Wasp-Inspired Construction Algorithms

Marcin L. Pilat
Department of Computer Science, University of Calgary,
2500 University Drive N.W., Calgary, AB, T2N 1N4, Canada
pilat@cpsc.ucalgary.ca

November 11, 2006

Abstract

In this paper, we present our research into wasp-inspired construction algorithms. We were able to implement a typical construction algorithm in a 3D simulation environment and reproduce the results of previous research in the area. Finally, we present case studies that consider the impact of rule changes on the resulting evolved architectures.

1 Introduction

Many species of animals exhibit simple individual actions but in groups are able to demonstrate quite complex emergent behaviours. This idea of self-organization [3] of animal behaviour interests researchers to explore the mechanisms required for this emergent phenomena and try to apply them in other domains (such as computation). In this work, we look at nest building in social wasp species and use a computer model of this behaviour in order to build artificial architectures.

Wasps are predatory or parasitical insects (see Figure 1) characterized by two pairs of wings, few body hairs, and a stinger (present in females). Wasp species can be classified as either solitary or social. Several species of social wasps have been identified and are generally characterized by a colony of individuals working together for the survival of the colony.

Figure 1: Typical social wasp species - the paper wasp (*Polistes dominulus*).

Wasps build nests mainly for reproduction (to host the development of the young). The nests range from simple structures of a few cells (nonsocial species)

to very complex architectures with a million cells (highly social species). A sample of complex wasp nests is provided in Figure 2. A typical social wasp nest is composed of combs containing reproductive chambers called cells (usually hexagonal in shape). Combs are usually joined together with structures called pedicels. Some wasp species build an outer covering of the nest called an external envelope. Wasp nests are usually built from plant fibers and salivary secretions forming a paper carton. However, other building materials such as mud are used by certain species.

Figure 2: Examples of complex social wasp nests.

The nest provides mainly reproductive purpose, however, it also acts as a social hub in social wasps. The external envelope provides protection from the environment (usually through thermoregulation) and a defense mechanism against predators and other biological enemies.

2 Wasp Model

For years, researchers have been working to explain how simple insects like wasps can coordinate their behaviour in order to build such complex nest structure. Applications of such research range from self-assembling robots and evolutionary design to architecture and art [1].

The blueprint theory (proposed by Thorpe [8]) states that wasps store an internal blueprint of the type of nest structure that they build. While building the nest, the wasps compare their blueprint with the environment in order to decide which action to perform. Strong experimental evidence against the existence of such blueprint were later demonstrated (see Smith [5]). The experiments were conducted by modifying a nest structure during construction and observing the effect on the nest building behaviour. It was observed that the wasps were not able to finish the nest as per the blueprint. The experiments of Smith [5] provided an insight onto the nest building strategy of social wasps by showing that cues in the environment seemed to be the driving force in the construction.

Stigmergy [4] is defined as indirect communicating through the environment by leaving signs in the environment that could be picked up by others. Two forms of stigmergy have been identified: quantitative and qualitative. In quantitative or continuous stigmergy, the stimulus in the environment does not change, however, the amount of the stimulus can differ and evoke different responses to the stimulus. This model can be used to explain ant foraging behaviours and termite nest building.

Qualitative or discrete stigmergy uses a discrete set of qualitatively different stimuli that elicit different responses. For example, if an agent notices type 1 stimulus, it will respond with type A behaviour; however, if it notices a qualitatively different type 2 stimulus, it will respond with a different type B behaviour. Through stimulus-action (if-then) pairs, we can construct a rule system that defines the behaviour of single agents. The current best model of wasp nest construction uses the idea of discrete stigmergy.

Using the stigmergy model of wasp nest building, researchers have come up with a class of algorithms that can perform construction of artificial architectures (see [6, 7]). The algorithms use a swarm of agents that move randomly and independently in 3D space and try to match their stimulus-response systems with the local environment. The simulation space is usually a discrete cubic or hexagonal lattice hence the name lattice swarms [7]. The elementary building blocks of the simulation are cubic or hexagonal bricks of different types. If an agent matches its local neighbourhood to a rule in the rule system, it deposits a brick of specified type at its current location in the lattice.

Several methods can be used for rule application. Rules can be matched deterministically or stochastically with a predefined probability. Only one set of rules can be matched or several sets of rules can be used either in seasonal manner or in a hierarchy of rule sets [2]. The algorithms can be either coordinated or uncoordinated [6]. In coordinated algorithms, several runs of the algorithm on the same rule system will yield architectures with common features. The architectures resulting from uncoordinated algorithms may not be similar. A high-level description of the construction algorithm is available in Figure 3.

Algorithm 6.3 High-level description of the construction algorithm

```
/* Initialization */
Construct lookup table /* identical for all agents */
Put one initial brick at predefined site /* top of grid */
For k = 1 to m do
  assign agent k a random unoccupied site /* distribute the m agents */
End For
/* Main loop */
For t = 1 to t_max do
  For k = 1 to m do
    Sense local configuration
    If (local configuration is in lookup table) then
      Deposit brick specified by lookup table
      Draw new brick
    Else
      Do not deposit brick
    End If
    Move to randomly selected, unoccupied, neighboring site
  End For
End For
/* Values of parameters used in simulations */
m = 10
```

Figure 3: Construction algorithm (from [3]).

The space of possible rule systems is very large even for a small number of rules. Rule systems that produce structured architectures with noticeable characteristics are rather rare [6]. Construction is accomplished in parallel by the swarm of agents; however, the resulting architectures provide building constraints enabling the agents to build well structured architectures. Researchers (see [7]) have used coordinated algorithms of lattice swarms to produce architectures that resemble natural wasp

nests. These architectures have clear plateaus of cells joined with either a single axis or multiple pedicels.

3 Implementation

In this work, we implement a coordinated algorithm with randomly moving agents in 3D continuous space and cubic and hexagonal lattices for building of architectures. We use several rule systems in order to reproduce the results of [7].

3.1 Breve

We use Breve for our simulation environment. Breve[1] is a 3D simulation environment specifically designed to simulate decentralized systems. The simulator and its source code are available under the GNU Public Licence (GPL) agreement for Mac OS X, Windows, and Linux. Simulation and testing was done under Breve version 1.9 for Windows.

Breve programs are written in an Object Oriented scripting language called Steve. The programs are then interpreted by the Breve software at run-time. External functions written in C can be called from within Breve through plugins (compiled C executables). The software provides pre-existing object classes and allows for the creation of new classes by subclassing.

The main feature of Breve is the 3D graphical shell that easily allows the user to create and manipulate objects in a 3D environment. The software provides a simple physics engine that models gravity and object collisions. Breve objects are evaluated in a decentralized manner without using global controller code.

3.2 Lattice Models

We implemented two lattice models in Breve: the cubic model and the hexagonal model. The cubic lattice is a regular 3D rectangular grid. Neighbourhood of a cell in the cubic grid is defined as: the 8 neighbouring cells situated on a plane crossing the middle of the cell (z plane), the 9 neighbouring cells in a plane above the cell (z+1 plane), and the 9 neighbouring cells in a plane below the cell (z-1 plane). Thus, neighbourhood of the cell is defined as the states of the 26 neighbouring cells in the 3D cubic lattice. For details, see Figure 4.

The hexagonal grid is of interest because of its closer resemblance to real wasp nest cells. The neighbourhood of a cell in the hexagonal grid is defined as: the 6 neighbouring cells situated on a plane crossing the middle of the cell (z plane), the 7 neighbouring cells in a plane above the cell (z+1 plane), and the 7 neighbouring cells in the plane below the cell (z-1 plane). Thus, neighbourhood of the cell is defined as the states of the 20 neighbouring cells in the 3D hexagonal lattice (as shown in Figure 4).

The 3D lattice simulation grid is represented by the PatchGridLazy class. This class is based on Breve's PatchGrid class with added speed improvements and structure generalization. The main feature of the PatchGridLazy class is lazy cell initialization. The objects representing the cells of the lattice are created when needed (as opposed to entire lattice initialization in PatchGrid). This makes the PatchGridLazy class ideal for simulations with dynamic cell growth and with sparse lattice occupancy.

Further improvement in the PatchGridLazy class is the introduction of the hexagonal lattice (in addition to already existing cubic lattice). The hexagonal lat-

[1]Breve web site can be found at: http://www.spiderland.org/breve

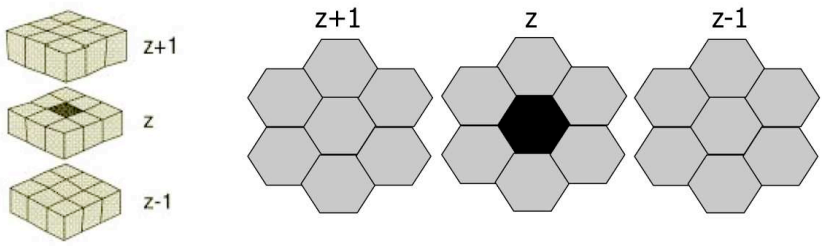

Figure 4: Cubic lattice model[2] and hexagonal lattice model. Black center cell represents the building cell.

tice implementation uses mathematical mappings from hexagonal simulation space to internal 3D sparse matrix representation of the lattice. Easy and quick access methods are provided in order to query the lattice for the neighbourhood state of a cell.

Cells in the PatchGridLazy class are represented as real breve objects of the PatchLazy class (subclassed from Stationary). This class improves on the Patch class of virtual patch objects used in the original PatchGrid representation. The introduction of real objects allows the user to interact with each cell in the lattice during the simulation. The shape of a PatchLazy cell can be specified using a Shape object. A square object is used for cubic grids and a hexagonal disk object is used for hexagonal grids. Cells store state information producing the desired lattice of various brick types. All neighbour queries in the PatchLazy class are done directly to the internal representation, thus not requiring storage of neighbourhood data as in the original Patch class.

3.3 Rule Systems

The rule system is represented by the WaspRuleSet class which stores a list of rules. Rules can be dynamically added to the rule set. Each rule represents the neighbourhood of a central cell and is stored as three slices through the third dimension (z) as follows:

```
{{z+1 states}{z states}{z-1 states}}
```

where each slice list contains 9 numbers (in cubic lattice) or 7 numbers (in hexagonal lattice).

Each z-slice of a rule is stored using a clockwise representation as shown in Figure 5. In the cubic lattice, the first number represents the state of the cell north (using standard coordinate system) of the center cell. The following numbers are states of the corresponding neighbouring cells in a clockwise walk around the center cell's neighbourhood. The state of the center cell is the last number given. In the hexagonal lattice, the first number stored is the state of the center cell followed by corresponding neighbouring cells in a clockwise direction.

Each rule stores both the stimulus and response parts of an if-then rule. The response is the state of the center cell in the z-slice (center of the 3D neighbourhood). States of all other cells are treated as a stimulus and are matched against the local neighbourhood of wasp agents. We use three cell types for cell type matching: 0 for no cell, and 1 or 2 for two defined cell types.

Rule set loading can be accomplished manually through rule specification in the simulation .tz file or from a data file. We implement the rule storage format used

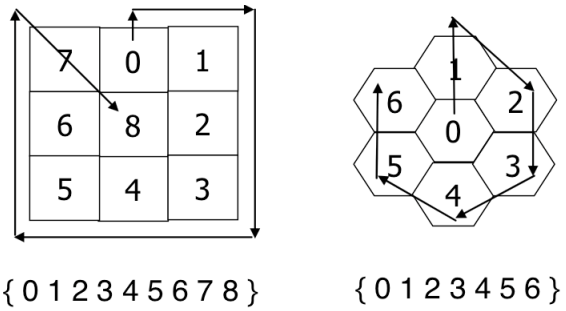

Figure 5: Clockwise representation of z-slice neighbourhood state storage for cubic and hexagonal rules. State of cell 8 in the z-slice is the output of a cubic rule and state of cell 0 in the z-slice is the output of a hexagonal rule.

in the Wasp Nest Building Simulator[2] by Sylvain Guérin. The simulator archive contains many predefined rule sets in .rul file format. Our simulator is able to load those rule set files.

Breve does not provide string parsing functionality. In order to parse our rule set files, we created a Breve plugin (stringfnsp). The plugin is written in C and provides two functions: StringToStringList and StringToIntList. The StringToStringList function takes a string as argument and returns a list containing substrings (space delimited) of the input string. The StringToIntList function offers the same functionality as StringToStringList but converts all substrings to integers and returns a list of integer values.

Through experiments, we have observed that our fixed rule orientation in the simulator prevented us from building many types of architectures. To remedy this, we introduced easy symmetric rule creation on rules added to the rule set. When required, the insert method would create 4 symmetric rules for the cubic lattice and 6 symmetric rules for the hexagonal lattice and insert them all into the rule system. Through the creation of symmetric rules, the north direction is no longer fixed in the rule representation.

4 Results

In this section, we present the results of our experiments. First, we provide results of our evolved architectures on various rule sets. Then, we vary the number of agents in our simulation in order to find the best simulation settings. In the Symmetry Problem section, we take a look at the problem of symmetric rule representation and provide results of the comparison of several methods. Finally, we discuss the impact of rule set modifications on the evolved architecture.

For most of our final experiments, we have used a lattice size of [30x30x30] cells with the initial seed brick at the center of the lattice. Unless otherwise stated, we have used 100 agents moving randomly through the cell lattice. No collision detection was implemented, thus, agents were able to fly through already existing brick objects.

[2]Simulator web site: http://www-iasc.enst-bretagne.fr/PROJECTS/SWARM/nest.html.

4.1 Architectures

We have run our construction algorithm on several cubic and hexagonal rule sets in order to build architectures. In all rule sets tried, we were able to reproduce previously published results (see [2, 6, 7]) and produce interesting architectures for unpublished problems. Some evolved architectures resembled structure of natural nests of certain wasp families. Other, non-natural architectures were evolved for their interesting structure.

Of the natural cubic rule sets, we successfully evolved architectures for the: *Agelaia* (or *Parapolibia*), *Parachartergus*, *Vespula* (or *Stelopolybia*), and *Vespa* wasp families. Figure 6 presents our evolved architectures compared to previously published results [7].

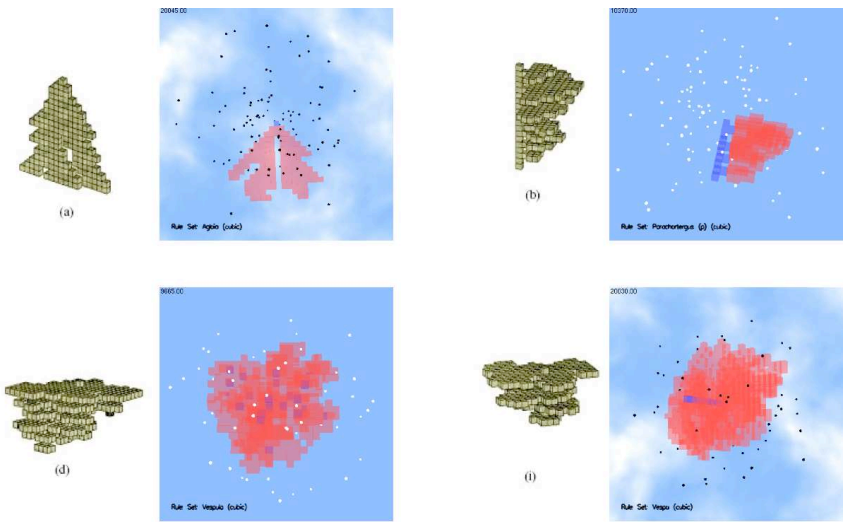

Figure 6: Evolved natural cubic architectures for wasp families: a) *Agelaia*, b) *Parachartergus*, d) *Vespula*, i) *Vespa*. Structures are compared to images from previously published results (on the left of each architecture) [7].

The *Agelaia* wasp family architecture is a flat (1D) tree-like structure with irregular growth. The *Parachartergus* family architectures contain a single axis of cells with plateaus at semi-regular intervals. The architecture for the *Vespula* family contains large plateaus with multiple interconnecting pedicels, while the *Vespa* family architectures use a single axis to connect the plateaus.

We have also simulated other cubic models that do not resemble natural wasp nests. The *immeuble* architecture is a complex multi-part design. The *tunnels* architecture contains a variety of tunnel formations built with 2 different brick types. The *helicoidale* rule set produces and interesting symmetric shape with simple repeating patterns. Images of our simulation results can be found in Figure 7.

The hexagonal lattice simulations provided structure that more resembled wasp nests because of their hexagonal cell structure. We have evolved architectures resembling nests of the following wasp families: *Parachartergus*, *Vespula*, and *Vespa* (see Figure 8). The nest structures were similar to those of corresponding families using the cubic model.

We have also evolved other architectures from the following rule sets: *colonne*, *plateaux*, and *dbl_helice* (see Figure 9). The *colonne* architecture has a symmetric structure with clear separation of the two brick types. The *plateaux* architecture

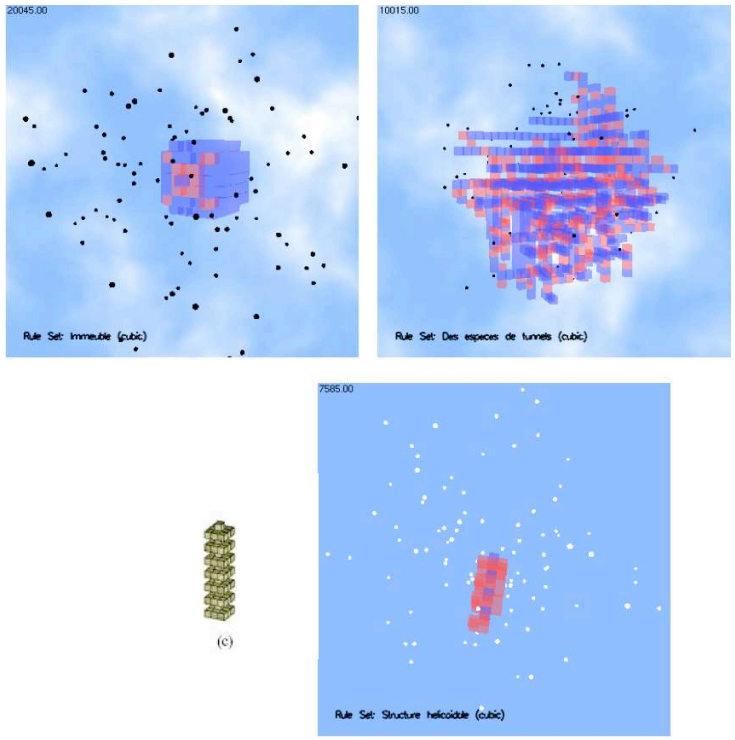

Figure 7: Other evolved cubic architectures: immeuble, tunnels, and helicoidale. The helicoidale architecture is compared to previously published result [7].

contains many small plateaus joined together by pedicels. The *dbl_helice* rule set evolves a double helix type architecture.

4.2 Agent Count

Simulation parameter settings are often critical in order to create proper simulations. In this section, we consider the number of agents parameter in our simulation and what impact it might have on the simulation results. We have used the vespa cubic rule set in order to test various values of the number of agents parameter. Results of our experiments are presented in table 1.

Agent Count	Lattice Size	Cell Count	Level Count	Sim. Steps
50	30x30x30	699 (78)	5.4 (0.5)	30000
100	30x30x30	878 (70)	5.4 (0.5)	20000
200	30x30x30	857 (89)	5.6 (0.5)	10000
400	30x30x30	910 (39)	5.8 (0.5)	5000
800	30x30x30	831 (81)	5.6 (0.5)	2300
50	20x20x20	668 (44)	5.0 (0.0)	30000
100	20x20x20	789 (77)	4.3 (0.6)	20000

Table 1: Summary of our number of agents tuning experiments. All experiments were run for 100 seconds. The cell count and level count values represent averages over 5 (30x30x30 lattice) and 3 (20x20x20 lattice) runs with standard deviation provided in parenthesis.

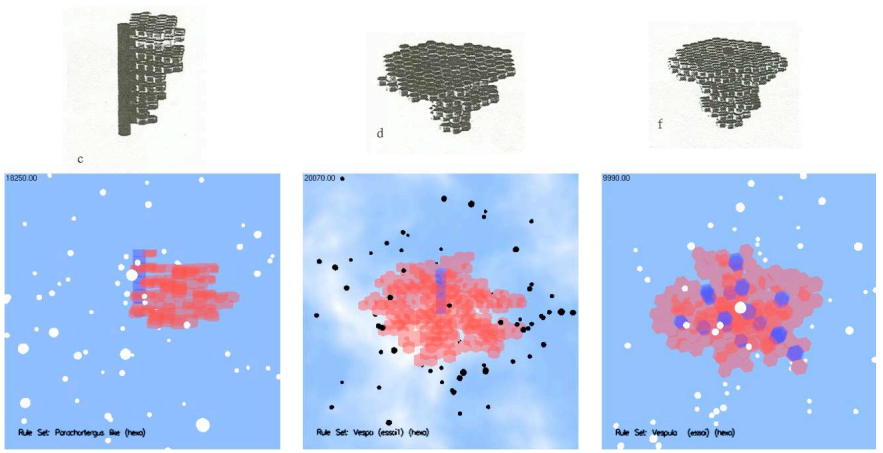

Figure 8: Evolved natural hexagonal architectures for wasp families: c) *Parachartergus* d) *Vespa* f) *Vespula*. Structures are compared to images from previously published results (on top of each architecture) [3].

Figure 9: Other evolved hex architectures: colonne, plateaux, and dbl_helice.

From our experiments, we notice that when using a 30x30x30 lattice size, the number of agents in the simulation (above 100) does not have much impact on the result of the simulation (with a fixed simulation time frame). Agent count of 50 provided results that were not as good as result of using over 100 agents. Using a smaller lattice size provided similar results but the resulting architecture reached the maximum lattice size within the simulation time. From these results, we have used 100 agents and a 30x30x30 lattice size in most of our final experiments.

4.3 Symmetry Problem

While performing our experiments, we have noticed that some rule sets did not produce a reasonable amount of cells. We have traced this problem to the fixed orientation of neighbourhood queries on our lattice. The agent's neighbourhood is defined using a fixed north direction. This presents a problem since rules can only be matched if they are in this fixed representation. However, many rule sets rely on the fact that one rule can be matched against multiple symmetrically oriented local configurations.

To remedy our problem, we have created 3 symmetry models. The NO symmetry model represents the fixed neighbourhood orientation and is necessary for rule sets that produce architectures with no symmetry (for example the *Parachartergus* architecture of Figure 8). The YES symmetry model creates and inserts symmetric

equivalents of each rule in the rule set. The resulting rule set if often much larger than the original but produces better results.

Our final symmetry model - the DIR model tries to improve on the NO symmetry model without introducing new rules. The model uses the direction of an agent in order to orient the local configuration. We have run several experiments to determine the impact of the three symmetry models on the evolved architecture of several rule sets. Results can be found in Table 2.

Rule Set	Symmetry	Cell Count	Level Count
parachart_sly (hex)	NO	24	4
	DIR	31*	6
agelaia (cubic)	NO	72	1
	DIR	6*	1
vespa (cubic)	NO	4	1
	DIR	64	2
	YES	458	4
vespa (hex)	NO	2	1
	DIR	13	1
	YES	228	4
vespula (hex)	NO	4	1
	DIR	20	2
	YES	450	5

Table 2: Summary of our symmetry method comparison experiments. All experiments within a rule set were run for the same number of simulation steps. First set of experiments represent architectures with no resulting symmetry and second set represents all other architectures. A star next to cell count denotes non-symmetry broken by the method.

From our results, the DIR symmetry model improves the performance of the NO model for architectures that can be symmetric. However, for those rule sets, the YES symmetry model performs significantly better than both the NO and DIR models. For rule sets that produce non-symmetric architectures, the DIR model introduced symmetry into the output thus breaking the desired non-symmetry. Thus, we used the NO model for non-symmetric architectures, and the YES model for other rule sets.

4.4 Rule Modification

In order to define rule sets to build specific architectures, we need to understand how rule changes can impact the resulting structures. We study this problem in this section by presenting 3 case studies.

4.4.1 Impact of Symmetry (Double Helix Case Study)

The first way of modifying rule sets can be observed by the introduction of symmetric rules. The double helix (*dbl_helice1*) rule set contains 3 rules. The first rule is responsible for adding the two symmetrically placed bricks making up the double helix strands (after which the rule is no longer used). The second rule extends the double helix strands to the next level and the third rule finishes the next level by extending the internal backbone. The simulation progresses by 2 applications of rule 2 and one application of rule 3 per level.

Simulating the rule set with the NO symmetry method does not produce any reasonable output. Through the YES method, the rule set is grown to 17 rules and

produces desirable double helix structure. We have modified the original rule set to contain 11 rules that produce proper double helix architecture (2 symmetric version of rule 1, 6 symmetric versions of rule 2, and 3 symmetric versions of rule 3). We conclude that with a well defined symmetric architecture, it is relatively easy to come up with a proper rule set.

4.4.2 Addition of Rules (Vespa1 vs. Vespa2 Case Study)

The *vespa1_hex* and *vespa2_hex* rule sets differ by 2 rules (16 vs. 18). The rules enable building of bricks attached to a plateau. We have simulated both of the rule sets for the same number of simulation cycles and the results can be found in Table 3 and Figure 10. The *vespa2_hex* rule set produces larger plateaus with smaller average height compared to the *vespa1_hex* set. The results demonstrate that small additions to the rule set can result in large changes to the evolved architecture.

Rule Set	Cell Count	Level Count
vespa1_hexa (hex)	268 (27)	4.0 (0.9)
vespa2_hexa (hex)	514 (105)	3.5 (1.8)

Table 3: Summary of our experimental results from runs of the two given *Vespa* rule sets. Each rule set was executed for the same number of simulation steps (20000). Cell count and level count values are averaged over 6 runs with standard deviation shown in parenthesis. All runs using the YES symmetry model.

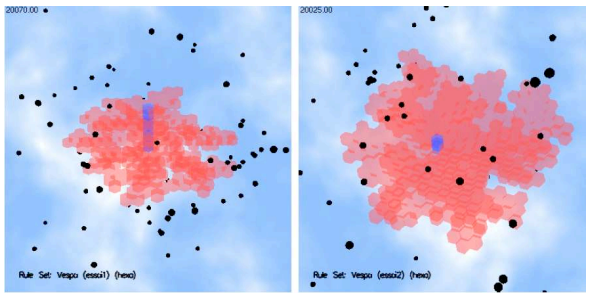

Figure 10: Typical architectures created using the vespa1_hex and vespa2_hex rule sets.

4.4.3 Rule Set Modification (Vespa1 Case Study)

For our third case study, we have used the *vespa1_hex* rule set. We have made several modifications to the rule set to observe the impact on the resulting evolved architectures. Summary of our results is provided in Table 4.

The original rule set (A) contained 16 rules. The first experiment (B) involved addition of 5 extra rules (in addition to 1 already present) for enlarging the central axis. We can see from the results in Figure 11 that the evolved architectures has significantly longer central axis than the original method.

The second experiment (C) moved the axis growth rules within (B) towards the beginning of the rule set. The resulting architectures were higher than those in (B) (see Figure 11 for a comparison). This shows that in such a deterministic rule application system, the placement of rules in the rule set influences the resulting architecture.

Rule Set	Cell Count	Plateau Count	Axis Height
vespa1_hexa (A)	251 (37)	3.0 (1.0)	6.7 (4.0)
vespa1_hexa_sr1 (B)	277 (70)	6.3 (1.5)	11.3 (0.6)
vespa1_hexa_sr2 (C)	220 (36)	8.7 (0.6)	13.3 (2.3)
vespa1_hexa_sr3 (D)	38 (5)	4.7 (0.6)	10.7 (1.5)
vespa1_hexa_sr4 (E)	116 (1)	1.0 (0.0)	1.0 (0.0)
vespa1_hexa_sr5 (F)	269 (20)	1.0 (0.0)	1.0 (0.0)
vespa1_hexa_sr6 (G)	684 (167)	4.7 (1.5)	8.3 (3.1)

Table 4: Summary of our experimental results from runs of 7 varieties of the vespa1_hexa rule set. Each rule set was executed for the same number of simulation steps (20000). Cell count, plateau count, and axis height values are averaged over 6 runs with standard deviation shown in parenthesis. All runs with the YES symmetry model.

The third experiment (D) used the original rule set (A) with all plateau growth rules removed (resulting in 6 rules). The resulting architecture (shown in Figure 11) had a long central axis with small plateaus (one cell radius around the central axis).

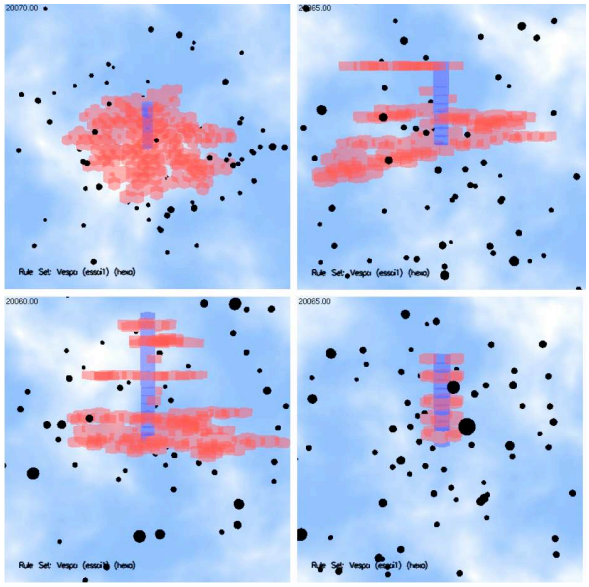

Figure 11: Typical architectures created using the experimental vespa1_hex rule sets: A, B (first row), C, D (second row). Changes in the length of the central axis and width in the plateaus are evident from A to D. All rule sets were executed for 20000 simulation steps.

The fourth experiment (E) used the original rule set (A) with all central axis growth rules removed. All multi-level rules were also removed since they would not have an impact on the architecture. The final rule set contained 10 rules. As expected, the growth only occurred on the plateau in two dimensions. However, the unexpected pattern of growth consisted of empty channels in the resulting plateau pattern (see Figure 12). The formation of the channels was due to the lack of rules that would normally fill in the empty space with a brick.

In experiment (F) we introduced the 3 missing rules to the rule set of (E). The

resulting structure produced a large plateau with no empty channels (see Figure 12). Finally, we have inserted these 3 rules into the original set (A) to form rule set (G). The rule set resulted with an architecture (shown in Figure 12) containing many tightly packed cells.

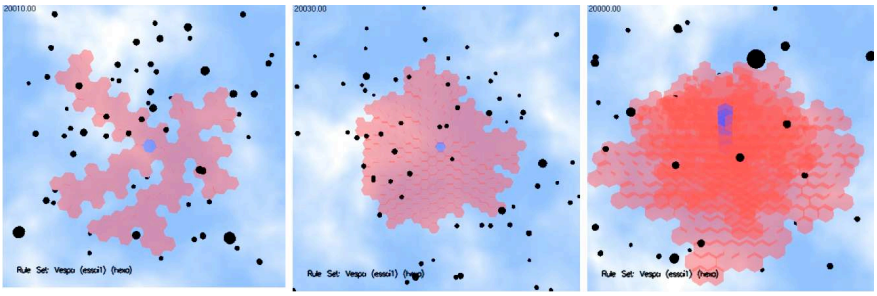

Figure 12: Typical architectures created using the experimental vespa1_hex rule sets: E, F, G. All rule sets were executed for 20000 simulation steps.

5 Conclusions

Through our research, we have demonstrated that coordinated construction algorithms are able to build nest-like architectures. By experimenting with various rule sets, we were able to reproduce the results of [7] using cubic and hexagonal lattices. We have run experiments in order to increase the effectiveness of our algorithm and we have analyzed how changes to the rule systems can impact the resulting architectures.

In further research, we would like to consider different wandering strategies for the agents in our model in order to increase the speed and accuracy of the algorithm. We have considered various symmetry models to solve the fixed orientation problem of our algorithm. The results are promising but further research needs to be done to verify whether better models can be designed.

The main bottleneck of the simulation is the number of rules in the rule system. Further research is necessary in order to determine whether and how can rule sets be simplified without much human input. We think that an evolutionary approach can be used to tune the number of rules in a rule set.

References

[1] Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York.

[2] Bonabeau, E., Theraulaz, G., Arpin, E., and Sardet, E. (1994). The building behavior of lattice swarms. In Artificial Life IV, Brooks, R. and Maes, P. eds., pp. 307–312, MIT Press.

[3] Camazine, S., Deneubourg, J.-L., Franks, N.R., Sneyd, J., Theraulaz, G., and Bonabeau, E. (2003). *Self-Organization in Biological Systems*. Princeton University Press, New Jersey.

[4] Grassé, P.-P. (1984). *Termitologia, Tome II. Fondation des Sociétés. Construction*. Masson, Paris.

[5] Smith, A.P. (1978). An investigation of the mechanisms underlying nest construction in the mud wasp *Paralastor* sp. (Hymenoptera: Eumenidae). *Animal Behaviour*, **26**, 232–240.

[6] Theraulaz, G. and Bonabeau, E. (1995). Coordination in distributed building. *Science*, **269**, 686–688.

[7] Theraulaz, G. and Bonabeau, E. (1995). Modelling the collective building of complex architectures in social insects with lattice swarms. *J. theor. Biol* , **177**, 381–400.

[8] Thorpe, W.H. (1963). *Learning and Instinct in Animals*. Methuen, London.