Planar trinet dynamics with two rewrite rules

Tommaso Bolognesi
CNR/ISTI, Pisa, Italy
tommaso.bolognesi@isti.cnr.it

Abstract
We propose a deterministic network mobile automaton for the creation of planar trivalent networks (trinets) based on the application of only two simple rewrite rules, and we enumerate and explore the possible brownian dynamics of the control point. A useful behavioral complexity indicator is introduced, called revisit indicator, exposing a variety of emergent features, involving periodic, nested and random like dynamics. Regular structures obtained include 1-D graphs, oscillating rings, and the 2-D, hexagonal grid. In two cases only, out of over a thousand we have inspected, a remarkably fair, random-like revisit indicator is found, whose trinets exhibit a slow, square-root growth rate; some properties of these surprising computations are investigated. Finally, one 2-D case is found that seems to be unique in the way regularity and randomness are mixed.

Keywords. Digital physics, trivalent network, complexity indicator, elementary cellular automata, two-dimensional Turing machine, turmite, emergent space, graph rewriting.

1. Introduction

Wolfram [W2002] supports the 'digital physics' view, according to which the ultimate laws of physics are of computational nature, the entire history of our universe is the output of a small, possibly deterministic program, and all simple and complex natural phenomena correspond to emergent properties of this universal computation. In particular, he suggests that physical space could be a giant trivalent network that evolves according to few, simple rewrite rules. Trivalent networks (*trinets*) are finite, undirected graphs where each node has exactly degree 3, that is, three neighboring nodes. It is easy to realize that trinets are sufficient for 'implementing' graph structures of any complexity: given a graph with unrestricted node degrees, the basic trick is to replace any node x of degree n by a cycle X of n nodes of degree 3, each connected to one of x's neighbors. In particular, by letting k be the *dimensionality* of a graph G whenever the number of nodes reachable from a generic node of G in at most r steps (edges) grows like r^k, examples of regular trinets of dimensionality 1, 2 and 3, and of planar trinets with fractal dimensions between 1 and 2 are shown in [W2002] (p. 477 and p. 509),

In general, graph rewriting involves nondeterminism, in the selection both of a rule and of a place where to apply it. Various trinet rewrite rules, and policies for eliminating nondeterminism, are discussed in [W2002]. One solution is to restrict to causal-invariant rewrite systems, which generate a unique partial order of rewrite events, regardless of the order in which rules are applied. Another solution enriches the rewrite process by state information that records the 'age' of nodes, and then always selects the rule and location that involves, say, the youngest nodes. Causal invariance is a powerful and elegant concept, but the search for systems of rules that guarantee this property is hard, unless quite restrictive sufficient conditions are adopted; and the mechanism of time stamps appears as unnatural as, say, the synchrony assumption for the updating of an unbounded set of cells, as adopted by cellular automata. (See [W2002] for a detailed description of these two approaches, which we have already assessed in [B2007a].)

A third solution for reducing nondeterminism is to adopt what Wolfram calls *network mobile automata*: these consist in setting up a single active node, in letting rules replace clusters of nodes around it, and in moving control to an adjacent node. However, despite looking at several hundred thousand cases, involving clusters with up to 4 nodes and 4 dangling links, Wolfram reports that he has not been able to find automata with especially complicated behavior, which explains why this model is relegated to a small note of the NKS book (p. 1040). In conclusion, none of the experiments on network evolution described by Wolfram could fully replicate the success achieved by elementary cellular automata, with their visually appealing, rich variety of emergent properties, and with their ability to create interacting particles, as observed in the well known rule 110 computations.

The two, inter-related objectives of this paper are: (i) to further explore algorithms for the evolution of trinets, and (ii) to identify visually effective complexity indicators and techniques that can help screening large spaces of trinet-based computations. In pursuing the first objective, we have avoided the difficulties related with causal-invariance, and have refrained from enriching the structure of trinets by state information. Rather, we have devised a trinet growth algorithm along the lines of network mobile automata. In summary, the major differences with the approach (cursorily) mentioned by Wolfram in his book are that:

1. we have used an extremely small set of rewrite rules, consisting of only two elements,
2. we have restricted to planar trinets,
3. we have adopted a refined notion of control point -- the *focus* described below,
4. we have attributed more importance to the dynamics of the latter, whose steps are made also dependent on the applied rewrite rule.

Furthermore, we have designed the algorithm around the manipulation of trinet *duals* -- a choice that has to some extent facilitated the identification and exhaustive exploration of policies for control point movement.

In Section 2 we introduce our planar trinet growth algorithm, with its two rewrite rules, and its three parameters, one of which, the *threshold*, induces a convenient classification of our computational space. In Section 3 we discuss a few ways in which the overall character of a trinet computation can be visualized, and introduce a useful *revisit indicator*. Based on this technique, we exhaustively explore our computational classes in Sections 4 to 9, which correspond to increasing values of the threshold parameter, and describe the progressive appearance of various emergent features. In Section 10 we summarize our results and discuss items for future work.

Similar to the space of (elementary) cellular automata, the space of trinet computations, as created by our algorithm, offers such an abundance of aspects to be investigated that a single paper cannot cover all of them in depth. Then, the purpose of this work is to provide a first exploration of the whole computation space, to identify all features that may possibly emerge in it, and to single out the most interesting cases. We believe that these results shall trigger a number of specific questions that we look forward to investigate in forthcoming papers.

2. The algorithm

A *trinet* is an undirected graph where each node has degree 3. Trinets may include loop edges and double edges, and, if v is the number of vertices (nodes) and e is the number of edges, then $3v = 2e$. The proof is simple. By definition, each node is connected to three distinct edges, or to a loop edge and a 'normal' one. By charging three edges to each node, via the incidence relation, we count each edge exactly twice, thus establishing the above equation. Note that a loop edge contributes two units to a node degree. A consequence of the above equation is that v is a multiple of 2 and e is a multiple of 3.

A graph is *simple* when it does not include loop edges nor double edges. Our algorithm shall only handle trinets without loops, but possibly with double edges; we call them *trinets with doubles* (but we shall often omit the attribute). The inclusion of loops introduces further difficulty -- but appears interesting too! -- , and is left for future investigation.
A graph is *planar* when it can be embedded on the surface of a sphere without edge crossings. An embedding partitions the surface of the sphere into *regions*, and induces a *dual* graph (also planar), in which nodes correspond to regions and edges connect the nodes representing adjacent regions. Note that there is an obvious one-to-one correspondence between the edges of a planar graph and of its dual.

If T is a planar trinet with doubles, and D is its dual (see Figure 1, where the two graphs have, respectively, black and white nodes), then:

- regions in T will be delimited by at least two edges, thus nodes in D have at least degree 2 (see, e.g., case (a));
- D may include *both* loops and doubles (cases (c) and (d));
- all the regions of D are triangles, formed by three distinct edges, and every edge is shared by two distinct triangles (recall that one of them may be the external, 'infinite' triangle).

The latter fact is established by realizing that the dual of D is T itself, so that nodes of T, with degree 3, represent faces of D, with three sides; and the two triangles sharing an edge in D are distinct essentially because there are no loop edges in T.

Graph D may include degenerate triangles formed by three distinct edges but less than three nodes. For example, in Figure 1, case (c), the loop edge in the dual graph delimits a finite triangle with two nodes only, while the infinite,

external region of the dual graph of case (d) is a triangle with just one node. (The inclusion of loop edges in trinets would lead to degenerate triangles with two vertices and *two* edges only.) Note, finally, that a loop edge in *D* corresponds to an edge in *T* whose removal disconnects the trinet (see cases (c) and (d)).

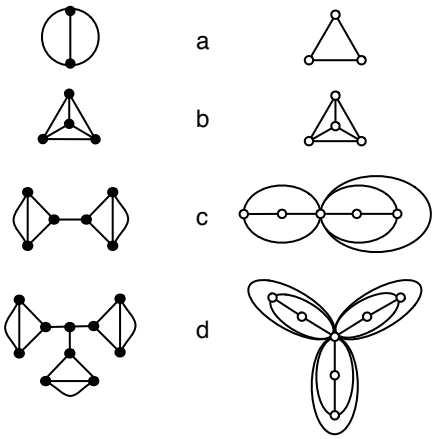

Figure 1 - Four planar trinets (black nodes) and their duals (white nodes)

In our algorithm we shall handle planar graphs -- trinet duals -- *with a specific embedding on the sphere*. For doing this, node adjacency information is not enough: we shall need to keep the list of triangles that form the embedding, as described below.

Representation of spherical (planar) graph embedding

We take now a closer look at the representation and properties of the spherical (planar) graph embeddings manipulated by the algorithm, as a basis for describing the algorithm itself and proving its invariant.

Definitions (oriented edge, oriented triangle, sphericity conditions, spherical set of oriented edge triples)
Let $G(V, E)$ be a connected, undirected graph, where V is the set of v vertices and E is the set of e edges. Let $ex(p, q)$ denote an *oriented edge*, where $ex \in E$ is an adge incident to vertices p and q, and (p, q) is an *ordered* pair. A triple of oriented edges is an *oriented triangle* (shortly, a *triangle*) when (i) it is formed by three *distinct* edges -- that is $ex \neq ey \neq ez \neq ex$, and (ii) its elements can be arranged as in $(ex(p, q), ey(q, r), ez(r, p))$ thus creating a cycle of (at most) three different nodes; which oriented edge appears first in the triple is irrelevant. Node symbols p, q, r are understood as *formal* variables, some of which could be assigned the same *actual* node identifier, thus yielding degenerate triangles. A set of t oriented edge triples, relative to sets V and E, is called *spherical* when it satisfies three *sphericity conditions*:

1. Every triple is an oriented triangle;
2. Every edge is shared by two distinct triangles, with associated node pairs appearing in opposite order -- that is, $ex(p, q)$ and $ex(p, q)$
3. $v - e + t = 2$ (Euler's formula) □

Based on graph theoretic arguments, the following fact can be easily established.

Proposition
If $G(V, E)$ is a connected, undirected graph, for which a spherical set of triples can be built, then G is planar, and every node has at least degree 2. The triples then describe the counterclockwise (by arbitrary convention) traversal of the border of the triangular regions of the spherical embedding of G. □

In particular, the minimum degree 2 is a direct consequence of sphericity condition 1. And, based on conditions 1 and 2, one readily establishes that $3t = 2e$, which, combined with Euler's formula, yields

$e = 3v - 6$
$t = 2v - 4$.

(If condition 3 is replaced by the more general Euler-Poincaré formula $v - e + t = 2 - 2g$, we have sufficient conditions

for embedding graphs on two-manifolds of genus g. For example, when $v - e + t = 0$ the graph can be embedded, without edge crossings, on the torus, whose genus is 1.)

Initial configuration, rewrite rules and computation step

Initial configuration

The most elementary trinet with doubles is the 2-node, 3-edge graph shown in Figure 1.a; thus, all our computations shall start from the corresponding triangular dual graph shown at its right, whose triangular, spherical embedding is:

triangles = {(*e1*(1, 2), *e2*(2, 3), *e3*(3, 1)),
(*e1*(2,1), *e3*(1, 3), *e2*(3, 2))},

One can indeed check that the above data structure satisfies the three sphericity conditions.

Rewrite rules

The two rewrite rules used by our algorithm are illustrated in Figure 2, where the transformation of dual graphs is emphasized.

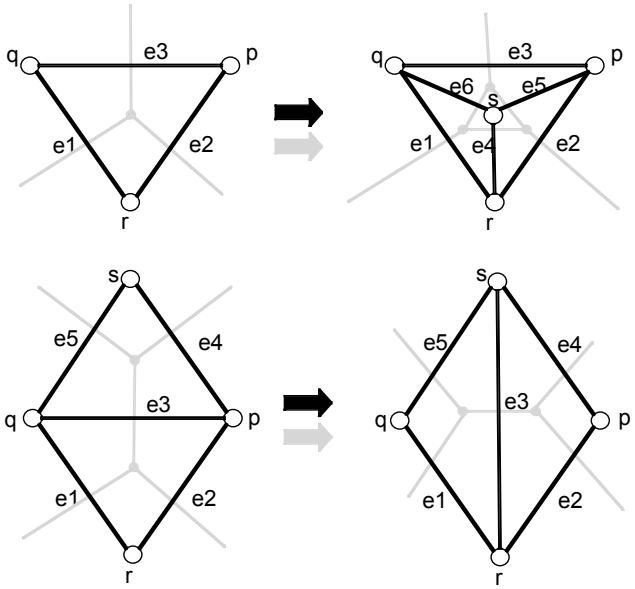

Figure 2 - Planar trinet rewrite rules: Refin (upper), Diags (lower)

In the context of our algorithm, these rules are called, respectively, *Refin* and *Diags*, since the former refines a triangle by partitioning it into three new triangles, and the latter flips the diagonal of a rhombus. These are among the simples rulest considered in [W2002] (p. 509), where their completeness is pointed out: they are sufficient for transforming any planar trinet into any other (with *Refin* used both ways).

Our implementation of these rules operates on sets of oriented triangles. Rule *Refin* removes a triangle and creates three new triples, by introducing a new node (*s*) and three new edges (*e4*, *e5*, *e6*):

removed triangle: (*e1*(*q*, *r*), *e2*(*r*, *p*), *e3*(*p*, *q*))
created triples: (*e1*(*q*, *r*), *e4*(*r*, *s*), *e6*(*s*, *q*)),
(e2(*r*, *p*), *e5*(*p*, *s*), *e4*(*s*, *r*)),
(*e3*(*p*, *q*), *e6*(*q*, *s*), *e5*(*s*, *p*)).

Rule *Diags* removes two triangles sharing an edge (*e3*), and introduces two new triples:

removed triangles: (*e1*(*q*, *r*), *e2*(*r*, *p*), *e3*(*p*, *q*)),
(*e3*(*q*, *p*), *e4*(*p*, *s*), *e5*(*s*, *q*)).
created triples: (*e5*(*s*, *q*), *e1*(*q*, *r*), *e3*(*r*, *s*)),
(*e3*(*s*, *r*), *e2*(*r*, *p*), *e4*(*p*, *s*)).

Again, *p*, *q*, *r*, *s*, are understood as *formal* variables, which may refer to the same actual node.

Computation step

The algorithm endlessly iterates an elementary computation step, starting from the inital condition described above. The step is illustrated in Figure 3.

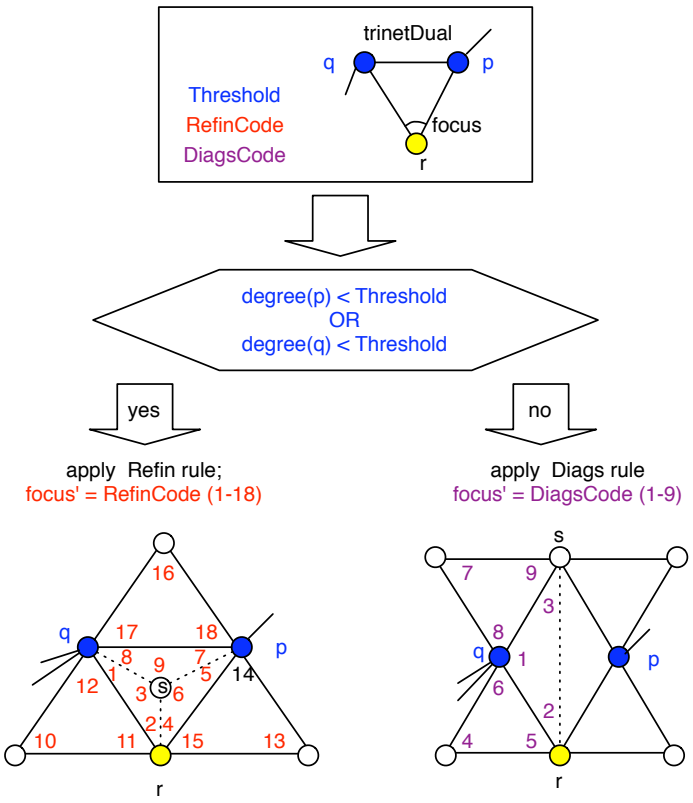

Figure 3 - The step of the algorithm

The step accepts the tuple (*trinetDual*, *focus*, *Threshold*, *RefinCode*, *DiagsCode*) and returns the tuple (*trinetDual*', *focus*', *Threshold*, *RefinCode*, *DiagsCode*), where:

- *trinetDual* is the current graph -- the dual of a trinet -- represented as a set of triangles.
- *focus* is an angle of a specified triangle in *trinetDual*, and represents the current location for control.
- *Threshold* is a constant natural number in the interval [3, ∞], The choice between rule *Refin* and *Diags* depends on the degrees of the two nodes, *p* and *q*, of the edge facing the *focus*: when the degree of *p* or *q* is lower than the *Threshold*, then rule *Refin* is applied, which increments by one the degree of both nodes; otherwise rule *Diags* is applied, which decrements their degrees by one.
- *RefinCode* and *DiagsCode* are constant parameters, ranging respectively in intervals [1, 18] and [1, 9]; they are used for choosing *focus*', the next focus, as shown in the lower part of Figure 3. In light of the symmetry of the initial graph, we can optimize the parameter space by dropping half of the 18 potential choices of the new focus, after rule *Diags* has been applied: we shall therefore consider only 18 * 9 = 162 pairs of values for these two parameters.
- *trinetDual*' and *focus*' are the updated values of these variables, used for iterating the computation step. *Threshold*, *RefinCode* and *DiagsCode* are constants, thus they are unchanged by the step.

Algorithm invariant

We now want to prove that the rewrite rules, as implemented, preserve the *sphericity*, as defined above, of the data structure they manipulate. For doing this we need to introduce a method for computing the degree of a node in a triangular, spherical embedding. If *n* is the number of occurrences of node *p* in a set of oriented triangles, then degree(*p*) = *n*/2, since each edge occurs twice in the structure (recall that a loop edge *e*(*p*, *p*) contributes 2 units to degree(*p*)). As an alternative, we may traverse all edges incident to *p* while rotating clockwise around *p*, as follows: pick from the set of triples an oriented edge *e*(*n1*, *p*) in which *p* appears as second node, and compute what we call the *cyclic star path*:

$$e(n1, p), f(p, n2), f(n2, p), g(p, n3), \ldots, e(p, n1)$$

in which two adjacent elements with different edge identifiers, e.g. *e*(*n1*, *p*) and *f*(*p*, *n2*), represent edges that share node *p* and appear in (cyclic) sequence in some triangle, while adjacent elements with the same edge identifier, e.g. *f*(*p*, *n2*), *f*(*n2*, *p*) represent two opposite traversals of the same edge, as found in two distinct triangles. It is easy to check that degree(*p*) is half the length of the star path around *p*.

As an example, consider the list of two oriented triangles representing the initial configuration introduced above. The cyclic star path for, say, node 1 is: *e3*(3, 1), *e1*(1, 2), *e1*(2, 1), *e3*(1, 3). This yields degree(1) = 2. The advantage of this techinique is that it allows us to possibly discover the degree of a node without scanning the whole list of triangles. We are now ready to prove an important invariant of the algorithm.

Proposition 1 (Algorithm invariant)

When applied to a spherical set of triangles, and when *Threshold* ≥ 3, the step of our algorithm produces another spherical set of triangles.

Proof

We must prove that when a set of triples satisfies sphericity conditions 1-3 above, then the set of triples obtained from it after one step also satisfies them. We shall refer to the edge and node identifiers appearing in the above rule implementations. We distinguish two cases.

Case 1: rule *Refin* is applied.

Each of the three triples created by this rule is an oriented triangle by construction, so that condition 1 is preserved. Furthermore these triangles: (i) re-introduce the instances *e1*(*q*, *r*), *e2*(*r*, *p*), *e3*(*p*, *q*) that were removed, so that each of these 'old' edges is shared precisely by a new and an old triangle, and (ii) collectively introduce two oriented edge occurrences, with opposite node orderings, of each new edge (*e4*, *e5*, *e6*), so that each new edge is shared by two of the new triangles; thus, condition 2 is also preserved. Finally, *v*, *e*, *t* are incremented, respectively, by 1, 3, 2, so that the value of *v* - *e* + *t* is unaffected and condition 3 preserved. The effect of *Refin* on a triangle involving one, two, or three distinct vertices is illustrated in Figure 4.

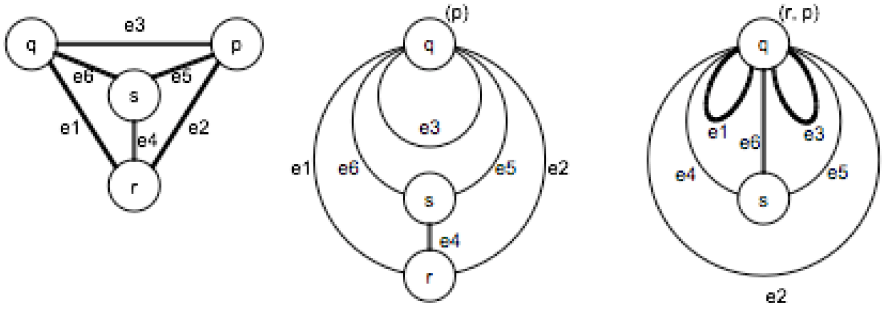

Figure 4 - Applying rule Refin to triangles with three vertices, two vertices, or one vertex

Case 2: rule *Diags* is applied.

Consider the two removed triangles. They share *at least* edge *e3*, but may share more; again, symbols *e1*, *e2*, *e4*, *e5* are understood as *formal* variables, some of which may assume the same *actual* value. Thus, we distinguish three cases, depending on the number of actual edges they share. In all these cases condition 3 is trivially guaranteed, since the counts of vertices, edges, and triples is left unchanged.

Case 2.1 - The two removed triangles share one edge.

Since *e1*, *e2*, *e4*, *e5* are all different, each of the two created triples is a triangle by construction, hence condition 1 is guaranteed. Condition 2 is guaranteed by the fact that oriented edge occurrences for *e1*, *e2*, *e4*, *e5* are only moved around by the rule, while occurrences *e3*(*p*, *q*) and *e3*(*q*, *p*) are replaced by occurrences *e3*(*r*, *s*) and *e3*(*s*, *r*), that still appear in different triangles.

Case 2.2 - The two removed triangles share two edges.

We distinguish two subcases.

Case 2.2.1. The two shared edges (one is *e3*) appear in the same order in the two removed triangles. Let us then assume, w.l.o.g., that *e1* = *e4* (the case *e2* = *e5* is symmetric), so that the two removed triangles can be written:

$(e1(q, r), e2(r, p), e3(p, q))$
$(e3(q, p), e1(r, q), e5(q, q))$,

where the second triple is obtained by reversing the order of nodes for *e1* and *e3*, and by letting the nodes for *e5* complete the triangulation. But for the second triple to be a correct triangle, it must also be $p = r$, so that the triangles can be rewritten as:

$(e1(q, p), e2(p, p), e3(p, q))$
$(e3(q, p), e1(p, q), e5(q, q))$.

These triangles are depicted in Figure 5 (left). By applying rule *Diags* to these two oriented triangles, we obtain the two triples:

$(e5(q, q), e1(q, p), e3(p, q))$
$(e3(q, p), e2(p, p), e1(p, q))$.

We have obtained two oriented triangles (condition 1), and it is trivial to verify that, after the replacement, condition 2 holds.

Case 2.2.2. The two shared edges (one is *e3*) appear in opposite order in the two removed triangles. Let us then assume, w.l.o.g., that $e1 = e5$ (the case $e2 = e4$ is symmetric). The two triangles to be removed would be:

$(e1(q, r), e2(r, p), e3(p, q))$
$(e3(q, p), e4(p, r), e1(r, q))$,

where the second triple is obtained by reversing the order of nodes for *e1* and *e3*, and by letting the nodes for *e4* complete the triangulation. Consider node *q*, which is shared by the two edges *e1* and *e3,* in turn shared by the two triangles: its *cyclic star path* is $(e3(p, q), e1(q, r), e1(r, q), e3(q, p))$, its length is 4, thus degree$(q) = 2$. This is in conflict with the assumption $Threshold \geq 3$: rule *Diags* could not be applied to edge *e3* since one of its nodes has degree lower than the threshold. The two triangles are depicted in Figure 5 (right).

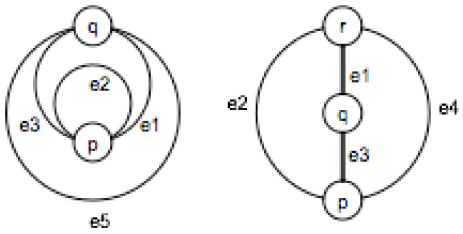

Figure 5 - Pairs of triangles for cases 2.2.1 and 2.2.2.

Case 2.3 - The two removed triangles share three edges.

The first removed triangle is $(e1(q, r), e2(r, p), e3(p, q))$, thus the second removed triangle must be composed by the three oriented edges $e1(r, q)$, $e2(p, r)$ and $e3(q, p)$, thus it can only be $(e1(r, q), e3(q, p), e2(p, r))$. In this case, the star path of any of the nodes has length 4, hence all nodes have degree 2. This is, again, in conflict with the assumption $Threshold \geq 3$. Q.E.D.

We have verified earlier that the set of triples in the initial configuration is spherical. In light of the above invariant, we now conclude, inductively, that all sets of triples produced by the algorithm are spherical, that is, they represent spherical embeddings of planar, triangular graphs.

Computation classes

We shall let $c[T, \{RC, DC\}, L]$ denote the $(L\text{-}1)$-step computation of our algorithm, starting from the initial condition above, with $Threshold = T$, $RefinCode = RC$ and $DiagsCode = DC$. (A pedantic but necessary clarification: usually a *step* is understood as a *pair* of consecutive states, hence a 1-step computation is two states long: the '*L*' in $c[T, \{RC, DC\}, L]$ refers to the *Length* of the computation, intended as the number of states it includes.) We shall also use a convenient notation for representing subsets of the computation space. For example, $C[Threshold = 3]$ shall denote the set $\{c[3, \{RC, DC\}, S] \mid RC \in [1, 18], DC \in [1, 9], S \geq 1\}$, that is, the family of all computations with $Threshold = 3$, of any length. Since we impose $Threshold \geq 3$, the first computation step inevitably applies rule *Refin*, and produces the tetrahedron graph shown in Figure 1 (b), Then computations start to differentiate, depending on the parameter settings.

3. Visual indicators for planar trinet computations

A computation can be defined as a sequence of states. The state of our algorithm is essentially formed by the pair of variables (*trinetDual*, *focus*), that represent a planar graph and the location of control in it. We are interested in the emergent properties of planar trinets, with the idea that they might eventually capture properties of physical space. However, it may be hard to visually detect emergent properties when directly using graphs, for the following reasons: (i) a trinet or trinet dual may soon become a complex structure, and a sequence of thousands of them can hardly be inspected at a glance, as opposed to what happens, for example, with the computations of elementary cellular automata; (ii) there exist many alternative methods to draw graphs on the plane, such as using a predefined arrangement of the nodes (e.g. circular) or applying attractive or repulsive forces to nodes, and the emergence and detectability of patterns is largely dependent on the method.

One can of course look just at the final graph, either in dual or in primal form, and we shall do it too. However, emergent properties are better detected when looking at the whole computation. Thus, in our investigation we shall be much interested in the fluctuations of the other state variable -- the *focus*. This variable captures only a tiny fraction of state information, but this is indeed an advantage, since we can easily plot a whole computations as a compact, readily inspected diagram. We have defined the *focus* as the angle {*e1*, *e2*} between two edges of triangle {*e1*, *e2*, *e3*}. For further simplification, we shall simply monitor the *edge* opposite to this angle, namely *e3*. Note that this is the edge whose vertices *p* and *q* are tested at every step: we call it *the current edge*. Looking at *e3* rather than at pair {*e1*, *e2*} introduces further ambiguity, or abstraction, since *e3* identifies *two* possible foci. And yet, the sequence of current edge identifiers turns out to be a useful indicator. New edges are created in the trinet dual, three at a time, only by the *Refin* rule, and are assigned progressive natural numbers; plotting the sequence of current edge numbers reveals the extent to which the control point can revisit and update old parts of the graph, and whether some regions are definitively abandoned. For this reason we call these numeric sequences, and their plots, *revisit indicators*.

For illustrating the idea above, and for comparison with some of the revisit indicators we shall discuss later, consider the two plots in Figure 6. These depict the revisit indicators for two extremely simple and regular graph growth patterns; the relevant revisited elements are now the nodes, which are numbered sequentially as they are created. In the first case the algorithm maintains a *linear* topology, and creates a new node only after having sequentially scanned the current list {1, 2, ..., *n*} of nodes, in both directions, up and down. The second case is similar, except that nodes are arranged in a growing circle, and a new node is added after a *circular*, single-scan visit of the graph.

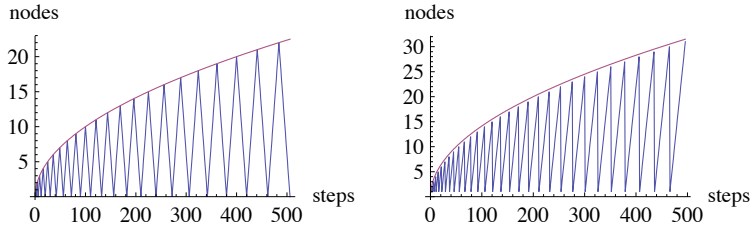

Figure 6 - Revisit indicators for graphs with linear (left) and circular (right) topology

An easy calculation shows that the node-growth functions, also plotted in the diagrams, are n = Sqrt[s] for the linear graph, and n = Sqrt[2s] for the circular graph. More generally, a growth rate n = Sqrt[2s/k] corresponds to a 'grow-and-revisit' algorithm that, in the interval between the creation of nodes n-1 and n, takes k*n revisit steps.

A rather obvious visual complexity indicator, even simpler than the revisit indicator, consists in plotting the number of nodes in the trinet dual as a function of the algorithm steps. We call this the *dual node count* indicator, with the attribute 'dual' often omitted. Recall that these nodes represent trinet faces. This indicator is a monotonic, non-decreasing function, since rule *Refin* adds one node to the trinet dual (two nodes to the original trinet), and rule *Diags* leaves the node count unaffected. In the sequel we mainly refer to the revisit indicator, since it confirms but also refines in interesting ways the classification induced by pure node counting.

4. Threshold 3: 1-D trinets, simple oscillators and trees

Figure 7 shows the revisit indicator for all the computations of Length 500, with Threshold 3 -- a set of 18 * 9 = 162 elements that we denote *C*[*Threshold* = 3, *Length* = 500], and Figure 8 shows the corresponding final trinets. The numbers appearing at the left of each small diagram represent the highest current edge identifier used in the computation: this number cannot exceed 3**Length*, since the initial trinet dual has three edges, and each step can at most contribute three new edges.

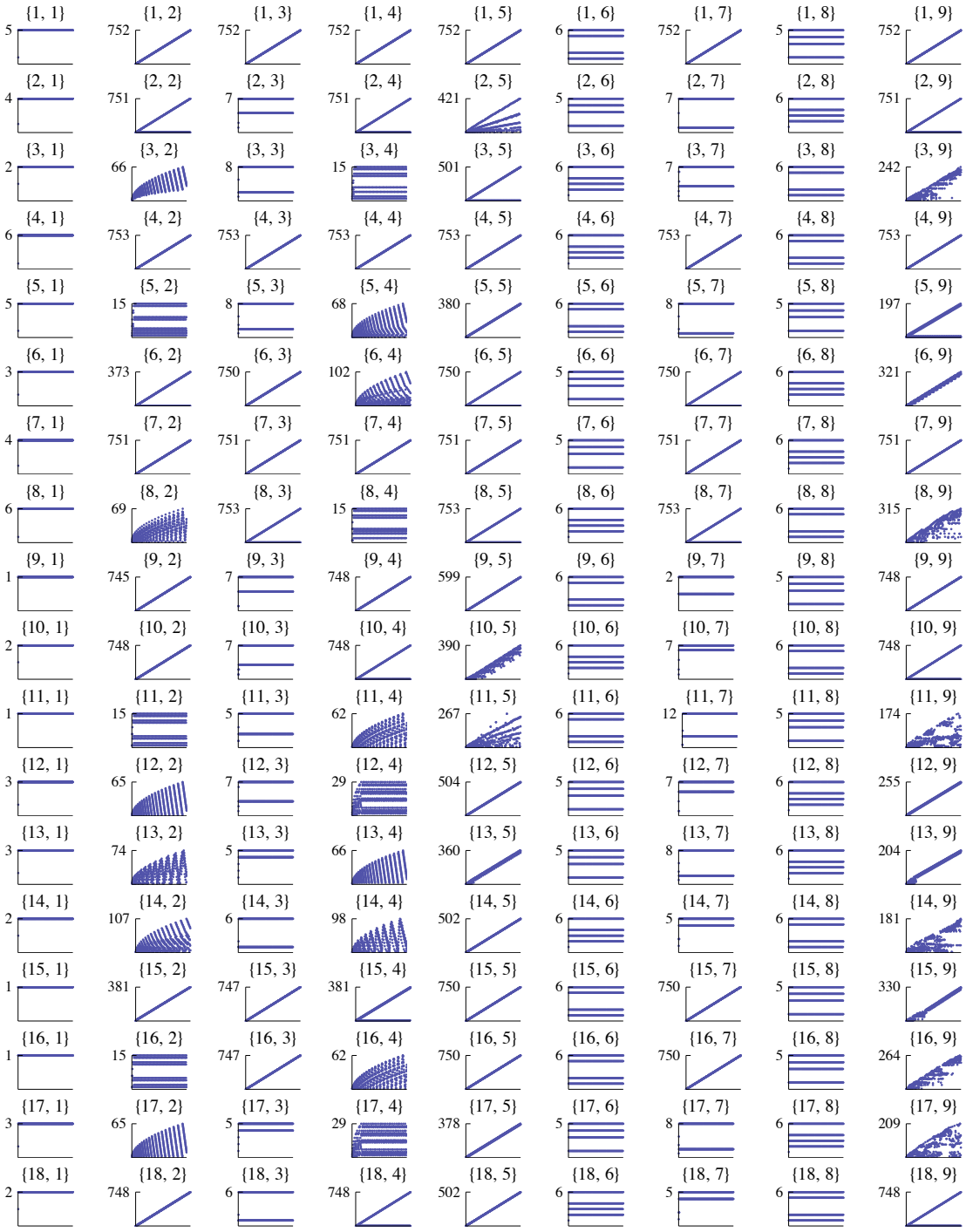

Figure 7 - Revisit indicators for all threshold-3, length-500 computations

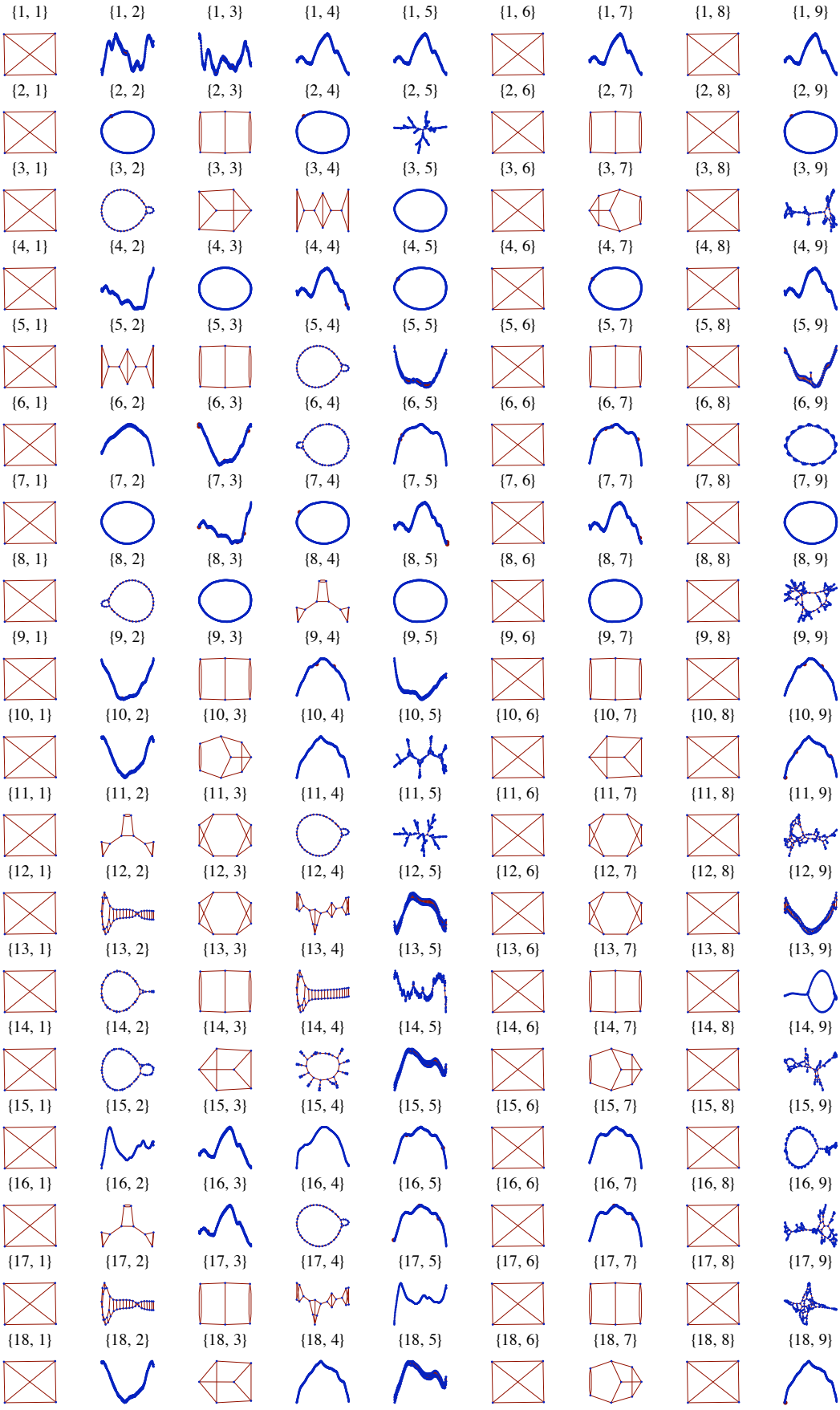

Figure 8 - Final trinets for all threshold-3, length-500 computations

Constant revisit indicator: periodic trinet sequences

The simplest type of computation corresponds to a constant revisit indicator. An example is computation *c*[3, {1, 1}, _] (of unspecified length), in which, after one step, edge 5 becomes the current edge forever, and *Diags* the only applied rule. In this case, the sequence of trinets has period 2 and oscillates between the tetrahedron and a square-like graph with two double edges; the tetrahedron happens to be the final trinet of the length-500 computation, as shown in the corresponding entry of Figure 8. Many similar cases of constant plots, involving a finite number of distinct current edge identifiers occur in this computation class, as well as in subsequent ones; they all correspond to periodic sequences of bounded-size trinets, and to graphs with very few nodes and edges in Figure 8. For example, in computation *c*[3, {2, 3}, _], the current edge oscillates between 7 and 4, and the sequence of trinets has period 10 (or period 5, if node identities are ignored). Note that in this class also the node count indicator must be a constant function.

Linear revisit indicator: regular, 1-D, growing trinets

The next simple case, quite common too, is that of revisit indicators with linear growth. An example is computation *c*[3, {1, 2}, _], in which the sequence of current edge identifiers gives a regular numeric sequence where every third natural number is skipped. The corresponding graph is the simplest 1-D, ladder-shaped trinet. Four consecutive trinets from this computation are shown in Figure 9. Node numbers help understanding the growth mechanism.

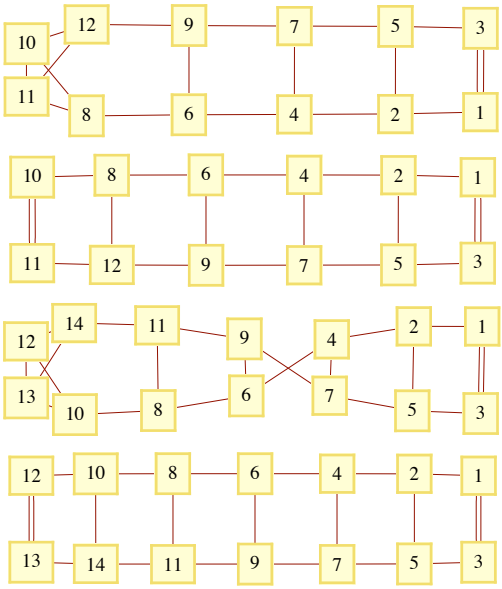

Figure 9 - Four consecutive trinets from computation c[3, {1, 2}, _]

Other simple 1-D patterns are observed. For example computations *c*[3, {12, 5}, 60] and *c*[3, {13, 5}, 130] yield the trinets shown in Figure 10, which grow linearly and regularly: the details of these structures are lost in the linear thumbnail diagrams of Figure 8.

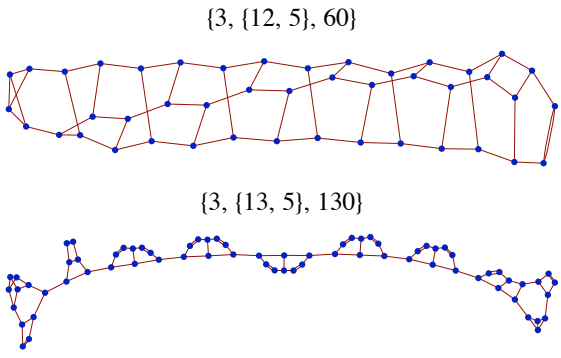

Figure 10 - 1-D final trinets of c[3, {12, 5}, 60] and c[3, {13, 5}, 130]

In all linear cases above, the growth takes place at one extreme of the graph. A slightly different growth pattern is

achieved by computation c[3, {5, 9}, _], whose revisit indicator is illustrated in more detail, with the corresponding 1-D trinet, in Figure 11: in this case the growth takes place in the central part of the trinet.

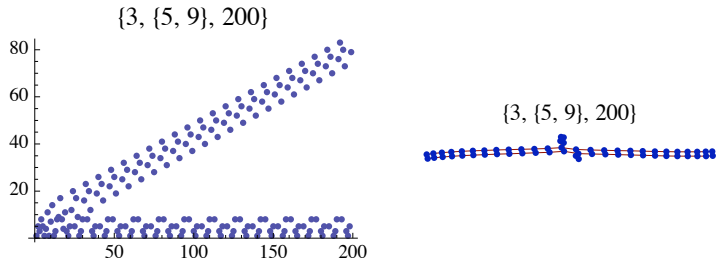

Figure 11 - Revisit indicator and final 1-D trinet growing from its center

The second 1-D pattern found in this family of computations is the circle: four examples are shown in Figure 12. In the last case, the circle is formed after a relatively large initial transient phase; the portion of the trinet created in this phase is then permanently abandoned, and the growth takes place at the extreme of the circle opposite to it, as suggested by the picture.

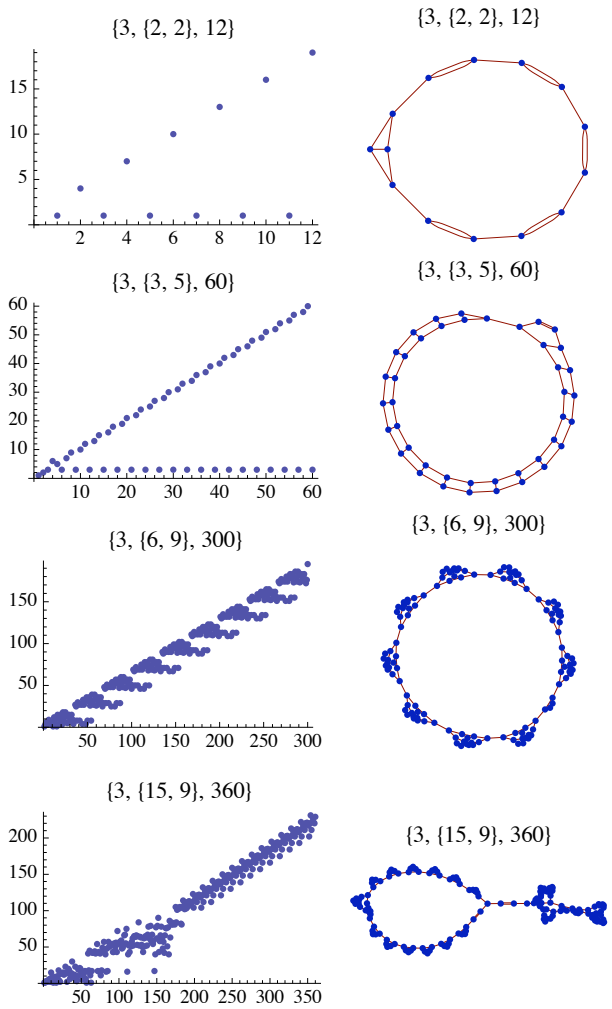

Figure 12 - Revisit indicator and final, circular 1-D trinet for three computations

The simple linear and circular 1-D structures appear combined in *c*[3, {13, 9}, _] (Figure 13); in this case the active region is at their junction, and they grow at the same speed, providing a stable overall shape.

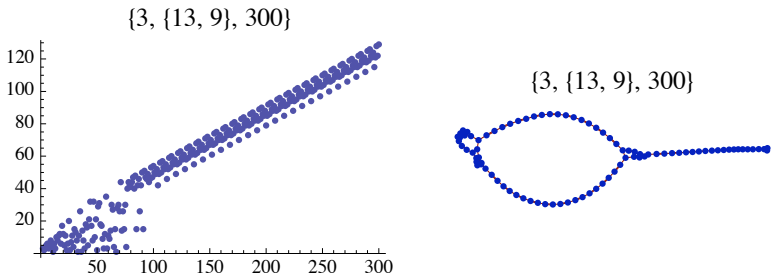

Figure 13 - Revisit indicator and final 1-D trinet for c[3, {13, 9}, 300]

All the revisit diagrams of this group eventually exhibit the same linear growth; the node count grows linearly with the computation steps, all parts of the growing trinet are eventually abandoned, except, possibly, for a finite part, and the trinet is one-dimensional.

Nested revisit indicator: oscillating, segment-circle and circle-circle 1-D trinets

Twelve examples of *nested* revisit indicator are found in Figure 7: six are in column 2, and have (*RefinCode, DiagsCode*) pairs (3, 2), (8, 2), (12, 2), (13, 2), (14, 2), (17, 2), and six in column 4, with codes (5, 4), (6, 4), (11, 4), (13, 4), (14, 4), (16, 4). The final trinets in Figure 8 fail to reveal the substantial difference between these computations and those in the previous group, but the revisit indicators are more informative; in particular, these diagrams indicate that the trinet growth process sweeps an increasingly large portion of the net, by actually sampling all parts of it, except for case (3, 2) in which a slowly growing region is permanently abandoned. Two distinct types of dynamics emerge from these 12 computations; they are described below.

Segment-circle. The first growth pattern is exhibited by five computations, with codes (12, 2), (13, 2), (17, 2), (13, 4), (14, 4). We call this pattern 'segment-circle' because the trinet is formed by connecting a linear segment and a circle; this is similar to the trinet shown in Figure 13, except that both parts now grow and shrink, with opposite phase, so that the trinet oscillates smoothly between a purely circular and a purely linear form. Activity always takes place at the junction between the two structures. As observed in the previous cases, the micro-structure of the segment and the circle is based on a variety of different, simple building blocks.

As an example, Figure 14 illustrates computation *c*[3, {14, 4}, 2300].

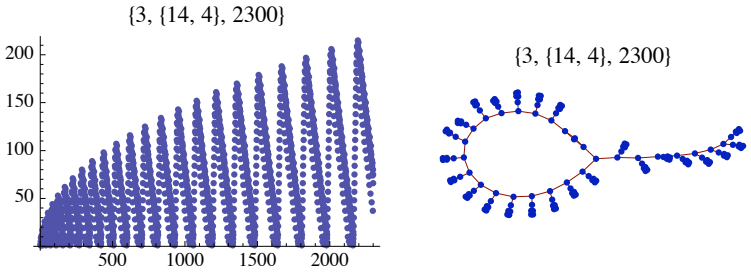

Figure 14 - Nested computation c[3, {14, 4}, 2300].

Note the similarity between the revisit diagram of Figure 14 and the one on the left in Figure 6. For a better illustration of the dynamics of these trinets, a subsequence of several steps from computation *c*[3, (13, 2), _] is shown in Figure 15.

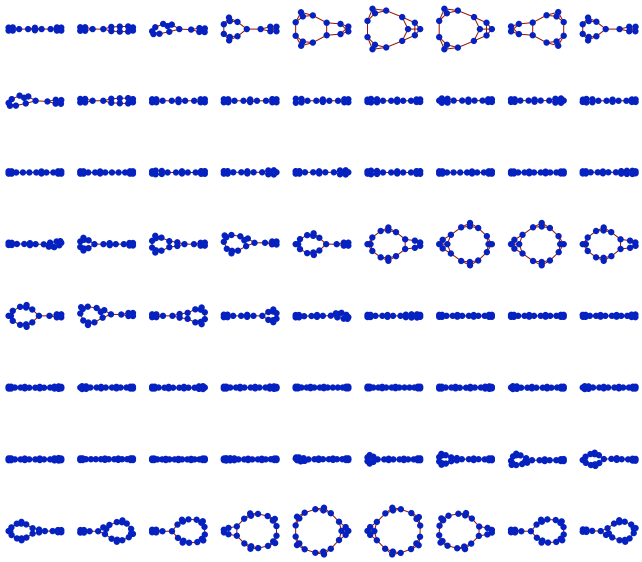

Figure 15 - Steps of computation c[3, {13, 2}, _], showing circular and linear components that grow and shrink

Circle-circle. The second growth pattern is exhibited by seven computations, with codes (3, 2), (8, 2), (14, 2), (5, 4), (6, 4), (11, 4), (16, 4). We call the pattern 'circle-circle' because the trinet is formed by connecting two circles, that grow and shrink with opposite phase. Again, the growth always takes place at the junction between the two structures, and various types of simple building blocks are observed. Several steps of computation *c*[3, (6, 4), _] are shown in Figure 16, where the micro-structure is the same of Figure 15.

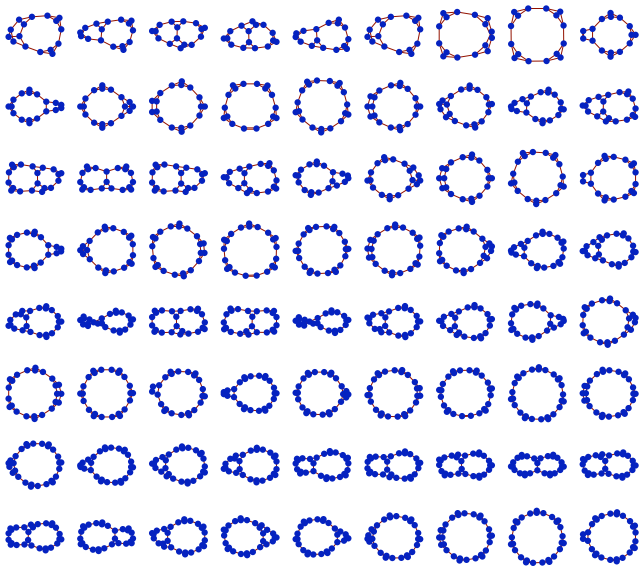

Figure 16 - A few steps of computation c[3, {6, 4}, _] showing a growing and a shrinking circle

The twelve computations with nested revisit indicator discussed above are also directly identified by inspecting their *node count* indicators: out of the 162 threshold-3 elements they are precisely those that exhibit a regular staircase shape with a sub-linear growth. A closer investigation of these data has revealed the facts described below.

The 8 computations with *RefinCode-DiagsCode* pairs (3, 2), (8, 2), (12, 2), (17, 2), (5, 4), (11, 4), (13, 4), (16, 4) exhibit *identical* node growth functions, although they yield trinets of different types: codes (3, 2), (8, 2), (5, 4), (11, 4), (16, 4) yield the *same circle-circle* trinet, and codes (12, 2), (17, 2), (13, 4) yield the *same segment-circle* trinet. Their node growth function is matched with excellent precision by function $f(x) = 3 + \text{Sqrt}[x]$, as shown in Figure 17. Recall that the initial trinet dual has indeed three nodes.

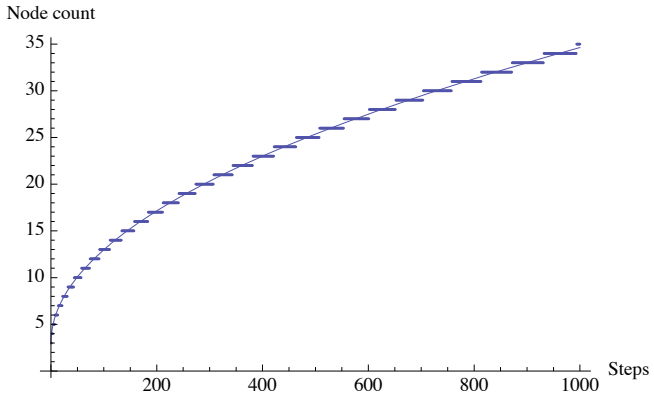

Figure 17 - Node growth for eight threshold-3 nested computations and function 3 + Sqrt(x)

Of the remaining 4 nested cases, those with code pairs (13, 2) and (14, 4) are of *segment-circle* type, while (14, 2) and (6, 4) are of *circle-circle* type. All four node growth functions are different, but they all approximate quite closely the O(*n*^1/2) growth of the previous group. Indeed, by using *Mathematica* function `FindFit` over computations of length 10,000, and by using parametric function *f(x) = a + b*x^c*, in all four cases a *c* exponent in the close proximity of 1/2 was found.

'Radial' revisit indicator: tree-like irregular trinets

Two computations in class *C*[*Threshold* = 3] exhibit revisit indicators that are slightly perturbed versions of a very regular pattern consisting of potentially infinite straight lines ('rays') emanating from the origin: these have code pairs (2, 5) and (11, 5). Their corresponding trinets are tree-like *irregular* graphs, and the active point on them also moves quite irregularly, visiting every part infinitely often. The revisit indicator and final trinet for computation *c*[3, (2, 5), 2000] are shown in Figure 18. We had found a cleaner (that is, not perturbed) version of this radial revisit indicator by using another trinet algorithm, as described in [B2007a], and we shall find it again with the present algorithm, later in the paper.

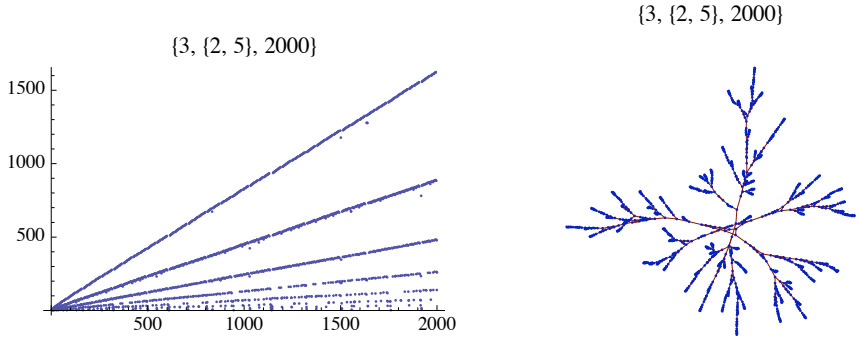

Figure 18 - An 'infinite rays' revisit indicator and the corresponding tree-like trinet.

These two cases also illustrate the usefulness of our revisit indicator: the pure node count indicator for them is basically linear, and would not be as effective as the infinite-radii pattern in discriminating them from the many other computations with linear node count.

Revisit indicators with long random transients: complex trinets with 'highways'

By inspecting Figure 7, six computations in class *C*[*Threshold* = 3] exhibit a high degree of apparent randomness in their revisit indicator. They are all in column 9, and have code pairs (3, 9), (8, 9), (11, 9), (14, 9), (16, 9), (17, 9). In fact, by looking at longer computations, *all* of them eventually stabilize to a regular growth pattern, called 'highway' in analogy with the phenomenon observed in some two dimensional Turing Machines. (The node count indicators in all these cases appear as roughly linear, even in the transient phase preceding the highway.)

For example, Figure 19 shows the periodic revisit indicator and periodic trinet for computation c[3, {3, 9}, 2000].

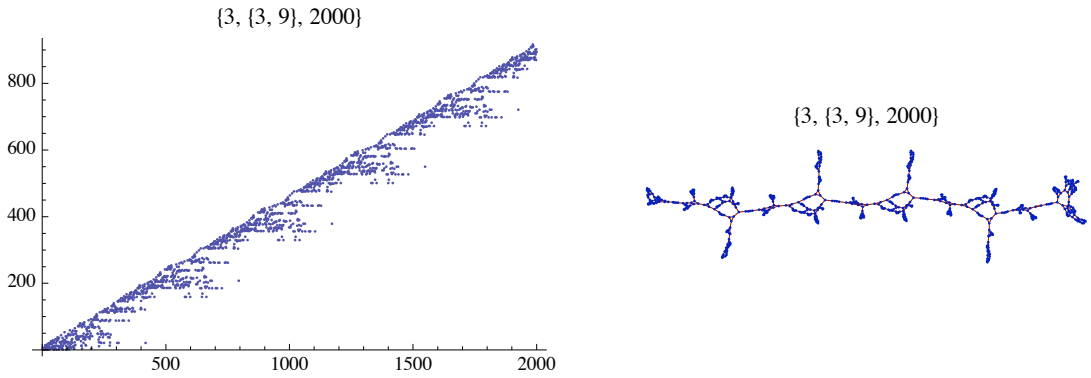

Figure 19 - A periodic computation with short transient and long period.

As another example, Figure 20 illustrates computation *c*[3, {11, 9}, 12000]. More than 8000 steps are necessary for stabilizing the growth, with settles in what we have called a *segment-circle* pattern, with the active part at the junction of the two components. In fact, in spite of the possibly very long initial transient and period, these computations are not qualitatively different from those with 'linear revisit indicators' discussed at the beginning of this section.

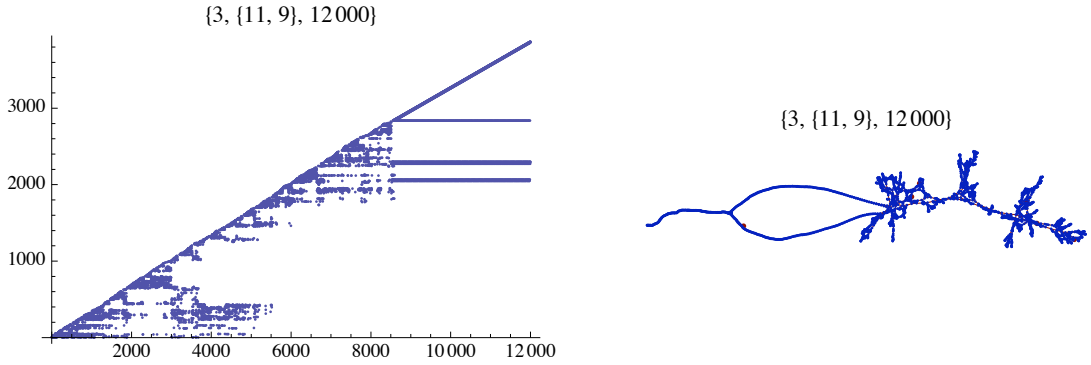

Figure 20 - A computation yielding a trinet which eventually settles to a segment-circle pattern with linear growth

Finally, we have investigated computation *c*[3, {14, 9}, _]. Running it for 160,000 steps allowed us to detect the periodicity of its revisit indicator, with a period of over 11,000 steps.

5. Threshold 4: nested trinets and uniform randomness

Similar to the previous section, Figure 21 shows the revisit indicator for all the 162 computations in class *C*[*Threshold* = 4, *Length* = 500], and Figure 22 shows the corresponding final trinets.

Figure 21 - Revisit indicators for all threshold-4, length-500 computations

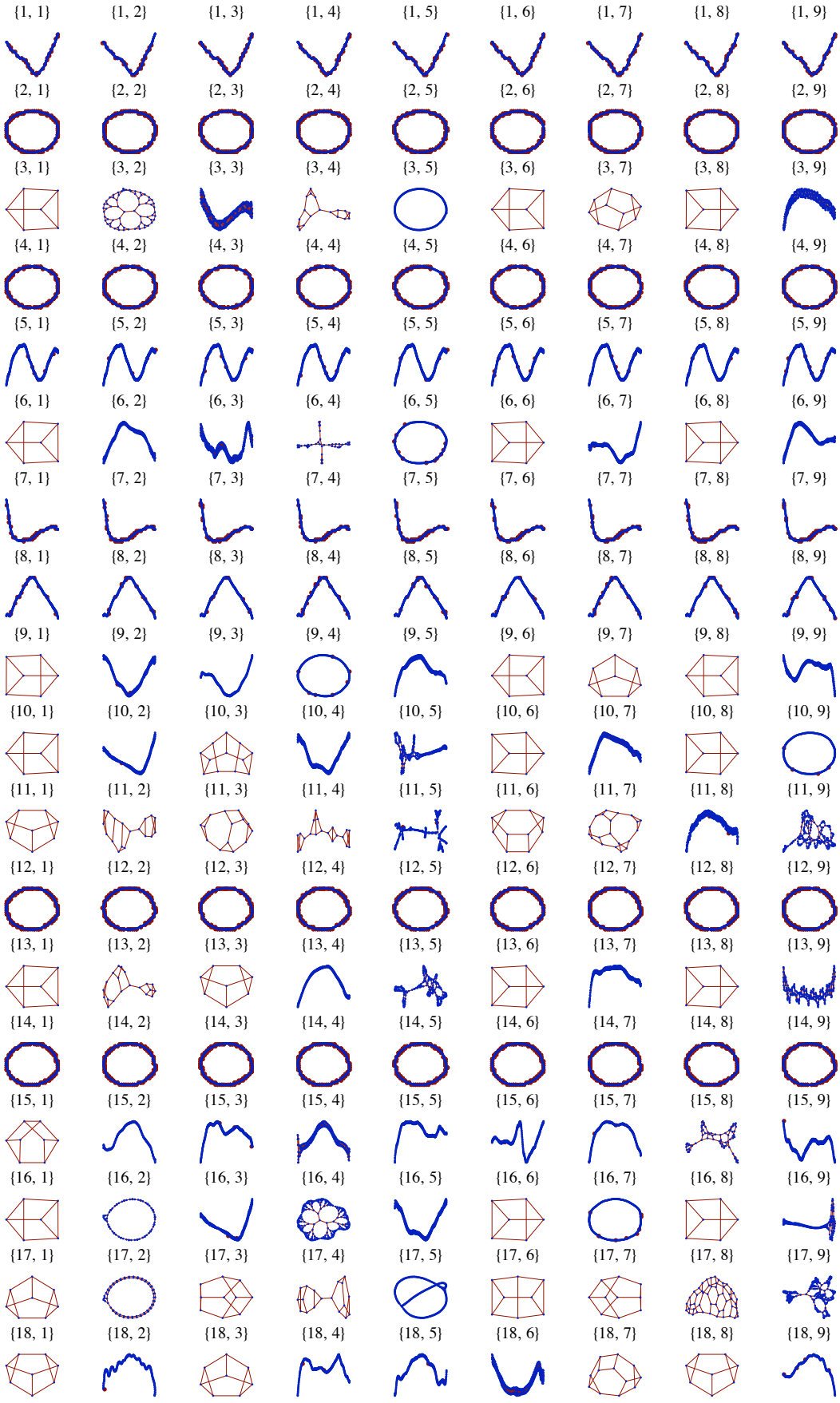

Figure 22 - Final trinets for all threshold-4, length-500computations

By inspecting the two figures it is immediately clear that the large majority of these computations exhibits emergent features that are qualitatively the same we have already observed in the previous class. Let us shortly overview these cases, before moving to the novel, most interesting ones.

We still have periodic sequences of bounded trinets, and 1-D trinets that grow linearly and without bound, as segments or circles. Interestingly, among the latter, computation *c*[4, {17, 5}, _] provides a trinet whose uniformly growing structure appears similar to a circle with its diameter, so that its macrostructure reproduces the initial trinet -- a two-node graph with three double edges. We find also two computations, namely *c*[4, {16, 2}, _] and *c*[4, {17, 2}, _], with nested revisit indicators and corresponding trinets that oscillate while growing, according to the already observed circle-circle pattern. As observed with *threshold* 3, we find one computation with noisy, radial revisit indicator, namely *c*[4, {11, 5}, _], which yields an irregular tree-like trinet, similar to that obtained with computation *c*[3, {11, 5}, _]. Finally, computations *c*[4, {11, 9}, 500], *c*[4, {13, 5}, 500], *c*[4, {17, 9}, 500], whose revisit indicators do not manifest any regularity in 500 steps, all eventually settle to a 1-D, periodic, unbounded trinet. We are left with three novel and quite interesting cases, that are discussed below.

c[4, {16, 4, _]: nested binary tree trinet with circular boundary

This computation exhibits a cleaner version of the radial revisit indicator, and the trinet has now a regular and nested structure, shown again in Figure 23: it is formed by a binary tree with trivalent root, with the addition of edges interconnecting adjacent leaves in a circle. The same trinet was obtained also by the algorithm introduced in [B2007a].

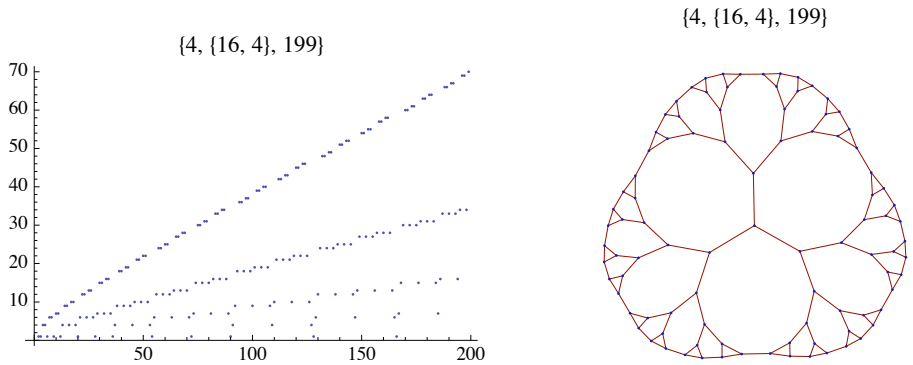

Figure 23 - Revisit indicator and final trinet of computation c[4, {16, 4}, 199]

c[4, {3, 2}, _]: nested binary tree with oscillating, circle-circle pattern

The thumbnail for the trinet of this computation, shown in Figure 22, misleadingly suggests a similarity with the previous computation *c*[4, {16, 4}, _]. However, the nested revisit indicator (Figure 21) reveals rather different dynamics. In fact, the trinet oscillates while growing, and resembles the already discussed circle-circle pattern (see Figure 16), except that now a nested structure is involved, rather than a simple circle.

c[4, {17, 8}, _]: randomized square root growth

This computation is perhaps the most surprising we have found; the only similar case is c[5, {9, 8}, _], to be discussed later. Figure 24 shows the revisit indicator, the final trinet, and the node count as a function of the algorithm steps, for c[4, {17, 8}, 6000]. Recall that the we count the number of nodes in the *trinet dual*, corresponding to the number of *faces* in the original trinet; in figure, this function is then matched against function f[*steps*] = 3 + Sqrt[2**steps*], which exactly characterizes also the regular, circular 'grow-and-revisit' algorithm introduced in Section 3: statistically, the growth process is such that, between two new trinet face creations, a number of re-visits is performed that equals the current number of faces.

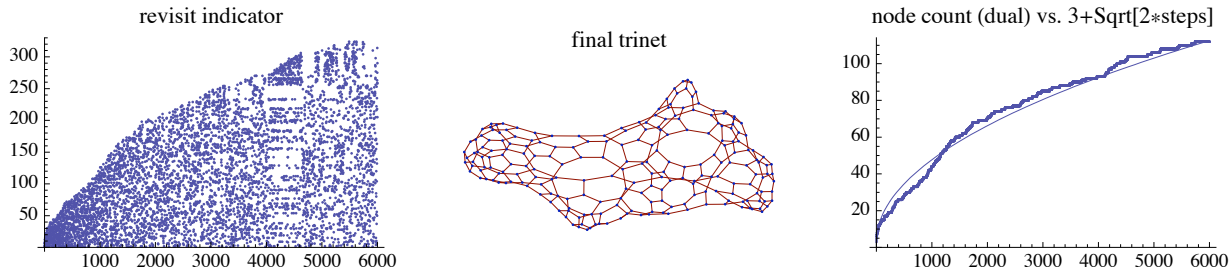

Figure 24 - Revisit indicator, final trinet, and node growth for c[4, {17, 8}, 6000]

Figure 25 shows similar data for a computation of length 100,000.

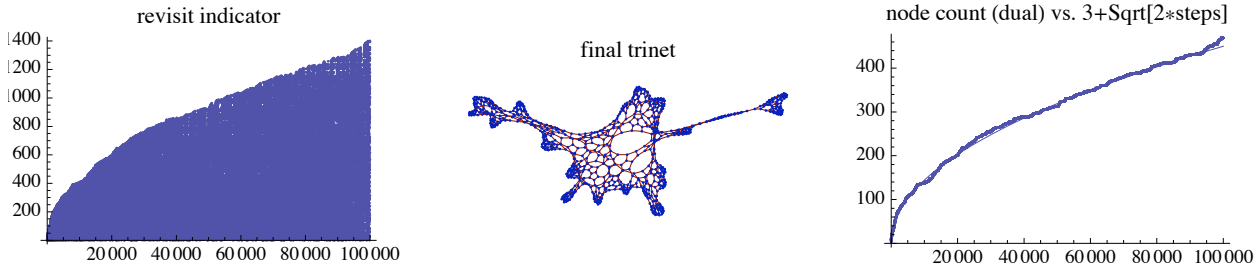

Figure 25 -Revisit indicator, final trinet, and node growth for c[4, {17, 8}, 100,000]

This ability of this computation to revisit its past uniformly, densely, and indefinitely, while exhibiting random-like dynamics, is indeed quite remarkable.

6. Threshold 5: second case of uniform randomness

Most of what emerges in this class has been observed before. For any value of *RefinCode* in range [1, 8], and for values 12 and 14, the computation is independent from the value of *DiagsCode* and yields a linear or circular, 1-D trinet. All other cases are illustrated in Figures 26 and 27.

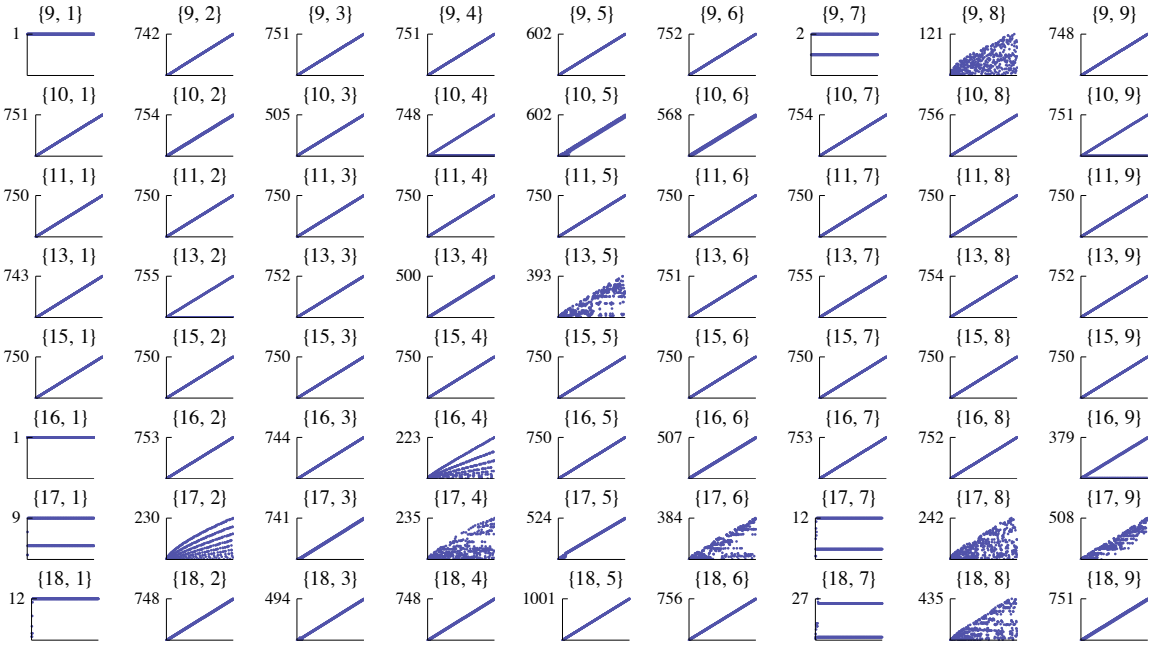

Figure 26 - Revisit indicators for all threshold-5, length-500 computations

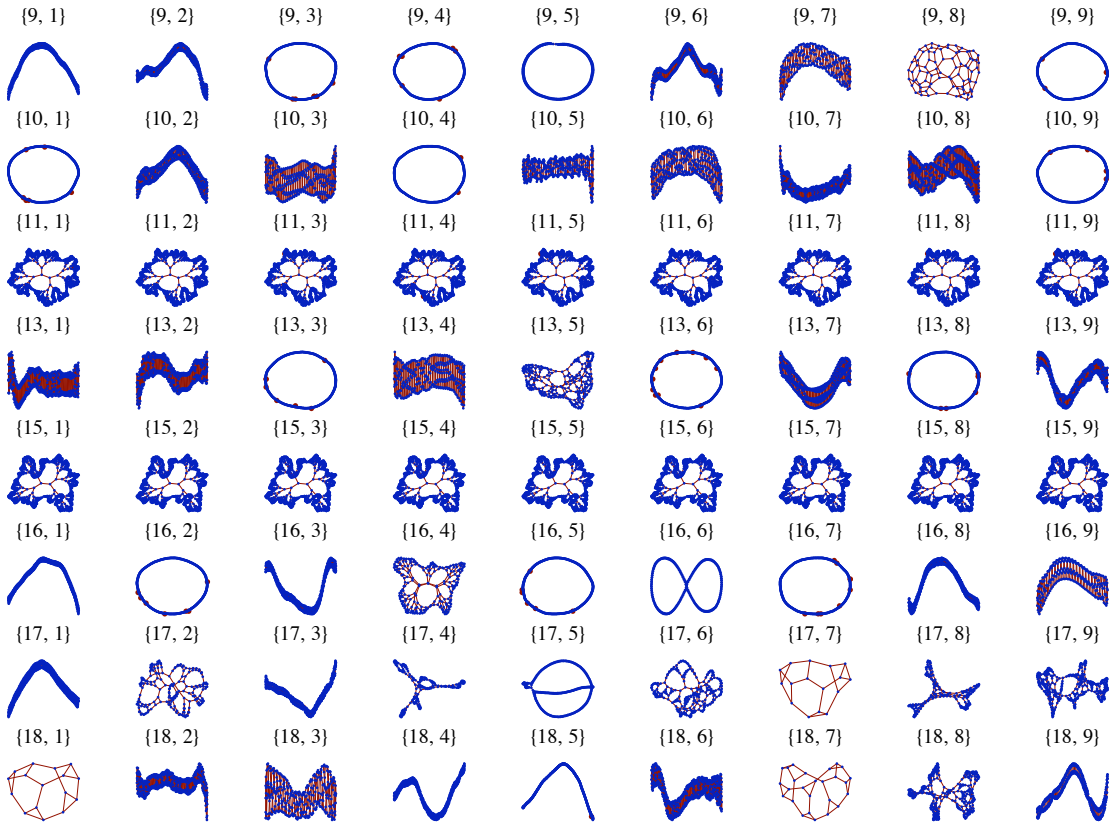

Figure 27 - Final trinets for all threshold-5, length-500 computations

RefinCodes 11 and 15, again regardless of the *DiagsCode*, yield plain, linear revisit indicators, but the corresponding trinets are now nested, a combination that was not observed in previous classes. The same trinet structure can therefore be obtained by different revisit policies, and, correspondingly, in a different number of steps. Figure 28 shows three different computations with different revisit indicators that produce the same trinet in, respectively, 22, 22 and 85 steps. Nodes have been labelled for showing the different orders in which the graphs are created.

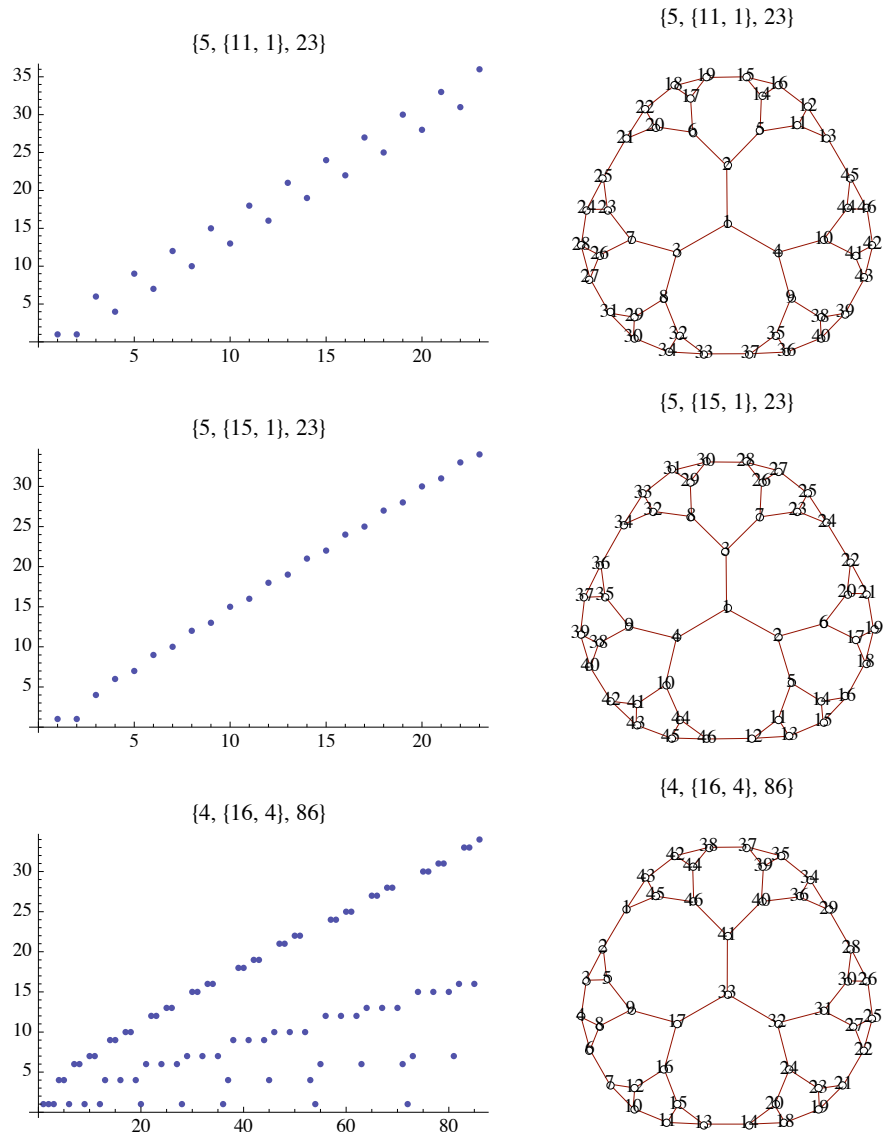

Figure 28 - Three computations yielding the same trinet

No nested revisit indicator of the types seen before (e.g. that of computation *c*[3, {3, 2}, _] or *c*[3, {5, 4}, _] is found in this class. But we do find an unperturbed radial revisit indicator for c[5, {16, 4}, _] which corresponds to a nested trinet.

A peculiar tree-like trinet is obtained for *c*[5, {17, 2}, _]; the revisit indicator and the trinet are shown in Figure 29. (In this case the sublinear, node growth function for the dual graph is approximated by *f*(*steps*) = 5.31 + 1.57**steps*^0.65.)

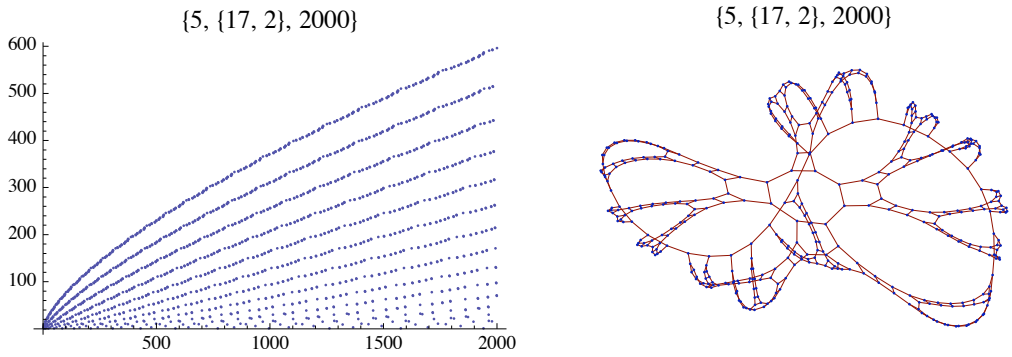

Figure 29 - Revisit indicator and final trinet of c[5, {17, 2}, 2000]

As observed for *threshold*-4 computations, all the 6 revisit indicator thumbnails that appear as random in Figure 26, corresponding to code pairs (13, 5), (17, 4), (17, 6), (17, 8), (17, 9), (18, 8), end up settling to a regular behavior. But case (17, 8) is the only one for which the revisit indicator stabilizes to a square root growth pattern. The last, most interesting case, is discussed below.

c[5, {9, 8}, _]: second case of randomized square root growth

This is the only other example, similar to case *c*[4, {17, 8}, _], of a computation which exhibits a dense, random but uniform revisit indicator, with node growth well approximated by a square root function. Figures 30 and 31 show the revisit indicator, the final trinet, and the node growth function, with approximating functions, for computations of lengths 6000 and 100,000, respectively.

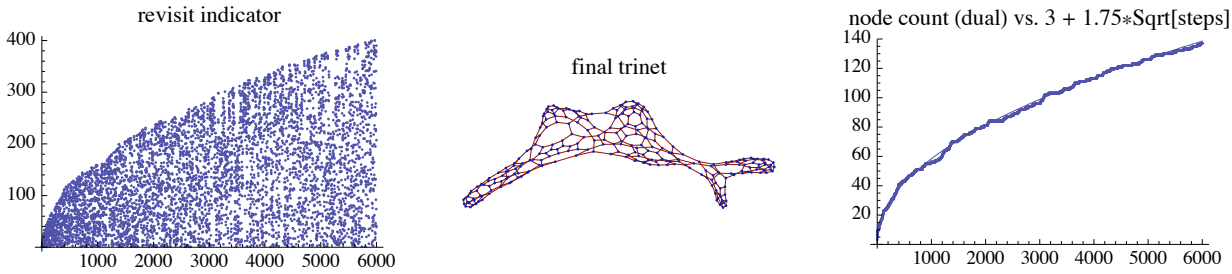

Figure 30 - Revisit indicator, final trinet, and node growth for c[5, {9, 8}, 6000]

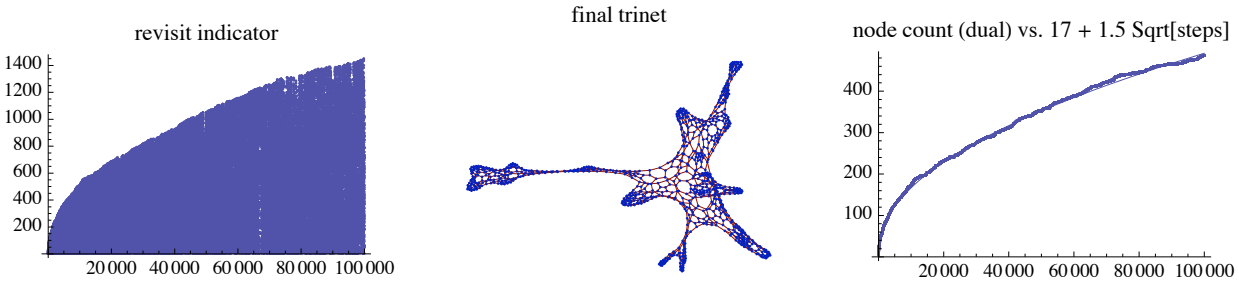

Figure 31 - Revisit indicator, final trinet, and node growth for c[5, {9, 8},100,000]

7. Threshold 6: regular and irregular 2-D grids

For *threshold* 6 and higher, the computations for *RefinCode* values in range [1, 8], and values 11, 12, 14, 15, appear exactly the same as those obtained for threshold 5, and are (individually) independent from *DiagsCode* values. The six interesting values left for *RefinCode* are illustrated in Figures 32 and 33.

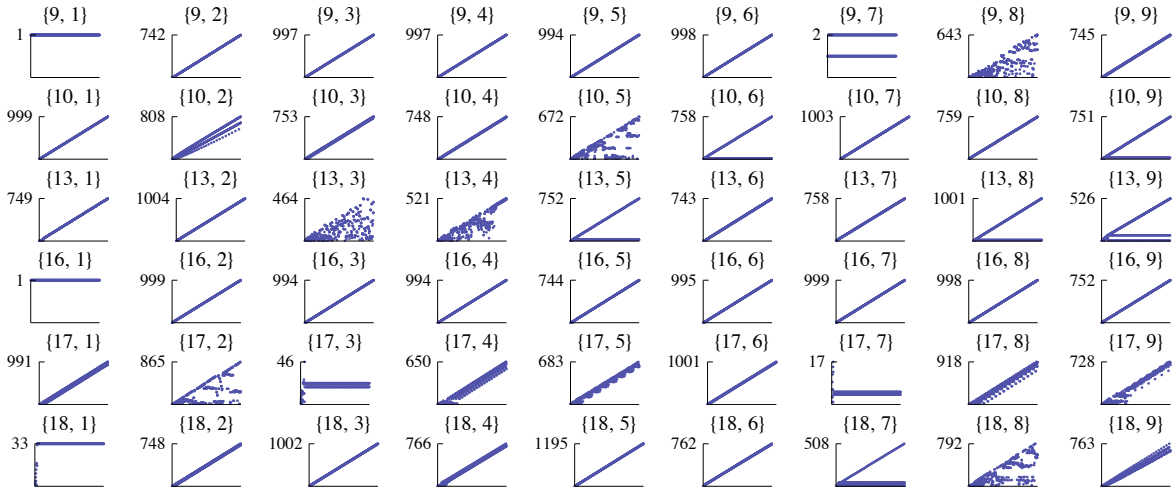

Figure 32 - Revisit indicators for all threshold-3, length-500 computations with RefinCodes 9-18

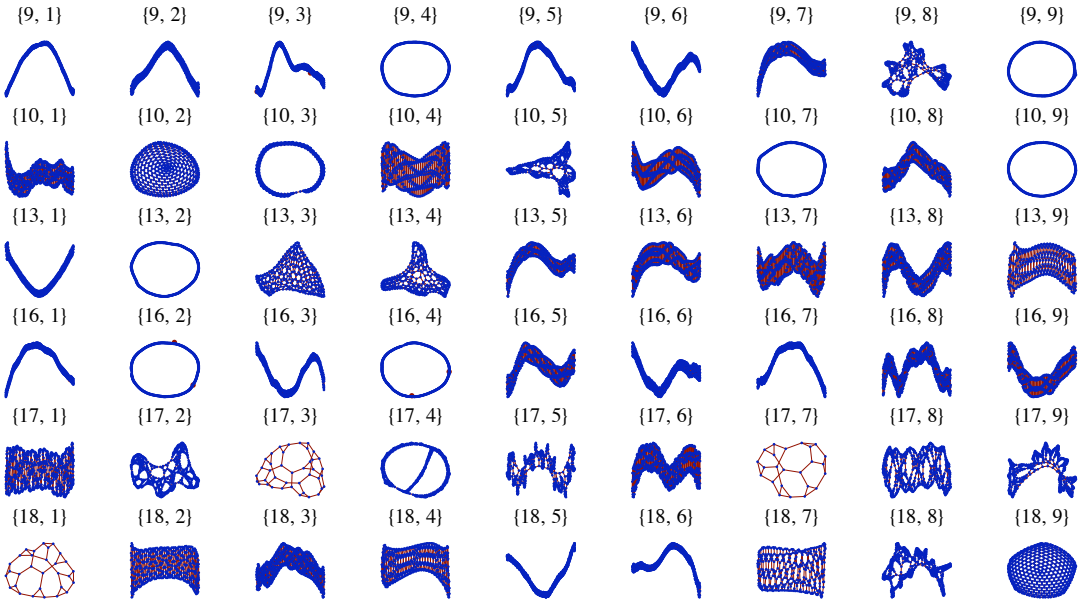

Figure 33 - Final trinets for all threshold-3, length-500 computations with RefinCodes 9-18

Most of the computations in *C*[*Threshold* = 6] exhibit the already discussed emergent features. For example, computations *c*[6, {9, 8}, _], *c*[6, {10, 5}, _], *c*[6, {13, 4}, _], *c*[6, {17, 2}, _], *c*[6, {18, 8}, _], with irregular thumbnails, all end up settling to regular behavior, possibly with long initial transients. For example, *c*[6, {9, 8}, _] takes about 35,000 steps to stabilize.

However, three novel cases are observed, that produce, for the first time in our investigation, 2-D trinets. They deserve special attention.

c[6, {10, 2}, _]: hexagonal grid with three central pentagons

The trinet produced by this computation is a 2-D, hexagonal grid that develops around a nucleus of three pentagons; this

is shown in Figure 34. The active point is always at the border of the graph. The fact that this border grows with the graph itself explains the peculiar shape of the revisit indicator, with three slightly divergent radii, and a growing part of the net being eventually abandoned.

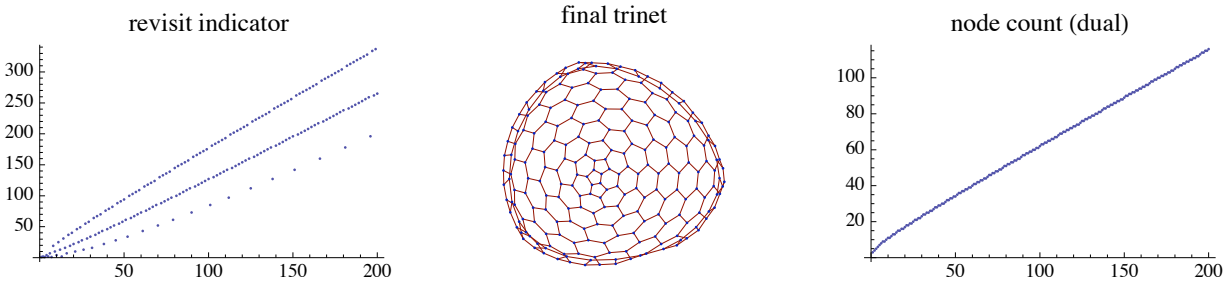

Figure 34 - Revisit indicator, hexagonal grid, and dual node count for c[6, (10, 2), 200]

For comparison with the other two 2-D trinet computations it is useful to analyze the distribution of polygon sizes. For computation *c*[6, (10, 2), 3000] the following distribution is observed in the final trinet:

{{3, 3},{4, 1},{5, 93},{6, 1453},{98, 1}},

where {*x*, *y*} indicates that there are *y* faces with *x* edges. Note the presence of a large, external face, with 98 sides.

c[6, {13, 3}, _]: irregular trinet based on hexagonal grid

Unlike the previous case, the trinet produced by this computation exhibits an intrinsically asymmetric structure, as shown in Figure 35. The distribution of face sizes for a computation of length 3000 is:

{{3, 6}, {4, 1}, {5, 24}, {6, 979}, {7, 7}, {8, 7}, {9, 2}, {11, 1}}

It is clear that the largest part of the graph is a hexagonal grid, with 979 hexagons, but some larger faces are now present too. Interestingly, a large external face is now missing, and this is not surprising if we consider the complex revisit indicator, which reveals randomness and fairness in revisiting all parts of the growing net, although traces of regularity and symmetry are also visible. Note that the trinet is drawn in figure as two superimposed layers of roughly the same number of faces.

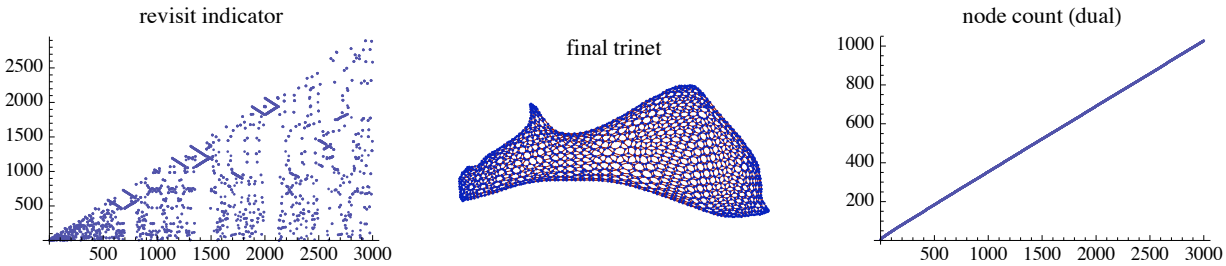

Figure 35 - Revisit indicator, hexagonal grid, and dual node count for c[6, (13, 3), 3000]

c[6, {18, 9}, _]: hexagonal grid with one central pentagon

The trinet produced by this computation is a 2-D hexagonal grid that develops around one pentagon (Figure 36). The distribution of face sizes for a 3000-step computation is:

{{3, 73}, {5, 51}, {6, 1404}, {7, 70}, {194, 1}}

A large, external face is present again. The active point is always at the border of the graph, and the growth process is similar to that of computation c[6, (10, 2), _] above.

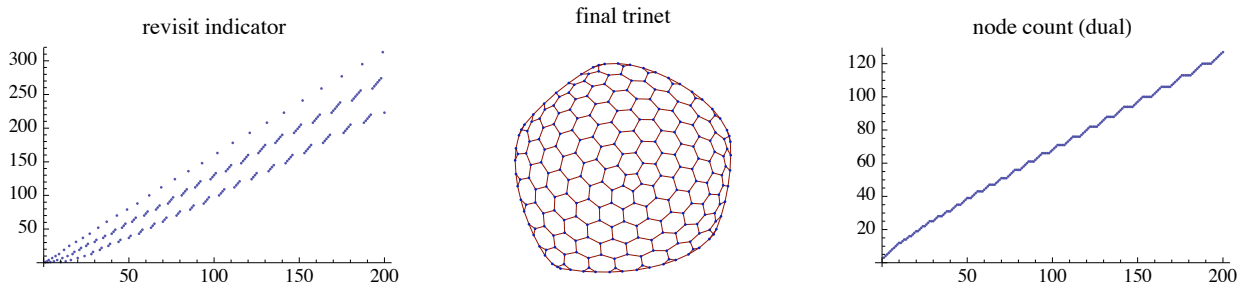

Figure 36 - Revisit indicator, hexagonal grid, and dual node count for c[6, (18, 9), 200]

8. Thresholds 7, 8, 9: further regular 2-D grids

Class C[*threshold* = 7] presents further types of nested trinets, both with linear (e.g. c[7, (9, 9), _]) and with radial (e.g. c[7, (10, 2), _]) revisit indicators, and two more cases of 2-D regular grid: c[7, {9, 2}, _], which gives a pure hexagonal grid, and c[7, {18, 9}, _], which gives a hexagonal grid with one central septagon. Both are illustrated in Figure 37

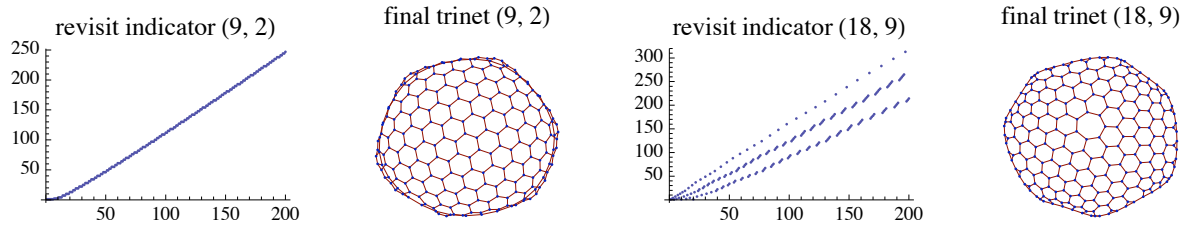

Figure 37 - Revisit indicator and hexagonal grid produced by c[7, (9, 2), 200]

No other 2-D regular trinet is found for *threshold* values 8 and 9, while nested trinets are still present.

9. Infinite threshold

The case *Threshold* = infinite is interesting because the algorithm is forced to always apply the *Refin* rule. The resulting computations are illustrated in Figures 38 and 39, where '*' stands for any value of *DiagsCode* -- a value that the algorithm never uses.

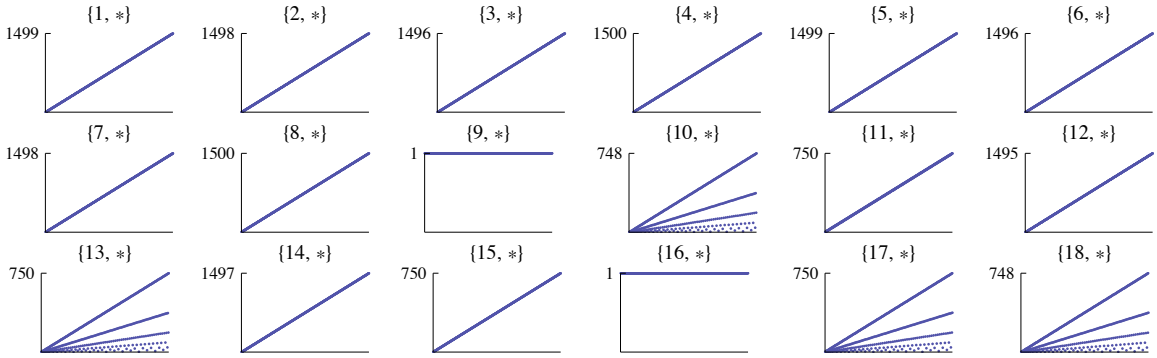

Figure 38 - Revisit indicators for all computations with Threshold = infinite and RefinCodes 1-18, of length 500

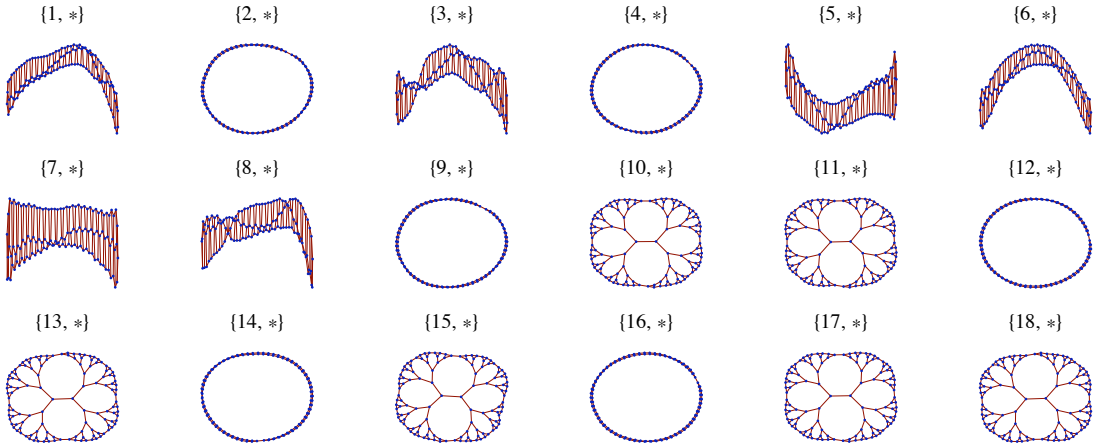

Figure 39 - Final trinets for all computations of length 64, with Threshold = infinite and RefinCodes 1-18,

Note that the trinets refer, in this last diagram, to computations of length 64 only; they evenly split into 6 linear, 6 circular, and 6 nested graphs. The regular, radial revisit indicator of *c*[infinite, {10, 1}, _] is such that each edge (with identifier) in the set

$$\{1, 2\} \bigcup \{3k + 1 \mid k = 1, 2, ...\}$$

appears infinitely often in the diagram, although at an exponentially decreasing rate. More precisely, denoting by $s[e, n]$ the step (number) at which edge e is the current edge for the n-*th* time, the following holds for all visited edges as identified above:

$$s[e, n] = 2\, s[e, n\text{-}1] + 1 \quad (n > 1)$$

where $s[1, 1] = 1$, $s[2, 1] = 2$, and $s[3k + 1, 1] = 2(k + 1)$, $k = 1, 2, ...$ The arithmetic progression of initial edge occurrences, combined with the geometric progression of subsequent appearances of individual edges, determines the characteristic pattern of virtually infinite radii emanating from the origin.

10. Conclusions

In this paper we have introduced an original algorithm for the creation of dynamic planar trivalent networks (*trinets*), and have exhaustively explored all the classes of its computations corresponding to values in the range 3-9, and to infinity, of a *threshold* parameter. A number of quite interesting phenomena have emerged, while it is still unclear whether considering finite values beyond *threshold* 9 could yield qualitatively new features. Among the expected results we have found plenty of computations yielding periodic sequences of bounded trinets, as well as sequences of 1-D trinets growing without bound, where stabilization to regular, linear growth may occur after very long transient phases characterized by pseudo-random dynamics.

More interestingly, we have found complex, but still regular (nested) computations, with trinets that grow while oscillating between linear and ring structures based on simple building blocks, and computations that produce 2-D trinets based on the hexagonal grid. All of the latter exhibit a rather regular, roughly circular shape, and a regular growth pattern; but computation *c*[6, {13, 3}, _] is an exception, and yields a smoothly growing trinet with an irregular shape, no especially large, 'external' face, and irregular motion of the control point.

But in our opinion the most interesting computations are *c*[4, {17, 8}, _] and *c*[5, {9, 8}, _]. Their revisit indicators reveal a surprising ability to densely and fairly revisit the past, while exhibiting pseudo-random dynamics. Graph size, as a function of the number of algorithmic steps, grows as O(Sqrt[steps]). It is likely that this combination of randomness and fairness, guaranteed by an extremely simple algorithm that operates on a purely local basis, could find some useful application, still to be investigated.

We have listed, in the introduction, the differences between our algorithm and the techniques proposed in [W2002] for animating trivalent networks. Our previous proposal for trinet computation [B2007a] was different from the one presented here in the following aspects:
- we used only the *Refin* rule (this can now be obtained as a special case by setting *Threshold* = ∞);
- we used a rather different rule for driving the control point, based on rotations by a constant number of steps around the current node of the dual graph;
- we did not convert duals into primal form. In spite of the conceptual equivalence between the two forms, looking at the original trinet has been practically more helpful for spotting patterns such as oscillating rings.

A demonstration of the algorithm described in this paper is available at the Wolfram Demonstrations Project site [B2007c].

Much more remains to be done.

Loop edges.
Ways for handling trinets with loop edges should be investigated. This may lead to considering new forms of degenerate triangles, and perhaps even to difficulties in keeping simple geometric interpretations of the manipulated data structures. In fact, interesting developments might derive from relaxing or abandoning those interpretations, and concentrating just on abstract data structures and operations. This may also help in exposing possible bridges with other simple computational systems.

Dimensionality.
We have found many trinets for which the attribution of dimensionality 1 or 2 is obvious, but what about all other graphs? In the rather uninteresting cases of tree-like trinets, nodes grow exponentially with the distance from a given, reference node. The identification of a dimensionality for the *finite*, irregular sub-trinets corresponding to the initial phase of eventually periodic computations is not of much interest. But what about the irregular trinets of the two computations *c*[4, {17, 8}, _] and *c*[5, {9, 8}, _] with random revisit indicators? A difficulty here derives from their slow, square-root node growth: in both cases, 100,000-step computations yield trinets with less than 1000 nodes, and, due to this limitation, the analysis of internode distances has not led to clear results. Finally, one might perhaps associate different dimensionalities to the *same* trinet, depending on the observation scale, since some trinets may exhibit different structures at different levels. This would perhaps offer some bridge towards physical theories that envisage a multi-dimensional space -- notably string theory -- with a distinction between extended and compactified dimensions. More generally, it seems desirable to investigate possible connections between trinet mobile-automata

computations and some of the quantum gravity theories proposed by physicists in last few decades. A wide-spanning bibliographical review of theories concerned with the small scale structure of space-time is found in [G1966].

Relativistic, subjective views.
A research track that we consider of highest priority is concerned with the introduction of a relativistic view at trinet computations, based on suitable notions of spacetime / causal network. The trinet diagrams we have shown in this paper, if at all relevant to the understanding of real physical phenomena, would only represent an *external* view of a dynamic physical space (a sort of 'God's eye' view), while the spatial evolution perceived by an *internal* observer, one which is himself part of the universe, would be mediated by the causal relations between observation events. It is quite straightforward to define a causal network for the rewrite events of our algorithm, along the lines discussed in [W2002], chapter 9. Then, taking the point of view of a trinet face *f*, we could define discrete *time*, *relative to f*, in terms of the sequence of updatings of *f*, based on the assumption that a face, regarded as the smallest perceptive unit, does not experience anything, thus no progress of time, between two consecutive updates of *f*. Based on this notion of time, and taking into account the complete causal network of updating events, the idea would be to create a sequence of suitably defined 'pictures' of the universe as perceived by *f* at successive instants of its time. This sequence would represent the correct, internal view of the produced dynamic space, and the one to be appropriately compared with physical reality as we perceive it. We are particularly interested in applying this analysis to computations *c*[4, {17, 8}, _] and *c*[5, {9, 8}, _], whose randomness and fairness might prove essential for guaranteeing the equivalence of all possible observers.

Acknowledgment
We express our gratitude to Marco Tarini, at ISTI, for several discussions on trivalent network representation.

References

[B2007a] T. Bolognesi, 'Planar trivalent network computation', in: J. Durand-Lose, M. Margenstern (eds.), Machines, Computations and Universality, Proceedings of MCU 2007, Lecture Notes in Computer Science, N. 4664, Springer 2007.

[B2007b] T. Bolognesi, 'Planar trinet dynamics with two rewrite rules', Technical Report ISTI-015/2007, CNR, ISTI, Pisa, 2007.

[B2007c] T. Bolognesi, "Planar Trivalent Network Growth Using Two Rewrite Rules", The Wolfram Demonstrations Project, http://demonstrations.wolfram.com/PlanarTrivalentNetworkGrowthUsingTwoRewriteRules/

[G1996] P. E. Gibbs, "The Small Scale Structure of Space-Time: A Bibliographical Review", http://arxiv.org/abs/hep-th/9506171v2

[W2002] S. Wolfram, A New Kind of Science, Wolfram Media, 2002.