A Multilevel k–way Partitioning Algorithm for Finite Element Meshes using Competing Ant Colonies

A.E. Langham
Department of Computer Science
University of Wales Swansea
Singleton Park, Swansea, SA2 8PP, U.K.
csjim@swansea.ac.uk

P.W. Grant
Department of Computer Science
University of Wales Swansea
Singleton Park, Swansea, SA2 8PP, U.K.
p.w.grant@swansea.ac.uk

Abstract

The self–organizing properties of ant colonies are employed to tackle the classical combinatorial optimization problem of graph partitioning. Structural information from the graph is mapped onto an environment upon which a number of colonies compete for resources. Using Genetic Programming, a Foraging Strategy is evolved which when executed by the ants in each colony leads to a restructuring of the global environment corresponding to a good partition. Multiple colonies allows for simultaneous k–way partitioning which can provide better partitions than current algorithms which are based on recursive bisection.

1 INTRODUCTION

Scientific Computation, such as simulation of fluid flow using finite element meshes, can be mapped onto a graph problem where each vertex in a graph represents a node in the mesh, whilst an edge in the graph represents the need for communication between two nodes. To distribute the task over several processors, the graph must be broken into approximately equal size sub–domains with as little communication between domains as possible. Hence partitioning involves balancing the computational load, with approximately equal number of nodes assigned to each processor, whilst minimizing the communication between processors with as few edges crossing between processors as possible. Most mesh partitioning is done with standard graph partitioning algorithms as described below.

Partitioning is an NP-complete problem and therefore we are looking for a near optimal partition in reasonable time. Most partitioning methods employ recursive bisection which can often provide a partition which is far from optimal (Simon and Teng, 1993) as regards minimizing the number of edge cuts. What seems optimal at the top level of recursion may provide a poor partition at lower levels given the benefit of hindsight. Recursive Spectral Bisection exploits the relationship between eigenvectors of the Laplacian Matrix of the graph and the structural information of the graph (Simon, 1991) and are highly effective compared to alternative methods. However little work has been done on the comparison of recursive bisection versus producing simultaneous multiple partitions (k-way partitioning) which take a more global view. Recently many spectral based methods have been generalized to partition a graph into more than two sets at each stage of recursion (Hendrickson and Leyland, 1995), however results have not been as promising as recursive bisection methods. Multilevel recursive spectral bisection methods have been shown to outperform single level approaches (Hendrickson and Leyland, 1993b). Partitioning is first done on a very coarse representation of the graph and gradually improved at less coarse–grained levels using the Kernighan Lin heuristic (Kernighan and Lin, 1970) for local improvement.

Swarm based methods use the concept of **stigmergy** introduced by Grasse (Grasse, 1959) whilst studying the building behaviour of termite colonies. He demonstrated that only indirect communication between workers through the environment is needed to perform certain global tasks. Kuntz and Snyers used a clustering technique based on brood sorting in ant colonies to partition graphs for VLSI design (Kuntz et al., 1997) with a colony of ant-like agents.They also used a swarm colonization technique (Kuntz and Snyers, 1994) with coadapting colonies of animats competing for territory by occupying nodes in the graph to be partitioned. Such methods have shown good results on small graphs. Applications of Genetic Al-

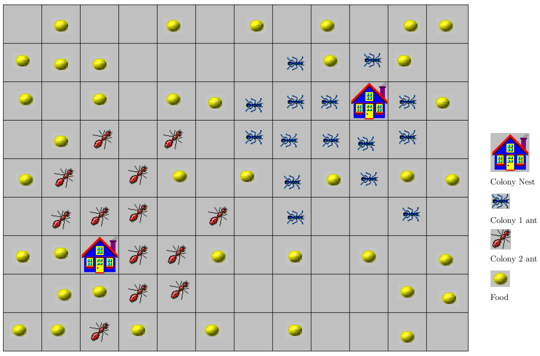

Figure 1: Grid Environment with two ant colonies competing for food.

gorithms (Laszewski, 1991) to graph partitioning have the disadvantage that a solution must be evolved for each new graph and are hence slower than conventional methods. Results are promising when compared to Kernighan Lin but lack of global information means that they do not compete favourably with multilevel spectral methods.

The ML-FS is a multilevel partitioning method. The basic method for partitioning is produced using Genetic Programming (Koza, 1992) and multilevel methods are applied to make the algorithm efficient for large graphs. Results are compared with Recursive Bisection (RSB) and Multilevel Kernighan Lin (ML-KL) as implemented in the package Chaco 2.0 (Hendrickson and Leyland, 1993a).

2 THE APPROACH

2.1 COMPETING COLONIES METAPHOR

Initially consider the bisection case in which two colonies of ants are used to split the graph into two partitions by competing for food. A diagrammatic representation is shown in Figure 1. Each colony is centered around a fixed cell, in a grid which represents the environment in which the ants can navigate. The ants must learn to forage for food, where each piece of food on the grid represents a node in the mesh which is being partitioned. The ants must find all the food and place in the appropriate nest such that the set of nodes represented by the food in $Nest_1$ forms a set V_1 and the set of nodes in $Nest_2$ forms a set V_2. The mesh partitioning problem is equivalent to a graph bisection problem, where given a graph $G = (V, E)$ with vertices V equivalent to nodes in the mesh and edges E equivalent to connected nodes in the mesh, a partition $V = V_1 \cup V_2$ must be found such that $V_1 \cap V_2 = \emptyset$, $|V_1| \approx |V_2|$ and the number of cut edges $|E_c|$ is minimized, where:

$$E_c = \{(v_1, v_2) \in E | v_1 \in V_1, v_2 \in V_2\}$$

Each colony has a fixed number of ants which must cooperate with each other to collect the food. When an ant tries to pick up food the weight of the food determines how many ants are needed for the task. The weight is calculated by finding the number of cut edges created by assigning the associated node to the partition associated with the nest of the current ant. Initially the food is not assigned to any colony and therefore a piece of food will not make any cuts in the graph unless the connected nodes have been assigned to another colony. The weight of each piece of food is determined by the size of the partition it will produce and the number of cuts. If the food is too heavy for one ant to carry it sends out a help signal which can be detected by other ants of the *same* colony within a given radius. Assigned food in another colony can be picked up with probability related to the change in number of cuts produced. Once an ant has found food and picked it up the ant secretes pheremone to act as a trail for other foraging ants. An ant can only distinguish pheremone of an ant from the same colony. Hence each colony must display an emergent self organization to coordinate this task.

2.2 MULTILEVEL PARTITIONING

Partitioning takes place at different levels of granularity. Starting with a coarse–grained representation of the original graph, a series of of ever finer–grained graphs are produced until the original fine–grained graph is used. To map a large graph onto a small one, each node in the small graph represents a cluster of nodes in the large graph. This graph is then made progressively more fine–grained by splitting the nodes in each cluster into two clusters, hence each level doubles the number of nodes in the graph until each cluster contains only one node making it equivalent to the original large graph. The original fine–grained graph G_n is divided into the required number of clusters C which represent the nodes in the coarse–grained graph Gc_1. To produce these clusters the original graph is marked with *distinct nodes*. Consider graph G_n which contains N nodes numbered from $1 \ldots N$, node 1 is a distinct node and all other nodes are distinct if they are not connected to any other node which is distinct. Nodes are checked in order 1 to N. The required number of nodes for the initial coarse graph is C. To form C clusters, C distinct nodes are randomly chosen which form the basis of the clusters. The rest of the distinct nodes are then allocated to the nearest cluster. Nearness is measured by the shortest path to the first distinct node in a cluster. If two clusters are equally near, the one containing the least nodes is chosen. When all distinct nodes are placed in a cluster the rest of the nodes are placed. Each non distinct node is connected to approximately three distinct nodes each belonging to a cluster. If two or more of these distinct nodes are in the same cluster the non distinct node is added to this cluster. Otherwise it is added to the cluster with the least elements. The equivalent graph is produced by treating each cluster as a node and calculating the connections between the clusters. The weight of each connection between nodes in Gc_1 is equal to the total number of edges which cross between the corresponding clusters in G_n. To produce a more fine–grained graph the clusters in the current graph are split in two to produce a graph with twice the number of clusters. To split each cluster roughly in two the nodes in each cluster are sorted using quick sort on the x coordinate. The sorted set is then split in two as equally as possible. If the x coordinate is equal for all nodes in the cluster it is sorted using the y coordinate. Our multilevel method uses a maximum of four levels of coarse–grained graphs, $Gc_1 \dots Gc_4$, then the fine–grained graph G_n. However if one of the coarse–grained graphs $Gc_1 \dots Gc_3$ has more nodes than $N/2$, partitioning proceeds to the node level straight away.

2.3 THE GRID ENVIRONMENT

The *Grid Environment* consists of a collection of cells on a square grid. Each cell can contain a list of food pieces where each piece represents a cluster of nodes. Clusters from the coarse–grained graph are mapped onto the grid in a manner that represents the structural connectivity of the graph. Each cluster from this graph is mapped onto a cell in the grid. The average position of nodes in each cluster is calculated and used to find the average position of all clusters C_{av}. The nearest cluster to this point is considered to be roughly the centroid of the coarse–grained graph and is placed at the centre of the grid and connected clusters are placed in the surrounding cells according to their relative geometrical positions. Each connected cluster then has its connected clusters placed and this is repeated recursively in a breadth-first fashion until all clusters are placed on the grid. Once the food has been placed the nests for each colony are placed at the centre of an equal number of food pieces. This is done using a recursive bisection technique. The median position of food on the grid is used. Consider the bisection case, using a grid of size $G_x * G_y$, nests are placed at (x_2, y_1) and (x_3, y_2). If x_1 represents the median x position of food between 0 and G_x then x_2 represents the median x position of food between 0 and x_1 and x_3 represents the median x position of food between x_1 and G_x. y_1 and y_2 are the median y positions between 0 and G_y in the regions 0 to x_1 and x_1 to G_x. This can be applied recursively to place any number of nests.

2.4 FUNCTIONS AND TERMINALS

Table 1: Functions for Genetic Programming

`if_food_here`	`if_carrying_food`
`if_food_ahead`	`if_help_signal`
`if_food_left`	`if_tired_of_foraging`
`if_food_right`	`if_nest_full`
`if_in_nest_locus`	`if_heavy_food_here`

Table 2: Terminals for Genetic Programming

`move_forward`	`move_to_help`
`turn_right`	`move_to_nest`
`turn_left`	`move_to_away_pheremone`
`pick_up_food`	`move_to_strong_forward_pheremone`
`drop_food`	`move_random`

All ants are placed on the cell representing the colony nest at time t=0. They are initialized to face a random direction i.e North, South, East or West. During each time step an ant can move only to a surrounding grid square. `move_forward` causes an ant to move one grid square in the direction it is facing. `turn_right` causes the ant to change the direction it is pointing by rotating clockwise 90 degrees. Therefore an ant which is originally facing North will then face East. Similarly, `turn_left` causes the ant to rotate its direction by 90 degrees in an anticlockwise direction. `move_random` causes the ant to point in a random direction and move forward one grid square in that direction. When an ant tries to move off the grid it is forced to turn left or right with equal probability. By preventing the ants from wrapping around to the other side of the grid a colony will be less likely to collect food corresponding to a disconnected partition of the graph which is mapped structurally onto the grid. To partition the graph each colony must collect a set of food pieces such that the associated nodes assigned to each colony correspond to a good global partition of the graph. To do this ants must forage for food and bring it back to the nest. When food is picked up it disappears off the map. It can only be dropped if the ant is near the nest, `if_in_nest_locus` is True if an ant is within a distance of 2 grid squares from the colony nest. `drop_food` causes the food to reappear on the map, it is placed around the nest cell using a clockwise search to find the first cell with enough space to hold the nodes corresponding to the food piece.

To coordinate the activity of each colony pheremone trails are used, which act as implicit communication signals. Pheremone is dropped only when an ant is carrying food, hence other ants can follow trails to find regions of the grid containing a high-density of food. Pheremone decays at a rate of 5 percent each time step and hence trails must be reinforced or they disappear. An ant can only detect pheremone trails laid by ants from the same colony.

There are two functions associated with pheremone trails, `move_to_strong_forward_pheremone` causes an ant to move one square either in a forward direction, to the right or to the left with a probability proportional to the amount of pheremone in each square. `move_to_away_pheremone` causes an ant to move away from the nest. If more than one direction will move the ant away from the nest the probability of movement in a particular direction is proportional to the amount of pheremone in the grid square of each away direction. An ant can pick up food if the function `if_food_here` is True. This occurs when there is food on the current grid square which has not already been collected and assigned to the ants colony. This stops ants trying to pick up food already assigned to that colony. Ants can pick up both unassigned food which has not yet been picked up and assigned food which has been placed in another colony's nest. These two eventualities in `pick_up_food` are governed by different rules and are known as **foraging** and **raiding**.

Unassigned food is given a weight corresponding to the number of cuts which will be caused if the associated nodes are assigned to the ants colony. To calculate this, consider the case where each food piece represents 1 node. The total number of cuts depends upon which colonies the connected nodes in the graph have been assigned to. If all connected nodes are unassigned there are no cuts created. All edges will have a weight of one, if more than half the edges will be cut, i.e. have nodes assigned to different colonies the ant cannot pick it up. A typical regular triangular meshes has 6 nodes and hence six edges connected to each node. Hence if more than three of these edges will be cut the node cannot be picked up. If only 1 edge will be cut the food is given a weight of 1, if 2 edges will be cut the weight is 2 and 3 edges cut gives a weight of 3. This weight indicates how many ants are needed to pick up and carry the food. Hence the less cuts the easier it is for the ants to collect a piece of food. In a multilevel implementation each piece of food can represent a cluster of more than one node. The weight of each edge between two connected clusters C_1 and C_2 is equal to the number of edges with one node in C_1 and the other in C_2. Hence the total number of cuts possible between a cluster and all connected clusters is equal to sum of all connected edge weights. If the total number of cuts created is greater than half the number possible then the food cannot be picked up. Otherwise the proportion of cuts caused, $p_c = numCuts/totalCuts$, determines the weight of the food.

$$Weight = \begin{cases} 1 & \text{if } p_c = 0.16 \\ 2 & \text{if } p_c = 0.33 \\ 3 & \text{if } p_c = 0.50 \end{cases}$$

where, $numCuts$ is the number of cuts caused by picking up the food piece and $totalCuts$ is the total possible number of cuts. If a weight of greater than 1 is assigned to a food piece an ant must send out a help signal, which is equivalent to a vibration which can be detected by other ants from the same colony with the function `if_help_signal` is True if there is a signal within a locus of 2 grid squares. The help signal is used to attract other ants as food can only be picked up if the appropriate number of ants are present to support its weight. The function `if_heavy_food_here` relates to food that has a weight greater than 1.

Assigned food is always given a weight of 1, the probability of pick-up is dependent on the change in the number of cuts caused when a piece of food is reassigned to another colony. As with the unassigned food, the cuts are calculated as a proportion of the total possible cuts and an ant can pick it up with a greater probability if the proportion of cuts decreases when food is reassigned. If the proportion of cuts will increase it can be picked up with a much lower probability. The reason for this is to encourage a better partition by making it easier for ants to pick up food which decreases the number of cuts. However, if the ants can only reassign food which reduces the number of cuts the system could easily get stuck in local minima, so moves which increase the number of cuts are allowed with a low probability to improve the search mechanism. Hence the probability of picking up an assigned piece of food is related to δp_c, the change in the proportion of total possible cuts which is caused by reassigning the food. δp_c is equal to the current p_c minus the new p_c.

$$Probability = \begin{cases} 1.0 & \text{if } \delta p_c > 0.0 \\ 0.5 & \text{if } \delta p_c = 0.0 \\ 0.0 & \text{if } \delta p_c < -0.33 \\ 1.0/(2.0 * (\delta p_c)^2) & \text{if } \delta p_c > -0.33 \end{cases}$$

Unsuccessful pick-up leads to `move-random`. The possible size of each partition is bounded by an upper bound U and a lower bound L. The lower bound determines the amount of food which must be present in a colony before ants from another colony can reassign the food thus raiding the colony. L is 60% of the optimal size of each set V_{opt}, where V_{opt} is the number of nodes divided by the number of partitions. This is rounded up to the nearest integer. U is equal to V_{opt} and is used in pick up food. If the number of nodes in a colony is greater than U an ant from that colony will be unsuccessful picking up food. U stops one colony from collecting too much food, hence producing disconnected regions. Similarly L prevents one colony from collecting the food from around another colony forcing that colony to forage further afield and hence producing disconnected regions with more likelihood. Disconnected regions tend to produce a greater number of cuts and are therefore discouraged.

Ants can sense food in the immediate and adjacent squares using the functions, `if_food_here`, `if_food_ahead`, `if_food_right` and `if_food_left` which all return True if food is located in the appropriate square. It is assumed that they have memory of whether the nest is full. `if_nest_full` returns True if the number of food pieces in the nest is greater than or equal to V_{opt}. Each ant also has a memory of how long ago it last encountered a piece of food. It can check this using `if_tired_of_foraging` which is True if no food has been encountered for $2*n$ where n is equal to the side of the grid. It is also assumed that they can remember the position of the colony nest. `move_to_nest` makes an ant move one step towards the nest using the following heuristic

```
if (|Nest.x-x| ≥ |Nest.y-y|)
   then if (x ≥ Nest.x) then dir = WEST
        else dir = EAST
   else if (y ≥ Nest.y) then dir = SOUTH
        else dir = NORTH
```

where (Nest.x,Nest.y) is the position of the colony nest and (x,y) is the current position of the ant. An ant can also move towards a help signal using `move_to_help` which moves the ant one step towards the nearest help signal in a similar fashion.

2.5 EVOLVING AN EFFICIENT ANT FORAGING STRATEGY

To evolve a Genetic Program, a small mesh of 18 nodes and 40 edges with a known optimal bisection of 7 cuts is used. The GP system must find a program which executed by the ants in each colony will lead to the optimal bisection of the corresponding graph by assigning nodes to a colony. The best partition found during execution is stored and used to calculate the fitness. Here the best partition corresponds to that with the least number of cuts. If two partitions have the same number of cuts, then that with the most equal sized sets is stored. A relatively large grid (21 by 21) is used compared to the amount of food to promote an efficient foraging strategy. Nests are placed at (5,10) and (15,10) and the food is placed in three blocks each equidistant from the two nests, centered at (10,5), (10,10) and (10,15). Food is randomly assigned to one of the blocks and is placed around the block in a clockwise fashion similar to placing food around the nest in `drop-food`. Hence ants must use pheremones to collect the food and must raid the other nest to find the optimal partition, as nodes are assigned randomly to food blocks each colony will have difficulty picking up nodes in a connected region. A population of 3000 is used with each member being run for 300 time steps. Each colony consists of 20 ants and each time step involves one evaluation of the Genetic Program for each ant in each colony, unless the ant is helping to carry food. The fitness function is optimal when the optimal bisection is reached with equal sized sets of the desired size (9 nodes) within t_{min} time steps. The standardized fitness f_s is given below.

$$f_s = 1.0 - (0.5 * f_c + 0.3 * f_{ep} + 0.2 * f_t)$$
$$f_c = (|E| - |E_c|)/|E|$$
$$f_{ep} = (|V| - \sum_{0 \leq n < N} \big|\, |V_n| - |V_{opt}| \,\big|)/|V|$$
$$f_t = 1.0 - ((t_{end} - t_{min})/(t_{max} - t_{min}))$$

where, N is the required number of partitions, V_n is the number of nodes assigned to partition n, V_{opt} is the desired size of each partition, $|V|$, $|E|$ are the total number of nodes and edges in the mesh and $|E_c|$ is the number of cuts produced by the current partition. Here, any edge containing an unassigned node produces a cut, otherwise programs can achieve a standard fitness of 0.5 without any foraging activity. The max number of time steps t_{max} is 300 and the minimum t_{min} is 100, with t_{end} being the time to reach the optimal number of cuts. If t_{end} is less than $100 * f_t$ is given a fitness of 1.0. The GP is run for 100 generations with a crossover probability of 0.89, a reproduction probability of 0.1 and a mutation probability of 0.01. Tree size is limited to 50 nodes and all other parameters are as for Koza (Koza, 1992).

2.6 THE MULTILEVEL ANT FORAGING STRATEGY

The ML–FS uses a grid of 21 by 21 cells each holding a list of clusters which represent sets of nodes. Bisection uses populations of 100 ants, quadrisection 50 and octrisection 25 ants per colony. Initially all ants are located in the grid cell representing the colony nest pointing in a random direction North, South, East or West. A maximum of five different levels of graph are used with the first representing the initial coarse–grained graph Gc_1 and the fifth representing the initial fine–grained graph G_n. At the coarse–grained levels food pieces represent clusters of nodes, at the fine–grained level food pieces represent single nodes. The number of clusters in graph Gc_1 is proportional to the number of nodes in the fine–grained graph. Graphs with less than 1000 nodes use 50 clusters, those with less than 3000 use 100 clusters and those with greater than 3000 use 200 clusters. Each coarse–grained level is run for 1000 time steps, where each time step represents one evaluation of the Genetic Program for each ant in each colony. At the end of each coarse–grained level, the state of the grid corresponding to the best partition found so far is reset. A new coarse–grained level is produced by splitting each cluster into two new clusters which are placed in the same position as the original cluster. Whilst partitioning is at a coarse–grained level, ants can pick up only one piece of food from a list of food pieces. Each piece has an equal probability of being picked up. At the fine–grained level ants can pick up a maximum of 10 pieces of food. The number of time steps used at the fine–grained level is proportional to the number of nodes in the graph. 2000 time steps for under 1000 nodes, 4000 time steps for over 1000 nodes and 6000 time steps for over 3000 nodes. If the number of food pieces on the grid is greater than 1000 the system is reset to the best partition so far, every 300 time steps, to stop search getting lost. The final partition is the best partition stored during execution.

3 RESULTS

The Foraging Strategy (FS) evolved is shown below. This best of run individual achieves the optimal partition with 2 sets of 9 nodes within 100 time steps. Results for ML–FS are compared with RSB and ML–KL in Table 3. Figures represent the number of cuts produced by the resulting partitions. ML–KL uses the same number of clusters in the initial coarse–grained graph.

```
( if-in-nest-locus
  ( if-carrying-food ( drop-food )
     ( if-food-here   ( pick-up-food )
     ( move-to-away-pheremone )))
  ( if-carrying-food ( move-to-nest )
    ( if-food-here  ( pick-up-food )
    ( if-nest-full
       ( if-food-ahead ( move-forward )
         ( move-to-strong-forward-pheremone)))
       ( if-food-ahead ( move-forward )
         ( if-food-right ( turn-right )
           ( if-food-left ( turn-left )
             ( if-help-signal (move-to-help)
     (move-to-strong-forward-pheremone))))))))
```

Table 3: Partitions for 4 Meshes over 2,4 and 8 Sets

Edges	Nodes	k	RSB	ML–KL	ML–FS
1046	286	2	29	28	28
		4	91	88	87
		8	176	164	160
3081	1068	2	85	65	61
		4	167	119	119
		8	266	239	236
6049	2067	2	98	104	94
		4	214	205	206
		8	388	360	350
13,722	4720	2	116	119	92
		4	258	236	235
		8	468	393	396

4 DISCUSSION

Bruce and Hendrickson report that both structural information and local improvement strategies are needed to provide a good partitioning scheme. The Foraging Strategy produces good partitions because the method incorporates both global structural information and a local improvement technique. Global structural information is provided by mapping the structure of the graph onto the grid environment and then placing colony nests at the median of an equal number of food pieces. Hence a rough partition is provided by each colony foraging in the local environment. Local Improvement is then provided by raiding other colony food piles and reassigning food with high probability if the number of cuts will decrease. To discourage disconnected regions within a partition set, each colony is placed around an equal number of food pieces and cannot raid another colony food pile until that pile is at least 60% of the optimal set size. Hence, ants must forage most of the food from the local environment rather than raiding, which is used to improve the initial partition. It should be noted that our method is tailored specifically to FEM meshes which are equivalent to sparse graphs in which each node is only connected to others which are physically local.

The Genetic Program evolved (FS) utilizes pheremone trails to coordinate the movements within a colony. Initially ants follow trails away from the nest. Once outside the nest locus they must follow the strongest trail in a forward direction. This stops each ant having to forage randomly each time it leaves the nest. It can follow existing trails which are only reinforced if food has recently been found. When moving around, an ant first checks for food in the current square. If there is none, it checks whether the nest is full. If the nest is full, FS uses a strategy which favours foraging by searching the local environment and using help behaviour. Otherwise, FS favours raiding by following strongly reinforced trails to other nests and returning to the colony nest if no food is encountered within $2n$ steps, thus making raiding more efficient.

The FS contains structural and global information which encourages a good partition and cuts down the search space. This can be seen by considering two swarm based partitioning methods by Kuntz and Snyers (Kuntz and Snyers, 1994). Colonization is faster as the environment represents the structure of the graph, whereas Clustering starts with nodes mapped randomly onto the environment. FS incorporates the structure of the graph in the environment and also incorporates global information by setting L and U to limit the size of each set. This stops unnecessary search involving sets which are unbalanced. Furthermore, nests are placed around an equal number of food pieces to provide an initial partition, hence cutting down on search time. Therefore FS needs a relatively small population. Multilevel methods provide further structural information. A rough initial partition is done on a much smaller graph hence vastly cutting down search time. Further levels of multilevel partitioning refine this rough partition, with each level starting from the best partition found in the last level.

5 CONCLUSIONS

The foraging strategy, described in this paper, produces high quality partitions because it incorporates structural information leading to a rough initial partition and a local improvement technique which is probabilistic and therefore stops the system from sticking in local minima. Results show that k–way partitioning can provide a better partition as it is not dependent on partitions at a higher level of recursion. Since other k-way partitioning methods based on spectral methods show relatively poor results, distributed approaches to k–way partitioning warrant further investigation. We are currently analysing the performance of this algorithm on a set of graphs found in FEM applications and the adaption of multilevel methods to larger graphs found in industrial problems.

References

P. Grasse (1959). La reconstruction du nid et les coordinations interindividuelles; la théorie de la stigmergie. *Insectes Sociaux*, 35:41–84.

B. Hendrickson and R. Leyland (1993a). The chaco user's guide, version 2.0. Technical report, Sandia National Laboratories.

B. Hendrickson and R. Leyland (1993b). A multilevel algorithm for partitioning graphs. Technical report, Sandia National Laboratories.

B. Hendrickson and R. Leyland (1995). An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal of Scientific Computing*, 16.

B.W. Kernighan and S. Lin (1970). An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–308.

J.R. Koza (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA.

P. Kuntz, P. Layzell, and D. Snyers (1997). A colony of ant-like agents for partitioning in vlsi technology. In *Fourth European Conference on Artificial Life*, July.

P. Kuntz and D. Snyers (1994). Emergent colonization and graph partitioning. In *Third International Conference on Simulation of adaptive behaviour: From Animals to Animats 3*, July.

G. Laszewski (1991). Intelligent structural operators for the k-way partitioning problem. In *Fourth International Conference on Genetic Algorithms*, July.

H.D. Simon (1991). Partitioning of unstructured problems for parallel processing. *Computer Systems in Engineering*, 2:135–148.

H.D. Simon and S.H. Teng (1993). How good is recursive bisection? Technical report, Systems Division, NASA, CA.