A Combined Evolutionary Search and Multilevel Approach to Graph Partitioning

Alan J. Soper
School of Computing and Mathematical Sciences
University of Greenwich
Park Row
LONDON SE10 9LS
A.J.Soper@gre.ac.uk

Chris Walshaw
School of Computing and Mathematical Sciences
University of Greenwich
Park Row
LONDON SE10 9LS
C.Walshaw@gre.ac.uk

Mark Cross
School of Computing and Mathematical Sciences
University of Greenwich
Park Row
LONDON SE10 9LS

Abstract

Graph partitioning divides a graph into several pieces by cutting edges. The graph partitioning problem is to divide so that the number of vertices in each piece is the same within some defined tolerance and the number of cut edges separating these pieces is minimised. Important examples of the problem arise in partitioning graphs known as meshes for the parallel execution of computational mechanics codes. Very effective heuristic algorithms have been developed for these meshes 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. A distinctive feature is the use of a multilevel heuristic algorithm to generate an effective linkage during crossover.

1 INTRODUCTION

The need for graph partitioning arises naturally in many applications. For example in finite element (FE) and finite volume (FV) computational mechanics (CM) codes meshes composed of elements such as triangles or tetrahedra are often better suited than regularly structured grids for representing completely general geometries and resolving wide variations in behaviour via variable mesh densities. Meanwhile, the modelling of complex behaviour patterns means that the problems are often too large to fit onto serial computers, either because of memory limitations or computational demands, or both. Distributing a graph (corresponding to the computational and communication requirements of the mesh) across a parallel computer so that the computational load is evenly balanced and the data locality maximised is known as graph partitioning. It is well known that this problem is NP-complete, so in recent years much attention has been focused on developing suitable heuristics, and a range of powerful methods have been devised, e.g [6,7,8]. Because of the NP-complete nature of the problem (i.e. an optimal solution cannot be found in polynomial time) it is impossible to know how good the results of such algorithms are. Nonetheless 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 [10] and METIS [7]. We also do not claim this 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 establish benchmark partitions, and to describe the method of finding them.

1.1 OVERVIEW

In this paper we discuss a strategy for combining evolutionary search techniques with a standard graph partitioning method. A particularly popular and successful class of algorithms which address the graph partitioning problem are known as multilevel algorithms. They usually combine a graph contraction algorithm which creates a series of progressively smaller and coarser graphs together with a local optimisation method which, starting with the coarsest graph, refines the partition at each graph level.

We employ the evolutionary search algorithm by constructing a population of variants of the original graph (differing from the original only by edge weighting) and then use this multilevel algorithm almost as a ‘black box’ operator to determine their fitness by computing a partition of each which hopefully will also be a good partition of the original graph. The population evolves either by individual members mutating or by several members crossing with each other to generate a different (and hopefully fitter) child.

We have conducted many experiments to test the technique and present some of the results including benchmarks of public domain partitioning packages, and tests for the effectiveness of the evolutionary search. The principal innovation described in this paper is the combination of evolutionary search techniques and a multilevel graph partitioner.

2 MULTILEVEL GRAPH PARTITIONING

In this section we discuss the graph partitioning problem and outline our multilevel algorithm known as JOSTLE, described in [10], for addressing it.

Let $G = G(V, E)$ be an undirected graph of vertices V, with edges E. Typically for graphs arising from FE or FV meshes the edges will model the data dependencies in the mesh and the graph vertices can either represent mesh nodes (the nodal graph), mesh elements (the dual graph), a combination of both (the full or combined graph) or some other special purpose representation. We assume that both vertices and edges can be weighted (with non-negative integer values) and that $|v|$ denotes the weight of a vertex v and similarly for edges and sets of vertices and edges (although it is often the case that vertices and edges are given unit weights, $|v|=1$ for all v in V and $|e|=1$ for all e in E and indeed this is true for all of our test examples). Given that the mesh needs to be distributed to P processors, a partition is defined to be a mapping of V onto P disjoint subdomains.

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 the smallest integer greater than $|V|/P$ 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). It is normal practice in graph partitioning to approximate the communications cost by the weight of cut edges and the usual (although not universal) definition of the graph partitioning problem is therefore to find a partition such that the imbalance is unity and such that the number of cut edges $|E_c|$ is minimised. Our experiments will use an imbalance of 1.03 (or 3%), since this is considered acceptable for mesh partitioning and JOSTLE performs well at this tolerance.

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) [9], and other optimisation algorithms.

The multilevel idea was first proposed by Barnard & Simon [1], as a method of speeding up spectral bisection and improved by both Hendrickson & Leland [5] and Bui & Jones [2], who generalised it to encompass local refinement algorithms. Several algorithms for carrying out the matching have been devised by Karypis & Kumar [7], while Walshaw & Cross describe a method for utilising imbalance in the coarsest graphs to enhance the final partition quality [10].

Graph Contraction

To create a coarser graph $G_{l+1}(V_{l+1}, E_{l+1})$ from $G_l(V_l , E_l)$ we use a variant of the edge contraction algorithm proposed by Hendrickson & Leland [5]. The idea is to find a maximal independent subset of graph edges, or a matching of vertices, and then collapse them. The set is independent if no two edges in the set are incident on the same vertex (so no two edges in the set are adjacent), and maximal if no more edges can be added to the set without breaking the independence criterion.

Having found such a set, each selected edge is collapsed and the vertices, u_1 , u_2 in V_l say, at either end of it are merged to form a new vertex v in V_{l+1} with weight $|v| = |u_1| + |u_2|$. A simple way to construct a maximal independent subset of edges is to create a randomly ordered list of the vertices and visit them in turn, matching each unmatched vertex with an unmatched neighbouring vertex (or with itself if no unmatched neighbours exist). Matched vertices are removed from the list

The Initial Partition

Having constructed the series of graphs until the number of vertices in the coarsest graph is smaller than some threshold, the normal practice of the multilevel strategy is to carry out an initial partition.

Here, following the idea of Gupta [4], we contract until the number of vertices in the coarsest graph is the same as the number of subdomains P, and then simply assign vertex i to subdomain S_i then commence on the expansion/optimisation sequence.

Partition Expansion

Having optimised the partition on a graph G_l, the partition must be interpolated onto its parent G_{l-1}. The interpolation itself is a trivial matter; if a vertex v in V_l is in subdomain S_p then the matched pair of vertices that it represents, v_1, v_2 in V_{l-1}, will be in S_p.

2.2 THE ITERATIVE OPTIMISATION ALGORITHM

The iterative optimisation algorithm that we use at each graph level is a variant of the Kernighan-Lin (KL) bisection optimisation algorithm which includes a local search mechanism to enable it to escape from local minima, with respect to the number of cut edges and single vertex migrations between subdomains.

Our implementation uses bucket sorting, the linear time complexity improvement of Fiduccia & Mattheyses [3], and is a partition optimisation formulation; in other words it optimises a partition of P subdomains rather than a bisection.

The algorithm, as is typical for KL type algorithms, has inner and outer iterative loops with the outer loop terminating when no migration takes place during an inner loop. The inner loop proceeds by examining candidate vertices, highest gain (improvement in cut-weight) first, testing whether the vertex is acceptable for migration (according to imbalance and gain criteria). The algorithm also uses a KL type hill-climbing strategy; in other words vertex migration from subdomain to subdomain can be accepted even if it degrades the partition quality and later, based on the subsequent evolution of the partition, either rejected or confirmed. The algorithm, together with conditions for vertex migration acceptance and confirmation is fully described in [10].

3 COMBINING EVOLUTIONARY SEARCH WITH THE MULTILEVEL GRAPH PARTITIONER

Evolutionary search is a stochastic search technique that generates new points (or individuals which in our case are partitions) in a search space using information from a finite population of already evaluated points. Typically a new population is generated of equal size to the current population, which in turn provides the basis for producing a further population (termed a generation) and so on. This process is given direction by selecting more information from the fitter individuals in the current population when producing new search points [11]; partition fitness taking account of the number of cut edges and the imbalance.

Each new search point is produced by one of two operations: crossover which combines information from more than one individual in the current generation in random fashion, and mutation which randomly modifies a single 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 probablility even when the current generation is already very good [12].

Evolutionary search algorithms have recently been successfully applied to a diverse set of problems providing useful examples of crossover and mutation operators which provide a guide for developing such operators for new problems. The operators described in this paper extend an approach to the Travelling Salesman Problem (TSP) [14] and the Constrained Minimum Spanning Tree Problem (CMSTP) [15], both of which require a search for a set of links satisfying constraints (forming a tour for the TSP) and for which the sum of their costs is a minimum. Clearly the graph partitioning problem is of similar character: find a set of links (cut edges) subject to the constraint that they generate a partition with acceptable imbalance and that the sum of their (edge) weights, which are all unity, is a minimum.

The approach normally works by first defining a parametric representation for candidate tours (or CMSTs), upon which the many crossover and mutation operators available for parametric problems can then act [11]. The parametric representations are produced by "biasing" the link costs, ie adding spurious positive values to the cost of each link, and then applying a particular heuristic algorithm to produce the corresponding tour or CMST. The heuristic algorithm used should give good solutions for a large range of different problems (sets of link costs). We use biased edge weights to alter the output of the heuristic algorithm used here, JOSTLE, but form our genetic operators differently.

3.1 INTERACTION WITH THE MULTILEVEL PARTITIONER

We first describe how JOSTLE responds to a graph with biased edge weights. In fact each vertex is assigned a bias greater than or equal to zero, and each edge a dependent weight of unity plus the sum of the biases of its end vertices. JOSTLE responds to these edge weights so that:

a) When contracting a graph, highest weight edges are collapsed first (subject to their being independent).

b) When performing iterative optimisation, vertex gains are calculated using the biased edge weights.

When applying JOSTLE to a graph with biased edge weights, the general effect will be that vertices with a small bias are more likely to appear at the boundary of a subdomain than those with a large one, and that edges with lower biased weight are more likely to become cut edges than those with higher weight.

It is easy to see for simple small meshes, simulating JOSTLE by hand, that particular partitions are achievable by choosing appropriate biases. However, we choose not to use the vertex biases as a representation for generating all partitions. Such a representation would have very high redundancy and would for the most part produce many partitions of low quality and hence little interest. Instead, for each offspring we explicitly construct a new set of biases, and hence a new partition, from one or more existing parent partitions. The bias values are chosen carefully so that JOSTLE's ability to produce partitions of good quality (with respect to the unweighted graph) is not too much impaired, while at the same time there is information transfer from the parents to the offspring.

Since the bias values are discarded after a new offspring has been produced, the evolutionary search algorithm described here is not a traditional genetic algorithm since no representation (or genotype) is maintained, distinct from an actual partition.

3.2 CROSSOVER OPERATOR

We create a new set of biases from a selected number of parent partitions as follows:

For each vertex in the graph, examine whether in two or more of the parent partitions that vertex is a border vertex (ends a cut edge). If so, assign the vertex a bias value chosen randomly and uniformly from the range [0, 0.01]. Otherwise assign a bias value of 0.1 plus a random number chosen in the same range.

Border vertices will occur in two or more parents if either identical or adjacent cut edges also occur. Either way we take this as evidence that the vertex should remain as a border one in the child - under the assumption that the presence of this particular border vertex is a contributing factor to the fitness of two or more fit parents. Hence a very small bias is assigned.

We make all other vertices less likely to become border vertices by assigning them a larger bias value. Since in both cases the bias value is small - much less than the unit weight of an edge - JOSTLE will produce a partition for the most part optimised with respect to the true edge weights. Hence the requirement of transfer of information to a partition of high quality can occur as required for a successful crossover operator.

Adding a constant to all bias values will have no affect on the partition produced. When considering their effect on graph contraction; only the relative ordering of the vertices by their biases and the edges at a vertex by their weights matters. However their relative size (over and above the effect on ordering) does alter what happens during the optimisation stage since the transfer of vertices between partitions may proceed so as to decrease the sum of biased edge weights, but increase the true total edge weight and hence cut edges. The greater the relative bias towards border vertices remaining so in the offspring, the greater the number that will be preserved on average, but at the cost of the offspring being more likely of poorer quality. The choice of a relative bias of 0.1 towards preserving border nodes is a value that has been shown by the experiments performed to give the right tradeoff between these competing effects. Informally we are relying on JOSTLE to preserve common border vertices from different pairs of partitions that can be "joined up" so as to produce a partition of high quality within the imbalance constraint, in effect finding an appropriate linkage [11].

At each mating, two, three or four mates were randomly chosen to crossover together, a range found to work well in trials on a number of graphs.

3.3 MUTATION OPERATOR

We create a new set of biases from a parent partition as follows:

For each vertex in the graph examine whether it is a border vertex, the neighbour of a border vertex or the neighbour of a neighbour of a border vertex. If so, assign the vertex a bias value chosen randomly and uniformly from the range [0, 0.01]. Otherwise assign a bias value of 2.0 plus a random number chosen in the same range.

Considering the vertex biases as forming a landscape over the graph, the bias at any vertex giving its height, the effect will be a deep, flat-bottomed trench along the partition boundaries. The trench is considered deep since edge weights within the trench will be 4.0 different from those outside, so that JOSTLE's optimisation stage will have a strong tendency to place boundaries within the trench.

Motivations for this arrangement are the following:

a) Since the edge weight biases within the trenches are small compared with unity, the true edge weight, JOSTLE can still successfully optimise within the trenches with respect to the true total edge weight.

b) Unstructured meshes often show considerable regularity, especially locally in the form of translational symmetry, so that good quality partition boundaries are often found, nearby and locally parallel to each other. Hence a good place to look for another partition given an existing partition of good quality is within its trenches.

c) It should be possible to find new partitions mainly within the trenches, that satisfy the balance constraint, since there is freedom for all subdomains to gain and lose similar numbers of nodes when boundaries are shifted to mainly parallel positions. Here we need to take into account that the width of a trench spans three edges and so allows some boundaries to move more than others. Clearly the extent to which this will hold depends on the graph structure and the number of subdomains and their shape.

d) The trenches allow variations orthogonal to the optimisations performed by JOSTLE ie there exist partitions for the most part lying in the trenches that will result from applying JOSTLE to bias values differing in their random parts only. This is because JOSTLE only considers the transfer of a limited number of vertices at a time between subdomains at any optimisation step. The movements of subdomain boundaries to adjacent parallel positions will in general require the transfer of larger numbers of vertices than JOSTLE allows.

Properties a) – d) taken together are desirable properties for a mutation operator since they should lead to the generation of “nearby” partitions of similar, and hence sometimes superior quality.

The choice of the value 2.0 to bias JOSTLE towards placing subdomain boundaries in the trench is again a compromise figure. A larger value on average places more border vertices in the trench but at a cost of poorer quality partitions. Again this value was chosen by experimenting with a number of different graphs.

3.4 FITNESS FUNCTION AND IMBALANCE

The fitness of a partition (to be maximised) was defined to be minus the number of cut edges times the imbalance. The imbalance was included in the fitness measure because the population of partitions can occasionally include individual partitions of greater imbalance than that sought, if JOSTLE fails to satisfy the constaint. This can occur because of a limit on the number of calls to the load-balancer which is hard coded to prevent cyclic behaviour in controlling the two minimisation variables, the cut-weight and the maximum subdomain size. However, the code rarely reaches this limit and (except if the imbalance is set to zero) JOSTLE usually achieves the required balance.

The fitness function imposes a soft, but heavy penalty on partitions with greater imbalance; sufficiently heavy so that partitions within the balance constraint eventually dominated the population as evolution progressed. The required imbalance is an input parameter to JOSTLE and was set at 3%.

3.5 GENETIC ALGORITHM PARAMETERS

Due to the size of the meshes and the time required to execute JOSTLE, a fairly small population size of 50 was used. Small population sizes have been used successfully when hill climbing is effective, and experiments with the mutation operator indicated that this was so.

A new generation was produced as follows: 50 new offspring were produced by either crossover or mutation at a ratio of 7::3. Mating groups of individuals for crossover and candidates for mutation were selected randomly from the current generation, but with each parent participating in at least one trial. The union of the set of offspring and parents was then ranked according to the fitness of the individuals. The best 50 form the new generation.

The fact that members of a population are only ever discarded when offspring of greater fitness are generated is known as an elitist strategy [11]. It is appropriate in this case because it encourages hill climbing, and because most of the offspring generated are not of very high quality [13].

The random initial population was generated by (for each individual) assigning values to all vertex biases chosen randomly and uniformly from [0, 0.1], and then using JOSTLE to generate the partitions. Each of the 50,000 random trials was generated in the same way. 1000 generations, giving 50,000 evaluations of JOSTLE, were allowed for each run of the genetic algorithm.

The genetic algorithm described here is a very simplified instance of the CHC Adaptive Search Algorithm [13] , but lacks incest prevention and restarts. The experiments performed showed that the genetic algorithm was able to produce new best individuals until near the completion of the allotted evaluations.

3.6 RELATED WORK

Martin and Otto [16] have also used a hybrid approach to graph partitioning. Their technique applied random changes to a partition, which was then subject to a local optimisation scheme (Kernighan-Lin) to improve it. Further changes and local optimisations were applied according to a simulated annealing scheme. The particular graphs used were not available for comparison. Mansour and Fox [17, 18] partitioned graphs with a GA using a direct encoding, where the subdomain membership of each vertex was explicitly represented by the value of a gene. Since these values were unconstrained, partitions of arbitrary imbalance were possible. These genes were concatenated and subject to 2-point crossover. The imbalance constraint was progressively enforced during evolution through the use of a penalty term in the fitness function. Meshes of upto approximately 550 nodes were partitioned. A genetic algorithm using the same direct representation has been applied to graph partitioning in the context of circuit partitioning, but was found to be outperformed by a mixed simulated annealing tabu search [19]. In contrast to the above, the work described here uses an optimisation scheme (JOSTLE) as the basis of a crossover and mutation operator for acting on partitions of unstructured meshes.

4 EXPERIMENTAL RESULTS

4.1 EXAMPLE GRAPHS

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 relatively homogeneous dual graph, where every vertex represents a mesh element, in this case a triangle (2D) and so every vertex has at most 3 neighbours, to the non mesh-based graph add32 which has vertices of degree 31.

We have implemented the algorithms described here within the framework of JOSTLE, a mesh partitioning

Table 1: Graph Properties

Graph	$\|V\|$	$\|E\|$	Max degree	Min degree	Ave degree	Type
data	2851	15093	17	3	10.59	3D nodal graph
3elt	4720	13722	9	3	5.81	2D nodal graph
uk	4824	6837	3	1	2.83	2D dual graph
ukerbe1	5981	7852	8	2	2.63	2D nodal graph
add32	4960	9462	31	1	3.82	32-bit adder (circuit)
crack	10240	30380	9	3	5.93	2D nodal graph
4elt	15606	45878	10	3	5.88	2D nodal graph

Table 2: A Comparison of Cut-edge Results for JOSTLE C^3_J, against those of the Evolutionary Search Algorithm C^3_{Ev}, both with 3% Imbalance Tolerance

Graph	P=4		P=8		P=16		P=32	
	C^3_J	C^3_J/C^3_{Ev}	C^3_J	C^3_J/C^3_{Ev}	C^3_J	C^3_J/C^3_{Ev}	C^3_J	C^3_J/C^3_{Ev}
data	453	1.20	763	1.18	1283	1.11	2077	1.10
3elt	203	1.02	406	1.21	630	1.11	1007	1.05
uk	76	1.81	105	1.27	191	1.24	316	1.19
ukerbe1	61	1.02	111	1.01	206	1.05	357	1.05
add32	54	1.64	106	1.54	185	1.58	265	1.25
crack	460	1.28	785	1.16	1242	1.15	1782	1.06
4elt	382	1.20	644	1.22	1024	1.11	1689	1.10

Table 3**:** A Comparison of Cut-edge Results for METIS C^3_M, against those of the Evolutionary Search Algorithm C^3_{Ev}, both with 3% Imbalance Tolerance

Graph	P=4		P=8		P=16		P=32	
	C^3_M	C^3_M/C^3_{Ev}	C^3_J	C^3_M/C^3_{Ev}	C^3_M	C^3_M/C^3_{Ev}	C^3_M	C^3_M/C^3_{Ev}
data	473	1.25	860	1.33	1371	1.19	2146	1.14
3elt	257	1.29	381	1.13	662	1.17	1049	1.09
uk	52	1.24	116	1.40	195	1.27	303	1.14
ukerbe1	78	1.30	133	1.21	235	1.19	394	1.16
add32	107	3.24	94	1.36	219	1.87	285	1.34
crack	474	1.32	784	1.16	1299	1.20	1910	1.14
4elt	359	1.13	759	1.44	1104	1.20	1842	1.20

Table 4: A Comparison of Cut-edge Results for the Random Search C^3_R, against those of the Evolutionary Search Algorithm C^3_{Ev}, both with 3% Imbalance Tolerance

Graph	***P*=4**		***P*=8**		***P*=16**		***P*=32**	
	C^3_R	C^3_R/C^3_{Ev}	C^3_R	C^3_R/C^3_{Ev}	C^3_R	C^3_R/C^3_{Ev}	C^3_R	C^3_R/C^3_{Ev}
data	389	1.03	664	1.03	1181	1.03	1945	1.03
3elt	199	1.00	340	1.01	579	1.02	987	1.03
uk	42	1.00	90	1.08	165	1.07	280	1.06
ukerbe1	61	1.02	111	1.01	206	1.05	357	1.05
add32	33	1.00	69	1.00	117	1.00	212	1.00
crack	361	1.00	697	1.03	1115	1.03	1757	1.05
4elt	321	1.01	544	1.03	948	1.03	1604	1.04

software tool developed at the University of Greenwich and freely available for academic and research purposes under a licensing agreement from http://www.gre.ac.uk/jostle. The test graphs have been chosen to be a representative sample of small to medium scale real-life problems and include both 2D and 3D examples of nodal graphs (where the mesh nodes are partitioned) and dual graphs (where the mesh elements are partitioned). We have also included a non mesh-based graph add32.

4.2 EXPERIMENTAL FRAMEWORK AND RESULTS

To demonstrate the quality of the partitions, we have compared the results obtained using the evolutionary algorithm with those produced by two public domain partitioning packages JOSTLE [10] and METIS [7], and to the best of many (50,000) randomly biased evaluations of JOSTLE. Comparisons of the number of cut edges found at 3% imbalance are recorded in Tables 2, 3 and 4 respectively. Partitions for each of the example graphs were generated for P = 4, 8, 16 and 32 subdomains.

The results show that the evolutionary algorithm was able to find results substantially better than the public domain packages JOSTLE and METIS, and in most cases superior to the randomly biased JOSTLE.

The difference in quality improvement over JOSTLE and METIS tends 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 decreases. Indeed in the limit where $P=V$ the only balanced partition (for an unweighted graph at least) is to put one vertex in each subdomain and so the differences vanish altogether.

For add32 the evolutionary algorithm gave no improvement over the randomly biased trials. This graph shows none of the regularity enjoyed by the "unstructured" meshes and which is exploited by the mutation operator.

Results of the same quality as those obtained using random search were usually found more quickly using the evolutionary algorithm.

4.3 FURTHER WORK

It is intended to perform tests to quantify the performance of the evolutionary algorithm and to understand how it depends on the relative biases of boundary and interior nodes, and the number of parents during crossover. It is also proposed to establish a benchmark archive of graph partitions.

The generation of graphs with known optimal partitions and zero imbalance is easy: e.g. one can simply repeatedly bisect a uniform, rectangular planar graph say 128 x 128 vertices into 16 pieces. It is intended to try to reproduce such partitions using the evolutionary algorithm.

References

[1] 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.

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

[3] C.M. Fiduccia and R.M. Mattheyses. A Linear Time Heuristic for Improving Network Partitions. *Proceedings of the 19th IEEE Design Automation Conference*, pages 175-181, IEEE, Piscataway, NJ, 1982.

[4] A.Gupta. Fast and effective algorithms for graph partitioning and sparse matrix reordering. *IBM Journal of Research and Development*, 41(1/2):171-183, 1996.

[5] B.Hendrickson and R.Leland. A Multilevel Algorithm for Partitioning Graphs. *Technical Report SAND* 93-1301, Sandia National Laboratories, Albuquerque, NM, 1993.

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

[7] G.Karypis and V.Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal Scientific Computing*, 20(1):359-392, 1998.

[8] G.Karypis and V.Kumar. k-way Partitioning Scheme for Irregular Graphs. *Journal Parallel Distributed Computing*, 48(1):96-129, 1998.

[9] B.W. Kernighan and S.Lin. An Efficient Heuristic for Partitioning Graphs. *Bell Systems Technical Journal*, 49:291-308, February 1970.

[10] C.Walshaw and M.Cross. Mesh Partitioning: a Multilevel Balancing and Refinement Algorithm. To appear in *SIAM Journal Scientific Computing* (originally published as University of Greenwich Technical Report 98/IM/35, 1998).

[11] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[12] 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, 1995.

[13] J. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in non-traditional genetic recombination, in G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 205-218. Morgan Kaufmann, 1991,

[14] C. L. Valenzuela and L. P. Williams. Improving Heuristic Algorithms for the Travelling Salesman Problem by Using a Genetic Algorithm to Perturb the Cities. In editor T. Back, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 458-464. Morgan Kaufmann, 1997.

[15] A.J. Soper and S. McKenzie, The Use of a Biased Heuristic by a Genetic Algorithm Applied to the Design of Multipoint Connections in a Local Access Network. In *'GALESIA 97' Second International Conference on Genetic Algorithms*: Innovations and Applications, University of Strathclyde, Glasgow, UK, pages 113-116. IEE Conference Publication 446, 1997.

[16] O. C. Martin and S. W. Otto. Partitioning of Unstructured Meshes for Load Balancing. *Concurrency: Practice and Experience*, Vol. 7(4): 303-314, 1995.

[17] N. Mansour and G. C. Fox. Allocating Data to Distributed-memory Multiprocessors by Genetic Algorithms. *Concurrency: Practice and Experience*, Vol. 6(6): 485-504, 1994.

[18] N. Mansour and G. C. Fox. A Hybrid Genetic Algorithm for Task Allocation in Multicomputers. In editors R. K. Belew and L. B. Booker, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 466-473. Morgan Kaufmann, 1991.

[19] C. Gil, J. Ortega, A. F. Diaz and M. D. G. Montoya. Annealing-based Heuristics and Genetic Algorithms for Circuit Partitioning in Parallel Test Generation. *Future Generation Computer Systems* 14:439-451, 1998.