subdomain weight divided by the optimal (since the computational speed of the underlying application is determined by the most heavily weighted processor). There is some discussion about the most appropriate metric for partitioning, e.g. [7], and indeed it is unlikely that any one metric is appropriate, however, it is common practice in graph partitioning to approximate the communications cost by $|E_c|$, the weight of cut edges or *cut-weight*. The usual (although not universal) definition of the graph partitioning problem is therefore to find π such that $|S_p| \leq \overline{S}$ and such that $|E_c|$ is (approximately) minimised.

In fact it has been noted for some time that partition quality can often be improved if a certain amount of imbalance is allowed, [15]. If we allow θ% imbalance then the partitioning problem becomes 'find a partition π such that $|S_p| \leq \overline{S} \times (100 + \theta)/100$ and that $|E_c|$ is (approximately) minimised'.

2.1 The multilevel paradigm

In recent years it has been recognised that an effective way of both speeding up graph partitioning techniques and/or, perhaps more importantly, giving them a global perspective is to use multilevel techniques. The idea is to match pairs of vertices to form *clusters*, use the clusters to define a new graph and recursively iterate this procedure until the graph size falls below some threshold. The coarsest graph is then partitioned (possibly with a crude algorithm) and the partition is successively optimised on all the graphs starting with the coarsest and ending with the original. This sequence of contraction followed by repeated expansion/optimisation loops is known as the multilevel paradigm and has been successfully developed as a strategy for overcoming the localised nature of the Kernighan-Lin (KL), [12], and other optimisation algorithms. The multilevel idea was first proposed by Barnard & Simon, [2], as a method of speeding up spectral bisection and improved by both Hendrickson & Leland, [8] and Bui & Jones, [4], who generalised it to encompass local refinement algorithms. Several algorithms for carrying out the matching of vertices have been devised by Karypis & Kumar, [11], while Walshaw & Cross describe a method for utilising imbalance in the coarsest graphs to enhance the final partition quality, [18].

[1] where the ceiling function $\lceil x \rceil$ returns the smallest integer greater than x

3 THE GENETIC ALGORITHM

Genetic algorithms produce new search points by one of two operations: crossover which combines information from two or more randomly selected individuals in the current generation, and mutation which modifies a single, randomly selected, individual. The construction of successful crossover and mutation operators is problem specific and often complex, especially where individuals are subject to constraints (as are the partitions) so that information from different individuals cannot be arbitrarily combined or modified. Further, the information needs to be effectively exploited so that new individuals result that are fitter than the current best individuals with sufficient probability even when the current generation is already very good, [1].

A number of genetic algorithms for graph partitioning (e.g. [10]) have been constructed using a 'linear' chromosomal representation consisting of a list of subdomain memberships of a graph's vertices, each list item representing the subdomain in which the vertex appears. Crossover combines information from two chromosomes using standard operations (one-point crossover etc) to produce a child chromosome. In this case the linkage is determined by distance apart in the list and given that the ordering of vertices is arbitrary for most graphs, so is the linkage. The linkage has been improved by defining orderings of the list items which place nearby vertices in the graph (separated by few edges) close together in the list, and by 'normalising' the chromosomes before mating by relabeling the subdomains in one parent so that it has more vertices with the same subdomain membership as when they appear in the second, [10].

Genetic algorithms using this representation usually apply a local optimisation procedure to the resulting offspring, which improves and repairs them so that they are again balanced partitions. A novel and more powerful such procedure, termed Cyclic Partitioning, has recently been used by Kang & Moon with a GA of this type. Their procedure provides a more comprehensive search for local improvements than previous Kernighan-Lin based schemes by investigating the possible improvements available by transferring

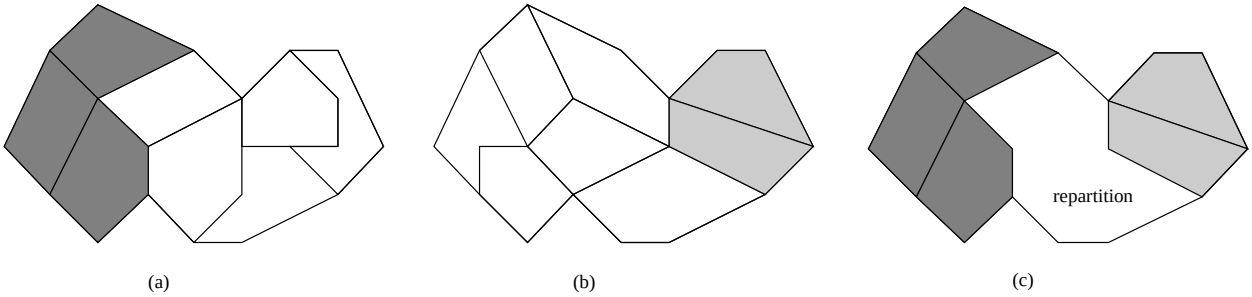


Figure 1: An illustration of the crossover operator

vertices across a set of partition boundaries (one vertex at each), such that the subdomains the vertices belong to form a cycle in the subdomain graph. They have run extensive tests comparing their genetic algorithm with recursive Kernighan Lin, pairwise Kernighan Lin and Cyclic Partitioning, using many repeated applications of the optimisation algorithm on different, initial, randomly-generated partitions. The authors have used the results to establish a set of high quality, benchmark partitions documented in [5]; 8-way & 32-way partitioning were performed.

Soper *et al.* have recently constructed a genetic algorithm which uses neither a linear chromosomal representation nor a traditional crossover operator, [16]. The crossover is implemented by modifying the graph to record where the parents had cut-edges by weighting them, and then applying a local optimisation procedure JOSTLE to the new graph so that cut edges of the parents are more likely candidates to be cut again due to the weighting. The mutation operator has an heuristic bias which exploits the local translational invariance possessed by many graphs of interest. This work produced benchmark partitions for evaluating public domain packages, and especially on graphs representing unstructured meshes. The current work is based on similar operators but further exploits the properties of the graphs being partitioned. The major difference is that the local optimisation procedure used during crossover and mutation needs only to be applied to a fraction – almost always less than half – of the graph to be partitioned. Much more information is transferred into the offspring from the parent(s) and the optimisation algorithm is more effectively focussed on one part of the problem at a time.

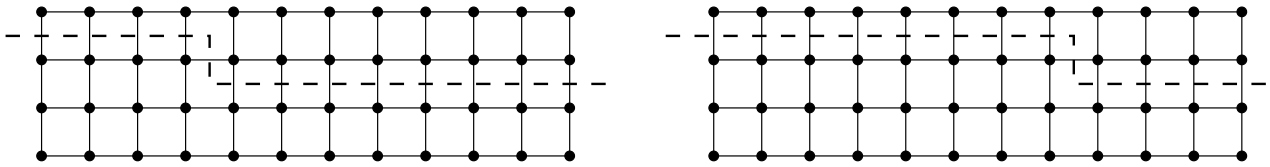
3.1 Recombining and mutating subdomains

Both crossover and mutations act on subdomains (or the set of of cut edges containing a subdomain). Crossover selects sets of complete subdomains from two individuals, and combines them in the child by partitioning the remainder of the graph as illustrated in Figure 1 ; Figures 1(a) & 1(b) show two parent partitions which have been selected for crossover. Sets of adjacent subdomains which do not intersect are selected (shown shaded) and the remainder of the graph – the unshaded part of Figure 1(c) – is repartitioned. Crossover seeks to exploit locality - the fact that graphs needing to be partitioned often only have vertices with low degree, showing local connectivity. This property holds for unstructured meshes which in their spatial embedding of physical origin only have short range connections, reflecting the locality of the physical systems they model. Locality allows subdomains from one individual to be successfully recombined with those from another when they are well separated.

Mutation takes a set of subdomains from an individual that constitute a cycle in the subdomain graph. The subgraph defined by this cycle is then repartitioned so as to exploit local translational symmetry; new partition boundaries are sought close to existing boundaries where they should have similar and so sometimes less cut edges. Another desirable property of mutations is that they are compatible or commute [14], i.e. their result does not depend on the order of their application. Our mutations will tend to have this property, either because their defining cycles don't intersect, or when they do because local translational symmetry, provides sufficient variations of common, partition boundaries to accommodate the balance constraint with a very similar number of cut edges.

In summary crossovers are constructed by producing cuts in the subdomain graphs of two individuals and mutations by constructing cycles in the subdomain graph of one individual. Figure 2 shows a case where a partially translated boundary has exactly the same number of cut edges.

Selection of subdomains for crossover: The number of subdomains selected from the first parent was chosen randomly and uniformly from the range $(P/4) - 1$ to $(P/2) - 1$, which choice prevented a parent from producing an offspring mostly identical to itself. The first subdomain was chosen randomly, then the second from its neighbours, the third from neighbours of both these subdomains, with probability proportional to the number of chosen neighbours (1 or 2), and so on. Thus there is a bias to choosing sets of subdomains with more internal or common partition bound-



(a) (b)

Figure 2: A translated boundary fragment with the same number of cut edges

aries. The choice of a more compact structure increases the chances of successful recombination; the extreme opposite a collection of scattered non-neighbouring subdomains would effectively prevent any substantial change in the parent.

Subdomains are selected from the second parent by a process of elimination. First delete subdomains from this parent which have any vertices in common with the subdomains selected from the first. Then delete those that have more than two fifths of their vertices in common with the neighbours of the subdomains selected from the first parent. The remaining subdomains are included in the offspring. The heuristic provides a balance between the competing demands of information transfer, i.e. copy more subdomains and their bordering cut edges into the offspring, and the need to allow successful recombination, i.e. is the remainder of the graph capable of being partitioned with a small enough number of cut edges?

In general, the crossover offspring partition will not be of sufficiently high quality to be accepted into the succeeding generation of the genetic algorithm, therefore it is immediately subject to a hill-climbing sequence of mutations.

In some cases during crossover, it is possible for the subdomains selected from the two parents to be such that the remaining subgraph cannot be partitioned within the imbalance constraint since it contains too many vertices. Such situations turn out to be rare however and the crossover is abandoned.

Selection of cycles of subdomains for mutations: Rather than selecting a cycle independently for each mutation, sets of mutations are carried out together in a hill-climbing sequence, the result of a mutation being the starting point of the next if it produces a better or equally good partition with respect to the number of cut edges. If the partition is worse, the mutation is ignored.

A random spanning tree of the subdomain graph is generated, and then the fundamental cycles with respect to this are recorded. Of these, cycles with lengths less than 4 and greater than 8 are discarded; small cycles because they allow little variation and larger cycles since it is more difficult for the partitioner to simultaneously improve more boundaries. When a cycle of subdomains is selected borders between subdomains are also targets for improvement, so that very small cycles tend to be included in the optimisation process already. Variations over longer cycles are provided by the joint effect of crossover and mutation - they will tend to be cut on crossover, and the resulting parts improved as part of other smaller cycles.

Thus a hill-climbing sequence is the set of mutations associated with the remaining fundamental cycles. These sequences are used since they are more efficient to implement than producing the mutations individually and their cycles will include most subdomain boundaries.

3.2 Partitioning Subgraphs

The implementation of the new partitions of subgraphs needed for both crossover and mutation are based on previous work, [16]. We use a multilevel technique as an efficient and effective partitioner. In fact the multilevel partitioner used is known as JOSTLE and we shall henceforth refer to it as such, although any graph partitioning heuristic which can deal with real (non-integer) edge weights could be used. JOSTLE is fully described in [18].

Both crossover and mutation require that some edges of the graph be made more likely to appear as cut edges under the action of JOSTLE. This is achieved by biasing the costs of the edges: the cost of an edge becomes unity plus a positive number and JOSTLE takes account of these additional costs when seeking low cost partitions. Mutations are implemented by making existing cut edges and their neighbours much less costly and crossover by making the cut edges of both parents occurring in the subgraph being repartitioned slightly less costly. New biases are explicitly and partially randomly constructed from the parent(s) for each operation.

3.3 The CHC adaptive search algorithm

The genetic algorithm framework chosen was Eshelman's CHC adaptive search algorithm, [6]. It has been shown to work successfully on a wide range of problems (e.g.

[13, 17]) with the same parameter settings and, importantly for partitioning large graphs, it uses a small population of 50 individuals. This allowed the simulations to run in a computer's memory. Its main features are: an elitist selection strategy, a highly explorative crossover operator, incest prevention and partial randomisations or restarts.

We adapted CHC as follows: Since the crossover provides less variation than that used in the original version of CHC, we also allow mutations. When a pair of parents are selected for mating and pass the incest test, they crossover with probability 0.3 and suffer mutation the rest of the time. When mutation only is applied, a separate offspring is not produced, rather, provided an improved partition or one of equal quality results (compared to the parent), it is overwritten. This procedure helps maintain diversity within the population.

When preventing incest, the distance between any two individual partitions was defined to be the number of vertices in the graph minus the number of edge vertices that they have in common. This measure clearly takes common cut edges into account, but also any nearby borders. The distance threshold is initialised, both when starting the genetic algorithm and on restarts, to the average distance apart of some randomly sampled pairs in the population. Clearly the distance between individuals is never zero, so that a distance threshold to initiate restarts has to be set. This is taken to be the distance of the best individual from itself, the expected distance apart of individuals in the population when it has converged. The distance threshold was decremented by 10 whenever no new offspring were accepted. This number need not be tuned to any great accuracy, since a small value will produce earlier subsequent decrements and vice-versa. However the value should be large enough to allow more parents to mate on average; a decrement of 10 allowed this.

At a restart the best individual is randomised by mutating the whole partition as described above, but with a heavier bias supplied to non-border vertices in order to retain approximately 65% of the border vertices. Three restarts are allowed after which the genetic algorithm is reinitialised.

The fitness of an individual was defined to be minus the product of the number of cut edges times the imbalance. JOSTLE occasionally produces partitions violating the balance constraint which are strongly penalised by this scheme.

The initial population was produced by repeatedly partitioning the graph with JOSTLE using random but small biases, of the order of 0.1.

4 EXPERIMENTAL RESULTS AND DISCUSSION

We have implemented the algorithms described here within the framework of JOSTLE, a mesh partitioning software tool developed at the University of Greenwich and freely available for academic and research purposes under a licensing agreement[2]. The experiments were carried out on a variety of different machines; with its very long runtimes (of several days in the case of the larger graphs), the evolutionary search approach can soak up CPU cycles and the tests were run so as to use up any spare capacity in the system. As a result we have not measured runtimes.

4.1 Results on unstructured meshes

Table 1: A summary of the test graphs

	size		degree			
graph	V	E	$\leq$	$\geq$	avg	type
data	2851	15093	17	3	10.6	3D nodal
3elt	4720	13722	9	3	5.8	2D nodal
uk	4824	6837	3	1	2.8	2D dual
ukerbe1	5981	7852	8	2	2.6	2D nodal
add32	4960	9462	31	1	3.8	circuit
crack	10240	30380	9	3	5.9	2D nodal
4elt	15606	45878	10	3	5.9	2D nodal

The test graphs have been chosen to be a representative sample of small to medium scale real-life problems and include mostly 2D (and one small 3D) examples of nodal graphs (where the mesh nodes are partitioned) and dual graphs (where the mesh elements are partitioned). The test suite also includes one non mesh-based graph, add32.

Table 1 gives a list of the graphs, their sizes, the maximum, minimum & average degree of the vertices and a short description. The degree information (the degree of a vertex is the number of vertices adjacent to it) gives some idea of the character of the graphs. These range from the relatively homogeneous dual graphs, where every vertex represents a mesh element, in these cases a triangle and so every vertex has at most 3 or 4 neighbours respectively, to the non mesh-based graph such as add32 which has vertices of degree 31. As the graphs are not weighted, the number of vertices in V is the same as the total vertex weight $|V|$ and similarly for the edges E.

Graph partitioning algorithms can usually find higher quality partitions if the balancing constraint is relaxed slightly. Indeed some of the public domain graph partitioning packages such as JOSTLE & METIS have an in-built, although adjustable, imbalance tolerance of 3% (i.e. the largest subdomain is allowed to be up 1.03 times the size of the maxi-

[2]available from `http://www.gre.ac.uk/jostle`

mum allowed for perfect balance). We therefore tested the evolutionary algorithm with various tolerances and Table 2 shows a comparison of the cut-weight results with 0% and 3% imbalance tolerances, C_E^0 and C_E^3 respectively, for four values of P (the number of processors/subdomains). For each value of P, the first & second columns show the cut-weight with the allowed imbalance, while the third column shows the ratio of cut-weight for 3% imbalance scaled by that for 0% imbalance, C_E^3/C_E^0. Thus the figure of 0.98 for the data graph and $P = 8$ means that the algorithm was able to find a partition 2% better if allowed a 3% imbalance tolerance. As can be seen, the improvement in quality for these tests is up to 7% and on average is around 3%.

To demonstrate the quality of the partitions, we have compared the results in Table 2 with those produced by a public domain partitioning package JOSTLE (JOSTLE 2.2, March 2000), [18]. Firstly Table 3 shows a comparison of the cut-weight results for the public domain version of JOSTLE compared to the evolutionary search algorithm. These results are an improvement on our previous evolutionary search implementation, [16]. The average difference in the quality ranges from 23% to 20% as P increases and can be as bad as 75%. Note that differences in quality tend to diminish as P increases. It is tempting to speculate that this is because the margins for difference decrease as the number of vertices per subdomain ($\approx V/P$) decreases. Indeed in the limit where $V = P$ the only balanced partition (for an unweighted graph at least) is to put one vertex in each subdomain and so the differences vanish altogether.

4.2 Comparison with the results of Kang & Moon

In this section we compare our results against a recent benchmark of Kang & Moon, [10]. Some of these graphs have similar structure to meshes, but some less structured examples are included, [5, 9].

Three types of graph were tested: U$n.d$, random geometric graphs of n vertices that lie in the unit square and whose co-ordinates are chosen uniformly from the unit interval; Grid$n.b$, a grid graph of n vertices whose optimal bisection size is known to be b and W-grid$n.b$, the same graph with wrapped boundaries; Breg$n.b$, a random regular graph of n vertices each of which has degree 3, and the optimal bisection size is b with probability $1 - o(1)$, [3].

We expect the random geometric graphs and grid graphs to be suitable for the crossover and mutation operators because of their geometric origin – they both arise from the embedding of graph vertices in a low dimensional Euclidean space, with only local connections between points giving rise to edges. The randomly generated Breg$n.b$ graphs, with edges possible between any pairs of vertices do not exhibit the structure required by the heuristic bias of the genetic algorithm. Caterpillar graphs were not used since they have a very different structure altogether.

Kang & Moon's benchmarks were produced by running their genetic algorithm 50 times on each problem graph, with each run given an allotted CPU time and keeping the best. The objectives of our experiments were twofold. Firstly to test whether our genetic algorithm was robust – could it find partitions as good as those of Kang & Moon without requiring repeated runs (or equivalently the repeated full reinitialisations after 3 restarts) within broadly similar total time budgets. This is a good test of the heuristic bias given to the crossover and mutation operators, since if insufficient very fit offspring are produced, the population will converge and the quota of 3 restarts soon used up. Secondly to support, improve and extend their benchmarks. For 32-way partitioning the number of vertices in the graphs did not divide exactly by 32, so that some partitions will have more vertices than others. We use a less restrictive constraint than theirs, which requires that the partitions differ by no more than one vertex, since we only constrain the maximum allowed number of vertices, so for 32-way partitioning we are extending the benchmark. For 8-way partitioning the number of vertices divides exactly, so our constraint is the same and hence we can justly claim to support and improve on their results.

The same parameters were used on all experiments except that for the W-grid5000.100 and U1000.40 graphs, the ratio of crossover to mutation was increased from 3::7 to 7::3. This slowed the rate of convergence of the population, allowing a more thorough search and providing evidence for the effectiveness of the crossover operator. Table 4 shows our results, giving the minimum number of cut edges found (with the change relative to those of Kang & Moon in brackets) and the number of subgraph evaluations taken to find the result.

For 8-way partitioning the results support or improve on those of Kang & Moon, except for the graph Breg5000.16. Even though improved results were found for the remaining examples of the Breg graphs, they required substantially more subgraph evaluations to find results as good as those of Kang & Moon, in agreement with our expectations of performance on this type of graph.

For 32-way partitioning, because of our less exacting constraint on the imbalance, we expected to find less cuts than the previous benchmark. This turned out to be the case, except for the Breg graphs. More evaluations were allowed for these graphs, since the genetic algorithm converged very slowly, showing further potential. Breg5000.0, the most suitable for our algorithm given its construction, eventually yielded less cut edges than the more constrained partition of Kang & Moon. Our genetic algorithm made much slower progress towards partitions of similar quality

to their benchmark on the Breg graphs.

The robustness of our genetic algorithm was confirmed by its requiring only one run to match the benchmark in almost all cases.

5 SUMMARY & FUTURE WORK

We have described and tested an evolutionary search algorithm for partitioning graphs and reported new benchmark partitions that it found. Distinctive features are the transmission and modification of whole subdomains (the partitioned units) that act as genes, and the use of a multilevel heuristic algorithm to effect the crossover and mutations. These features implement an heuristic bias suitable for graphs such as unstructured CFD meshes and their effectiveness is demonstrated by improvements on previously established benchmarks.

In future we aim to look at the integration of the evolutionary search procedure more fully into the multilevel framework. We also intend to study more carefully the parameters which govern the interaction between the multilevel scheme and the evolutionary algorithm.

References

[1] L. Altenberg. The Schema Theorem and Price's Theorem. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 23–49. Morgan Kaufmann, San Mateo, 1995.

[2] S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice & Experience*, 6(2):101–117, 1994.

[3] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph Bisection Algorithms with Good Average Case Behavior. *Combinatorica*, 7(2):841–855, 1987.

[4] T. N. Bui and C. Jones. A Heuristic for Reducing Fill-In in Sparse Matrix Factorization. In R. F. Sincovec *et al.*, editor, *Parallel Processing for Scientific Computing*, pages 445–452. SIAM, Philadelphia, 1993.

[5] T. N. Bui and B. R. Moon. Genetic Algorithms and Graph Partitioning. *IEEE Trans. Comput.*, 45(7):841–855, 1996.

[6] L. J. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann, San Mateo, 1991.

[7] B. Hendrickson and T. G. Kolda. Graph Partitioning Models for Parallel Computing. *Parallel Comput.*, 26(12):1519–1534, 2000.

[8] B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In S. Karin, editor, *Proc. Supercomputing '95, San Diego*. ACM Press, New York, NY 10036, 1995.

[9] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: Part I, Graph Partitioning. *Oper. Res.*, 37(6):865–892, 1989.

[10] S. Kang and B. R. Moon. A Hybrid Genetic Algorithm for Multiway Graph Partitioning. In D. Whitley *et al.*, editor, *Proc. Genetic & Evolutionary Comp. Conf. (GECCO-2000)*, pages 159–166. Morgan Kaufmann, San Francisco, CA 94104, 2000.

[11] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[12] B. W. Kernighan and S. Lin. An Efficient Heuristic for Partitioning Graphs. *Bell Syst. Tech. J.*, 49:291–308, 1970.

[13] K. Mathias, L. Eshelman, J. D. Schaffer, L. Augusteijn, P. Hoogendijk, and R. van de Wiel. Code Compaction Using Genetic Algorithms. In D. Whitley *et al.*, editor, *Proc. Genetic & Evolutionary Comp. Conf. (GECCO-2000)*, pages 710–717. Morgan Kaufmann, San Francisco, CA 94104, 2000.

[14] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer-Verlag, Berlin, 1982.

[15] H. D. Simon and S.-H. Teng. How Good is Recursive Bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, 1997.

[16] A. J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Approach to Graph Partitioning. In D. Whitley *et al.*, editor, *Proc. Genetic & Evolutionary Comp. Conf. (GECCO-2000)*, pages 674–681. Morgan Kaufmann, San Francisco, CA 94104, 2000.

[17] M. Vazquez and D. Whitley. A Hybrid Genetic Algorithm for the Quadratic Assignment Problem. In D. Whitley *et al.*, editor, *Proc. Genetic & Evolutionary Comp. Conf. (GECCO-2000)*, pages 135–142. Morgan Kaufmann, San Francisco, CA 94104, 2000.

[18] C. Walshaw and M. Cross. Mesh Partitioning: a Multilevel Balancing and Refinement Algorithm. *SIAM J. Sci. Comput.*, 22(1):63–80, 2000. (originally published as Univ. Greenwich Tech. Rep. 98/IM/35).

Table 2: A comparison of cut-weight results for the evolutionary search algorithm with 0% and 3% imbalance tolerances, C_E^0 and C_E^3 respectively

graph	$P=8$			$P=16$			$P=32$			$P=64$		
	C_E^0	C_E^3	$\frac{C_E^3}{C_E^0}$	C_E^0	C_E^3	$\frac{C_E^3}{C_E^0}$	C_E^0	C_E^3	$\frac{C_E^3}{C_E^0}$	C_E^0	C_E^3	$\frac{C_E^3}{C_E^0}$
data	671	656	0.98	1135	1118	0.99	1811	1783	0.98	2859	2795	0.98
3elt	348	336	0.97	596	565	0.95	963	949	0.99	1553	1524	0.98
uk	91	86	0.95	152	144	0.95	263	249	0.95	419	412	0.98
ukerbe1	113	111	0.98	201	196	0.98	338	334	0.99	548	542	0.99
add32	71	68	0.96	126	117	0.93	218	212	0.97	544	520	0.96
crack	683	683	1.00	1091	1076	0.99	1703	1662	0.98	2603	2519	0.97
4elt	552	529	0.96	946	909	0.96	1571	1530	0.97	2618	2552	0.97
Average			0.97			0.96			0.98			0.98

Table 3: A comparison of cut-weight results for JOSTLE, C_J^3, against those of the evolutionary search algorithm, C_E^3, both with 3% imbalance tolerance

graph	$P=8$		$P=16$		$P=32$		$P=64$	
	C_J^3	$\frac{C_J^3}{C_E^3}$	C_J^3	$\frac{C_J^3}{C_E^3}$	C_J^3	$\frac{C_J^3}{C_E^3}$	C_J^3	$\frac{C_J^3}{C_E^3}$
data	756	1.15	1263	1.13	2106	1.18	3140	1.12
3elt	418	1.24	603	1.07	1020	1.07	1666	1.09
uk	106	1.23	180	1.25	315	1.27	490	1.19
ukerbe1	121	1.09	233	1.19	378	1.13	593	1.09
add32	106	1.56	180	1.54	257	1.21	909	1.75
crack	751	1.10	1191	1.11	1804	1.09	2733	1.08
4elt	656	1.24	1012	1.11	1687	1.10	2772	1.09
Average		1.23		1.20		1.15		1.20

Table 4: The results of the evolutionary search algorithm with a 0% imbalance tolerance on the partitioning benchmark graphs showing the cut-weight, $|E_c|$, and the number of subgraph partitions required to find it

graph	$P=8$		$P=32$	
	$\|E_c\|$	# evals	$\|E_c\|$	# evals
Grid1000.20	114 (-0)	6932	302 (-12)	52183
Grid5000.100	250 (-0)	19639	658 (-1)	437063
W-grid1000.40	172 (-4)	4566	372 (-12)	246908
W-grid5000.100	400 (-0)	215090	811 (-9)	1342821
U1000.10	180 (-7)	57350	559 (-18)	463057
U1000.20	812 (-0)	10411	2325 (-42)	327932
U1000.40	2562 (-0)	70147	7241 (-88)	280541
Breg5000.0	1079 (-17)	457009	1675 (-45)	4991004
Breg5000.4	1081 (-12)	498954	1779 (+54)	2915867
Breg5000.8	1079 (-19)	450115	1786 (+49)	2814953
Breg5000.16	1180 (+103)	1148758	1755 (+62)	2941382