

Evolving the Building Activity of a Termite Colony for Finite Element Mesh Generation

A. E. Langham* and P. W. Grant

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

WWW home page: <http://www-compsci.swan.ac.uk/~csphil>

Abstract. Previous work in swarm intelligence applications has concentrated on combinatorial optimization problems and is based mainly on the trail laying properties of ant colonies. We tackle a very different kind of problem involving generation of complex structures. Our method is inspired by work on the building behaviour of social insects such as wasps and termites. Structures produced are 2-Dimensional triangular meshes which are used to approximate the solution of fluid flow problems using the finite element method. We employ a Genetic Algorithm to evolve a set of rules which can be used by a colony of termite-like agents to mesh an arbitrary domain. In this approach, meshing is in response to conditions in the local environment, hence changing conditions can easily be accommodated, making it potentially useful for applications such as injection moulding where the mesh must 'grow' with the domain.

1 Introduction.

Meshing involves discretizing a geometrical domain into a set of elements, usually triangles in two dimensional problems. Such elements are used in approximate engineering techniques such as the Finite Element Method, which solve fluid flow problems by approximating a solution at the nodes or vertices of each element. To produce an accurate solution, the domain must be discretized with elements of the appropriate size and shape. Ideally elements should be close to equilateral, relatively small in places where the solution values are changing rapidly (around holes for example) and relatively large where the rate of change is small. The required size of elements is defined by a domain specific density function.

The approach adopted here simulates a group of mobile termite-like agents which collectively construct a mesh by distributing the task. Agents interact indirectly through the environment — there is no central control mechanism — the swarm is coordinated solely by the structure of the mesh. This is known as *stigmergy* and was discovered by Grasse [5]. The success of such an approach

* Supported by a University of Wales Validation Unit studentship

can be seen in natural systems such as insect colonies. In particular wasps and termites build highly complex structures using only simple interactions with their environment and honey bees carry out a variety of three dimensional meshing when building their honey combs. Bonabeau et al. [1] investigated the building behaviour of artificial agents on a lattice capable of depositing or removing bricks according to the state of the local environment. It was found that, with very simple rules, the agents could build very complex structures in both two and three dimensions, many mimicking those seen in nature. Deneubourg et al. [3] investigated similar behaviour on a lattice and compared a stigmergic algorithm with a sequential one. The behaviour of an agent is affected by its past activities in the sequential algorithm whereas in the stigmergic algorithm it is only affected by the state of the local environment. The stigmergic algorithm was found to be more efficient (i.e., building rate per agent is higher) as the colony size increases and also capable of producing more complex structures.

A sequential algorithm was used by Jakiela and Saitou [4] to generate meshes using meiotic cell division to produce elements. Cells are grown from an initial line segment by moving a grow point perpendicularly from the a point on the base until an element of the correct size is formed. Once grown, a cell can then reproduce on one of its two new sides in a similar fashion. After Growth, a cell can be merged with another cell by moving the grow point to a corner of another cell. The algorithm starts with a root cell on the boundary and grows recursively left and then right until all cells are dead i.e. cannot grow. Parameters governing the position of the growpoint on the base and height of the cell are evolved using a set of rules which are dependent on the distance to the nearest boundary or element in eight directions. This can produce many poorly shaped and badly connected elements especially on the boundaries. Also the rules evolved for use on one domain do not generalize to another domain. Each domain needs to have a new set of rules evolved. Inspired by the work of Deneubourg et al. [3], our approach uses information in the local environment so that the rule set can generalize better to a new domain. Furthermore, building activity is not dependent on past activity, only on the state of the mesh in the local environment. This adds greater flexibility as do the addition of new types of building behaviour such as deleting elements which are badly shaped or connected. The algorithm is implemented with probabilistic rules, which give a probability of a particular action occurring, and deterministic rules, which give one possible action given the state of the local environment. Both types of rules are investigated and the quality of the mesh and generalization to new domains is compared.

2 The Method

A domain D is a 2-Dimensional polygon with a density function d determining the required size of elements at a given point in the domain. The domain is equivalent to the nest environment for the termite colony. It is required to mesh the domain with elements of the desired size and shape each connected ideally to six other elements. Because the value of a density function can vary greatly

across a domain the area describing the local environment and all distances used in the building operations are related to the density function.

2.1 Building operations

There are various operations, and combinations of operations, which can be performed. The basic building block is element growth which is similar to the cell growth mechanism used by Jakiela and Saitou [4]. Here growth always starts from the midpoint of the base segment \mathbf{x} and the grow point is moved perpendicular to the base segment until the maximum perpendicular height h is reached, the new element touches another element or the domain boundary. The perpendicular height of the element h is calculated as follows: $h = 0.75 * (d_h + e_h) / 2.0 + 0.25 * d_h$ where d_h is the desired height calculated from the density function d and the length of the base segment b , $d_h = 1.0 / (d(\mathbf{x}) * b)$ and $e_h = b * \sqrt{3.0} / 2.0$ which corresponds to the height for an equilateral triangle with base length b . An element can grow from the right, left or base segment. This segment becomes the base segment for the new element. The three growth operations are denoted as follows \mathbf{G}_L for grow left, \mathbf{G}_R for grow right and \mathbf{G}_B for grow from the base. After growth, an element can merge with the point of another by moving the growth point to this point if the distance to this point is less than $0.5 * d_h$ and the merge does not cause an overlap with other elements.

Morph and Delete are dependent on the average fitness of the current element and all elements connected to the grow point of that element. This average fitness is called the local average l_{av} and is a measure of the average size and shape of each element and given by $l_{av} = 0.6 * l_{shape} + 0.4 * l_{size}$, where l_{shape} is the average fitness for the shape of each connected element, and l_{size} is the average fitness for the size of the connected elements. l_{shape} and l_{size} are defined in a similar manner to f_{shape} and f_{size} in section 2.4 for a set of e elements, where e is the number of elements incident on the grow point of the current element. The Morph operation \mathbf{M} tries to improve the local average l_{av} by moving the grow point of the current element. This point is moved randomly within a small radius until a position is found which increases l_{av} or 10 attempts have been made without success. When the point is on a boundary, morph moves that point within the same distance but only along the line segment the point is currently on. The Delete operation \mathbf{D} deletes all connected elements and the current element if the local average is below 0.7. This is followed by the Jump operation \mathbf{J} which moves to another element making the new element the current element.

Badly connected meshes occur when other elements are too close but not joined, hence preventing growth. When a node is within $0.3 * d_h$ of the centre point of any side of an element it is called a *hanging node* and a special operation is used, Hanging Node Delete, \mathbf{H} . This deletes all elements containing the hanging node and then grows from the side of the current element where the hanging node was found. As with any grow operation this is followed by a merge which tries to merge with another element or the boundary and is only successful if no overlap is produced between elements. To stop small elements appearing on the

boundary, after a morph or grow operation, a Join operation is used to make two elements into one larger element. Morph can produce long thin elements as it may make one element worse in order to improve the fitness of the rest. Grow can also produce small elements if there is not enough space to grow an element of the desired size. Hence such elements can be joined into one element if the size of the element concerned is less than $0.3 * d_A$. Join can only occur if two elements both share a line segment which is not on a boundary line segment and both have another line segment which is on the same boundary line segment.

2.2 Main algorithm

The main algorithm consists of three phases: Boundary Growth, Inner Growth and Spice Up. Boundary Growth and Inner Growth use exactly the same rule set but, during the Boundary Growth phase (Figure 1) all Grow operations are restricted to element line segments with at least one node on a boundary line segment. In contrast, in the Inner Growth phase, Grow operations can be attempted from any element line segment. This produces well connected elements along the boundary which are difficult to produce without the Boundary Growth phase. Growth begins from root elements, which are placed recursively on each line segment of the boundary as follows. A root is placed at the midpoint of each line segment. Roots are then placed recursively at the midpoints of the left and right segments, if the density function value is less than the value at the nearest root and the distance to the nearest root is greater than $3.0 * d_h$ of the midpoint of the current segment. Otherwise if the density is greater than or equal to that at the nearest root, a new root is placed in the midpoint of this segment if the distance to the nearest root is less than $3.0 * d_h$ of the midpoint of the current segment.

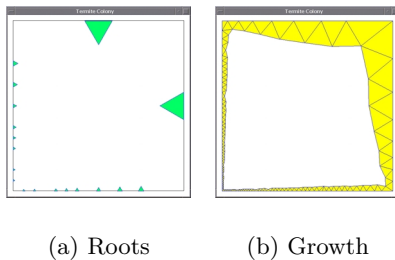


Fig. 1. Boundary Growth Phase

Spice Up is used to finish off the mesh if the whole area has not been covered or a hanging node has not been removed. It repeatedly removes hanging nodes and forces elements to grow if possible until the whole domain is filled. The algorithm then jumps to random elements calling Morph to improve the local

area fitness of elements. Spice up is not used during evolution of the rule set only when running on an arbitrary domain to make the method more efficient as it can be relatively time consuming to fill in the last one percent of the area.

To simulate the activity of a colony of termites on a sequential computer, the algorithm uses the Jump operation which can jump to any element in the mesh not just in the local area. This simulates the activity of many termites at different points in the domain each carrying out building operations. However to make mesh generation more efficient, Jump is biased towards elements which can grow. As the proportion of elements which can grow decreases, Jump is more likely to move to an element which can grow. This greatly increases the speed at which the mesh can be generated as a random Jump would make it difficult to find the last few elements which are able to grow. Jump either moves to a random element which can grow with probability P_{Jact} or a random element which cannot grow. This decision is related to the proportion of elements which are currently active or able to grow p_{act} as follows

$$P_{Jact} = \begin{cases} 0.0 & \text{if } p_{act} = 0 \\ 0.95 & \text{if } p_{act} \leq 0.05 \\ 0.90 & \text{if } p_{act} \leq 0.10 \\ 0.70 & \text{if } p_{act} \leq 0.30 \\ 0.50 & \text{otherwise} \end{cases}$$

Each phase has a termination criterion. Boundary Growth terminates when less than 0.05 of the elements can grow along the boundary. Inner Growth terminates when less than 0.01 of the elements are active or can grow. Spice Up is then run which terminates when all elements are inactive and cannot grow.

2.3 Rule format

Rules are of the form **if condition then action**. The condition part relates to information in the local environment (Figure 2) and has seven components. The action part relates to a building operation or set of possible operations with associated probabilities. The first three fields of the condition relate to the state of each line segment of the current element, representing whether it is possible to grow off the left, right and base segments. These fields are denoted by Can_{gl} , Can_{gr} , Can_{gb} , for can grow left, can grow right, can grow off base and take values **true** or **false**. The fourth field represents the local area fitness. If l_{av} is less than 0.75 it takes the value **true**, otherwise it is **false**. This field is denoted by Can_d . The fifth field is also boolean and is true if there is a hanging node and false otherwise. It is denoted by Is_h . The last two fields can take integer values in [0..3]. N_c is a measure of connectedness depending on n_c , the number of elements connected to the grow point of the current element given by

$$N_c = \begin{cases} 0 & \text{if } n_c = 1 \\ 1 & \text{if } n_c = 2, 3, 4 \\ 2 & \text{if } n_c = 5, 6, 7 \\ 3 & \text{if } n_c \geq 8 \end{cases}$$

and L_{act} measures the proportion of elements in the local area which are active (i.e. can grow) depending on l_{act} , the proportion of elements which are active in the local environment given by

$$L_{act} = \begin{cases} 3 & \text{if } l_{act} \leq 0.1 \\ 2 & \text{if } l_{act} \leq 0.2 \\ 1 & \text{if } l_{act} \leq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

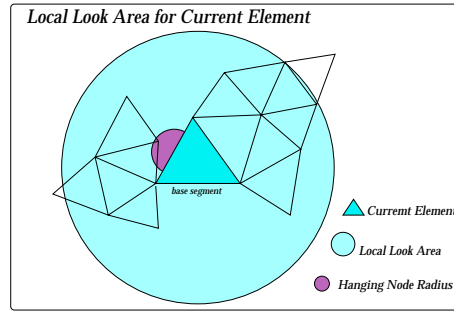


Fig. 2. Local Look Area

Before choosing a rule, the algorithm finds the current values of all fields in the condition and matches this against the condition part of each rule. The rule with the most matches is chosen. If there is more than one such rule, then the first rule is chosen. Once a rule has been chosen the associated action is invoked. Actions can be either deterministic, relating to a single operation, or probabilistic giving a probability distribution of each possible action occurring.

Deterministic Rule

if ($Can_{g_i} = b_1, Can_{g_r} = b_2, Can_{g_b} = b_3, Can_d = b_4, Is_h = b_5, N_c = n_c, L_{act} = n_a$) **then** ($action$)

where, $action \in \{G_L, G_R, G_B, D, M, H\}$

n_c, n_a are integers in $[0..3]$, $b_i \in \{\mathbf{true}, \mathbf{false}\}, i = 1, \dots, 5$.

Probabilistic Rule

if ($Can_{g_i} = b_1, Can_{g_r} = b_2, Can_{g_b} = b_3, Can_d = b_4, Is_h = b_5, N_c = n_c, L_{act} = n_a$)

then ($P_{G_L} = p_0, P_{G_R} = p_1, P_{G_B} = p_2, P_D = p_3, P_M = p_4, P_H = p_5, P_J = p_6$)

where the probabilities $p_0 \dots p_6$ represent the operations 0 to 6 as described in the deterministic rule and sum to 1.0 making a distribution. n_c, n_a are integers on $[0..3]$ and $b_i \in \{\mathbf{true}, \mathbf{false}\}, i = 1, \dots, 5$.

2.4 Evolving the rule set

A Genetic Algorithm [2] is used to evolve a rule set which can mesh the test domain according to the fitness function given below. Each rule set consists of 50 rules. Each rule is represented by a bit string. The condition part needs 9 bits, the deterministic action 3 bits and the probabilistic action 49 bits as each action is represented by 7 bits. The distribution can be found by dividing each number by the total sum of all numbers. A population of 50 rule sets are used. Initially all rules are randomly initialized. However, they are made consistent so that the condition fields do not conflict with the action or set of probable actions. For example if Can_{G_L} is **false** then G_L is not allowed as a possible action. For deterministic rules the action bit string is repeatedly randomly generated until a permissible action is produced. For probabilistic actions, the bits representing an action which is not allowed are all set to zero. If all bits are set to zero in the action string another rule is generated.

Each rule set is run for 800 time steps on the square domain. Each time step represents one action which is derived by matching the state of the local environment against the condition part of each rule. The first rule with the greatest number of matches is invoked. If less than 5 matches are found, a random rule is invoked. If an action is unsuccessful the Jump operation is used to stop the algorithm from sticking. The first 200 time steps are used for Boundary Growth and the remaining 600 for Inside Growth. After 800 time steps the fitness of each mesh produced is calculated.

Fitness function The fitness function f has five components f_s, f_c, f_a, f_{h_n} and f_t which are weighted as follows

$$f = (f_s + f_c + f_{h_n} + 2(f_a + f_t))/7$$

f_s is the average between the shape and size metrics. It is similar to that used in the Morph operation. It is defined as follows, $f_s = 0.4 * f_{size} + 0.6 * f_{shape}$, $f_{size} = \sum_{i=1}^e (1 - |1 - \frac{a}{d_a}|) / e$ and $f_{shape} = \sum_{i=1}^e \sum_{j=1}^3 (1 - |1 - \frac{\cos \theta_j}{0.5}|) / 3e$ where e is the number of elements, a is the area of an element and d_a is the desired area of an element and $\theta_j, j \in \{1, 2, 3\}$ represents the internal angles of an element. This is similar to the fitness measures defined by Jakiela and Saitou [4].

f_c is the average connectivity of elements. All nodes in each element should ideally be connected to six elements including themselves except boundary nodes which should be connected to three other elements. $f_c = \sum_{i=1}^n (1 - |\frac{n_c - d_{n_c}}{d_{n_c}}|) / e$ where n is the number of nodes, n_c is the number of elements connected to each node, and d_{n_c} is the desired number of connected elements.

f_a is related to the proportion of the domain area covered by elements. If this proportion is less than 0.75 then f_a is given the value zero. Hence, $f_a = \frac{p_{area} - 0.75 * D_{area}}{0.25 * D_{area}}$ if p_{area} is greater than $0.75 * D_{area}$ and zero otherwise, where p_{area} is the proportion of the domain area covered by elements and D_{area} is the total area of the domain D .

f_{h_n} is related to the proportion of elements which have a hanging node on one of the growth segments. If this proportion is less than 0.03 then f_{h_n} is given a value of zero. Hence $f_{h_n} = \frac{(1-p_{h_n})-0.97*e}{0.03*e}$ if p_{h_n} is less than 0.03 and zero otherwise, where p_{h_n} is the proportion of elements with hanging nodes.

f_t is the time taken to reduce the proportion of elements which are active to 0.05 i.e $p_{act} \leq 0.05$. If this occurs within a minimum time t_{min} of 500 time steps then f_t takes a value of 1.0. Otherwise f_t is indirectly proportional to the time taken t_{end} to reach $p_{act} \leq 0.05$. Hence $f_t = 1 - (\frac{t_{end}-t_{min}}{t_{max}-t_{min}})$ if $t_{end} \geq 500$ and 1.0 otherwise. Here $t_{max} = 800$ time steps.

Reproduction operators A new generation of rule sets is produced using Crossover on the existing population. Two members are chosen probabilistically using a distribution function proportional to the fitness of each rule set. The probability of Crossover occurring is 0.9, otherwise the first member is copied to the new population. Crossover between two rule sets takes place at the rule level. Crossover produces a new rule set by taking n_1 rules from rule set 1, where n_1 is a random number from $[1..N]$, and $N - n_1$ from rule set 2. The rules which fire are chosen first with equal probability and if there are not enough a non-firing rule is chosen, each with equal probability. Each rule can only be chosen once to avoid unnecessary duplication. Mutation has a probability of 0.1 of changing the value in a field of the rule bit string. If a field which is represented by more than one bit is to be mutated a random bit in that field is flipped. After Crossover and Mutation the rules are made consistent as described earlier. The position of rules in each new rule set is then randomly re-allocated so that the rules chosen from rule set 1 do not dominate in the matching process.

3 Results

Rule Set	R_P	R_P	R_P	R_P	R_D	R_D	R_D	R_D	R_P and R_D
Domain	f_s	f_c	f_{ta}	e	f_s	f_c	f_{ta}	e	Time Steps
D_t	0.88	0.93	1.00	196	0.80	0.89	0.91	196	800
D_1	0.89	0.96	1.00	1147	0.82	0.90	0.92	1014	7521
D_2	0.86	0.86	1.00	161	0.77	0.75	0.93	159	874
D_3	0.87	0.96	1.00	1124	0.83	0.89	0.82	978	9386

Table 1. Mesh Quality for New Domains

The evolved rule sets are run on the test domain D_t for 800 time steps. The rule sets evolved are then tested on three new domains with new density functions as shown in Figure 3. Fitness values are given in Table 1 for f_s , the average size-shape metric, f_c , the average connectivity metric, f_{ta} , the proportion of the total domain area filled. The number of elements e are also shown. The probabilistic

rule set R_P is used to mesh each domain to completion and the deterministic rule set R_D is then run for the same number of time steps to compare results.

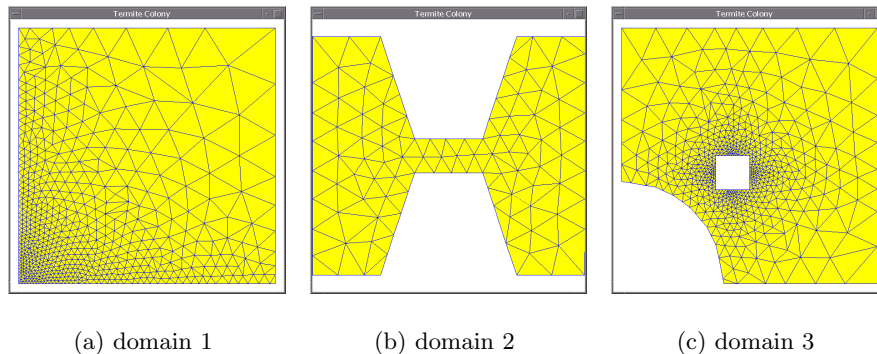


Fig. 3. Probabilistic Rule Set Generalizing to New Domains

4 Discussion

The approach outlined in this paper evolves good meshes because the task is distributed across the domain so that previous building activity can be modified in response to new configurations in the local environment. For example Merge is used to connect newly grown elements to existing elements. Similarly Morph is used to improve the average fitness of connected elements in the local environment and Delete eradicates connected elements in the local environment if the average fitness is below a given value. Meshes evolved by Jakiela and Saitou [4] show poorly shaped elements around boundaries. To overcome such problems our method concentrates first on boundary growth, by setting a root on each line segment this is distributed and generation takes place on all segments simultaneously. The Join operation is also used here to modify elements which have become too small and can be joined into one element of a more useful size. Boundary Growth produces well shaped elements around boundaries as can be seen from Fig 1. which shows root placement and boundary growth around a domain.

Why does this approach generalize better than the sequential approach which needs to evolve a new set of rules for each domain? The boundary growth produces well shaped elements around the boundaries and the extra operations make mesh generation much more flexible (as discussed above). Also the look condition uses information in the local environment which is directly relevant to building activity. For example Can_{g_t} is directly related to the grow left operation. Hence condition–action pairs evolved for use on one domain can be

re-used in new situations. This can be seen from table 1, both the probabilistic and deterministic rules show similar results on new meshes to those on the test meshes. The condition in the sequential algorithm uses the distance to the nearest boundary/element in eight radial directions. One condition can therefore represent rather different configurations of elements and is not so relevant to the possible meshing operations. However the Probabilistic rule set generates much fitter meshes and generalizes better to new domains. This is because each probabilistic rule encodes a set of possibilities rather than just one action. Hence the set of possible actions are greatly increased and generation is less likely to get stuck in a configuration where no building activity is occurring, e.g. if all conditions evoke Morph but the domain is not yet filled. Eventually another action will be evoked leading to a new condition and hence building activity can resume.

5 Conclusion

In this paper we outline a new application area for swarm intelligence and address efficiency problems for swarm algorithms when dealing with such real world problems. Using only stimuli from the local environment a colony of termite-like agents construct a mesh. To coordinate the activity of the group no central control mechanism is required neither is any a priori knowledge of the global domain. The rule sets evolved generate high quality meshes and generalize well to new domains and density functions, with the probabilistic rule set outperforming the deterministic. Applications such as adaptive remeshing and injection moulding which involve changing environments can easily be dealt with by this method as building activity is in response to the state of the current local environment. To make this method more efficient we simulate the activity of a swarm by 'jumping' from element to element. Elements are chosen with high probability if growth can occur. Hence computation is concentrated in areas where it is most needed and information needed to describe a large swarm does not need to be stored.

References

1. E.Arpin E.Bonabeau, G.Theraulaz and E.Sardet. The building behaviour of lattice swarms. In *Artificial Life Four*, July 1995.
2. D.E. Goldberg. *Genetic Algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.
3. G.Theraulaz J.Denoubourg and R.Beckers. Swarm-made architectures. In *First European Conference on Artificial Life*, July 1992.
4. K.Saitou and M.J.Jakiela. Meshing of engineering domains by meitotic cell division. In *Artificial Life Four*, July 1995.
5. P.Grasse. La reconstruction du nid et les coordinations interindividuelles; la theorie de la stigmergie. *Insectes Sociaux*, 35:41–84, 1959.