Planar trivalent network computation

Tommaso Bolognesi[1]

CNR-ISTI, Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", 56124, Pisa, Italy
T.Bolognesi@isti.cnr.it,

Abstract. Confluent rewrite systems for giant trivalent networks of unstructured nodes have been investigated by S. Wolfram as possible models of space and spacetime, in the ambitious search for the most fundamental, computational laws of physics. We restrict here to planar trivalent nets, which are shown to support Turing-complete computations, and take an even more radical, approach: while operating on network duals, we use just *one* elementary rewrite rule – which is singled out also by Loop Quantum Gravity – and drive its application by (two variants of) a simple, fully deterministic algorithm, rather than by pattern-matching. We devise effective visual indicators for exploring the complexity of computations with elementary initial conditions, consisting of thousands of graphs, and expose a rich variety of behaviors, from regular to random-like. Among their features we study, in particular, the dimensionality of the emergent space.

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

1 Introduction

The idea that physical laws and the whole evolution of our universe ultimately reduce just to computation was first proposed by Konrad Zuse in 1967 [1, 2], and later investigated, under various distinct viewpoints, by scientists such as Edward Friedkin, Gerard 't Hooft, Juergen Schmidhuber, Seth Lloyd. Sometimes called 'Digital Physics', this approach suggests that the entire history of the universe is precisely captured, in every tiny detail, by the discrete, fully deterministic output of a presumably short 'program', and has been recently adopted, further developed and popularized by Stephen Wolfram, with his so-called 'New Kind of Science' (NKS) [3].

As of today, no small program has been yet identified that can exactly reproduce fundamental physical constants such as subatomic particle masses. However, a rather convincing, though purely intuitive argument supporting the computational physics conjecture is represented by the spectacular emergent properties of simple programs such as elementary cellular automaton (ECA) n. 110, which plays a distinguished role in [3]. The computations of this ECA exhibit the

spontaneous appearance of a regular background 'space' and of 'particles' that move and interact in complex ways, giving to the observer the impression of an artifical universe following its own physical laws. The other remarkable property of cellular automaton 110 is that it is the only ECA (with its left/right and black/white symmetries) known to be Turing-complete, that is, capable of universal computations [4], and this is commonly regarded as another requirement for any candidate computational model of physics.

Science has always being concerned with trying to explain larger sets of phenomena with simpler, unifying formulae; with computational physics, one ultimately expects all of the complexity observed in nature to emerge from the computations of a very 'cheap' formal system. In this respect, cellular automata are quite unsatisfactory, since they assume the existence of an infinite, discrete, n-dimensional array of binary cells (with $n = 1$ for ECAs), representing the space where computations take place, rather than letting space and spacetime emerge from the computation itself. Cellular automata cannot be the actual laws of physics: they only metaphorically suggest that computational physics is on the right track.

For this reason, [3] proposes dynamic networks as a more adequate model for fundamental physics, with three dimensional space as one of the required emergent properties.

In Section 2 we quickly review Wolfram's ideas about space as a trivalent network that evolves by rewrite rules, and about spacetime as a causal network.

In Section 3 we restrict to planar trivalent networks and prove that they can support universal computation.

In Section 4 we devise, by exploiting network planarity, an alternative and more radical approach to the evolution of trivalent networks, based on (two variants of) a fully deterministic algorithm which is simpler than those proposed in [3], and avoids some of the difficulties they involve.

In Section 5 we introduce two convenient visual indicators and use them for analysing the computations of a first variant of our algorithm, which consist of sequences of trivalent network duals of growing size. This allows us to expose a number of interesting features.

In Section 6 we study the computations of a second variant of the algorithm.

In Section 7 we discuss dimensionality issues.

In Section 8 we draw some conclusions and identify items for further study.

2 Networks and rewrite systems for space and spacetime: the NKS approach

NKS adopts the idea that all the features of our universe emerge purely from properties of space, or, more precisely, of some underlying structure from which space itself, in its familiar 3D form, emerges. In order to obtain the rich variety of physical properties that we observe it seems appropriate to assume an underlying structure that is as simple and flexible as possible. Based on these observations, Wolfram suggests that, at the lowest level, space could be a giant network of

nodes (a graph) which is (i) undirected, and (ii) trivalent, meaning that all nodes have *degree* three – precisely three edges insist on every node. Nodes ('vertices') and edges are unlabeled and unstructured.

We shall use the term *trinet* for denoting this minimally structured type of object.

The *distance* between two nodes is the length – number of edges – of the shortest path between them.

We say that a trinet has (or induces a space of) *dimensionality* k when, given a generic node n, the number of nodes that are at distance at most d from n grows like d^k.

In spite of its simplicity, a trinet can achieve any dimensionality: a ladder-shaped trinet yields a one dimensional space, an hexagonal grid is a trinet yielding a two dimensional space, and so on. In general, a trinet can 'implement' the structure of any undirected graph – one with unconstrained node degrees – by replacing any node n of degree $r > 3$, by a cycle of r new trivalent nodes $n_1, ..., n_r$, where each edge originally insisting on n now insists on a different n_i. (This is similar to replacing the crossing of several roads by a roundabout.)

If our universe is a trinet, it cannot be a regular one, since its pattern of connections must encode all the complex phenomena we observe. According to Wolfram, on a small scale the net will most likely look quite random, but on a larger scale it must be arranged so as to correspond to ordinary three-dimensional space, and this feature should be preserved by the rules that make it evolve.

In discussing the evolution of trinets, Wolfram considers various graph rewrite rules, and policies for applying them. The very first considered rule is depicted in Figure 1. This rule transforms a single node into a cluster of three nodes,

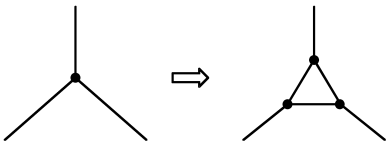

Fig. 1. A simple graph rewrite rule: node tripartition

and, in this respect, is analogous to the rules of a *context-free* grammar. In this paper, we call it *node tripartition rule*. Interestingly, this transformation plays a role, with its reverse, also in the theory of Loop Quantum Gravity [1], and is one of just two fundamental transformations singled out by Lee Smolin in his divulgative paper on this theory [5].

[1] Loop Quantum Gravity claims that both space and time are quantized, and uses *spin networks* (first introduced by Roger Penrose in 1964) for modeling the quantum states of space. A *spin network* is a graph where every node represents an elementary "quantum of volume" and every link is a "quantum of area" delimiting it; its evolution can be achieved by graph rewrite rules.

By applying the rule in parallel to each node, at each step, starting from a simple trinet such as the one formed by the edges of a tetrahedron, regular, nested structures can be obtained. Indeed, one of the recurrent observations from [3] is that context-free rewrite systems cannot produce, by themselves, more complexity than nesting. For this reason Wolfram investigates more elaborate trinet evolution systems, in which rewrite rules are somewhat analogous to those of *context-dependent* grammars, and transform *clusters* of nodes.

However, once some set of graph rewrite rules is selected, one has to decide how to handle the various sources of nondeterminism that their application may involve, that is, how to reduce to a deterministic evolution. If many rewrite rules are potentially applicable to the trinet in its current state, which one should be applied? Ordering the rules according to some priority does not eliminate nondeterminism completely, since even a single rule could be applicable in different ways. For example, the node tripartition could be applied in 4 different ways to the tetrahedron trinet. On the other hand, insisting on applying all applicable rules simultaneously does not solve the problem, since several different maximal sets of rewritings may be simultaneously enabled.

One way proposed by Wolfram for obtaining complete determinism consists in enriching the rewriting process by state information that records the 'age' of nodes: when several rewritings are simultaneously enabled at different locations of the net, the one that involves, say, the youngest nodes is selected (see [3], p. 511, for details).

The other graph rewriting approach investigated in [3] consists in restricting to *confluent* rewrite systems, that always guarantee the same evolution in-the-large, regardless of nondeteminism in-the-small. This idea can be made precise by the notion of *causal network*, which is proposed by Wolfram as a model for spacetime.

A *causal network* is a directed graph in which nodes represent events and edges represent their causal relationship. If the events are local state changes – e.g. the updating of a cell – then a direct causal dependency between events e_1 and e_2 is established when the cell(s) updated by e_1 directly influence the updating performed by e_2.

Causal invariance is the property of a rewrite system to yield exactly the same causal network regardless of the rule application order. If a rewrite system is causal invariant, then its causal net retains the essential, unique partial order of events behind any contingent total ordering. Then, if one is interested in building spacetime by some rewrite system, *causal invariance* becomes a very appealing requirement.

While a more detailed account of Wolfram's ideas and techniques for network rewriting is out of the scope of this paper, and the interested reader is invited to consult the mentioned sources, we wish to point out three problematic aspects of them:

– The idea to resolve nondeterminism in network rewriting by using state information in form of time stamps appears undesirably complex, and as un-

natural as, say, the requirement of a global clock for synchronizing rewritings in cellular automata.
- Causal invariance is an elegant property. By adopting a set of rewrite rules that enjoys it, and by identifying the computation with the unique, underlying causal network, one does not have to introduce any further machinery, such as a global clock or a localized, mobile control (as in Turing machines or mobile automata) in order to obtain a unique computation. Unfortunately, the search for systems that guarantee this property involves a number of technical difficulties (see [3], p. 493) and is computationally hard. And the sufficient conditions proposed for achieving causal invariance "*that all the replacements that appear in the rules should be for clusters of nodes that can never overlap themselves or each other*" is perhaps too restrictive.
- While the computations of elementary cellular automata lend themselves to a very effective visualization, where all sorts of emergent properties are immediately detected, the direct representation of sequences of trinets of growing size is not ideal for visual inspection.

Our paper is mainly motivated by the need to overcome these limitations.

3 Planar trinets, duals, and computational universality

In this paper we shall restrict to connected, planar trinets. A graph is *planar* when it can be drawn on (or 'embedded' in) the plane so that no edges intersect; the plane is thus partitioned into polygonal *faces*, including the external one, which has infinite size. [2]

In our developments we shall take advantage of the well known, *dual* representation of a planar graph, which is itself a planar graph representing the adjacency relations between faces of the original graph. For example, Figure 2 shows the only two distinct trinets with two nodes, and four out of the six possible distinct trinets with four nodes, and their respective duals. Recall that the dual operation d is the inverse of itself: $d(d(G)) = G$.

Note that some of the trinets include self-loops and/or double edges, which delimit degenerate polygonal faces. A graph without self-loops and double edges is called *simple*.

The following facts can be easily established about planar trinets and their duals.

Proposition 1. *A trinet may only have an* even *number of vertices.*

Proof. By definition each node is connected to three distinct edges, or to a loop-edge and to a 'normal' one. Assume there are n nodes and e edges. By charging three edges to each node, in the obvious way, we count each edge exactly twice, including loops:

$$3n = 2e.$$

[2] This 'external' face has no real special status, as it appears by drawing the graph on a sphere.

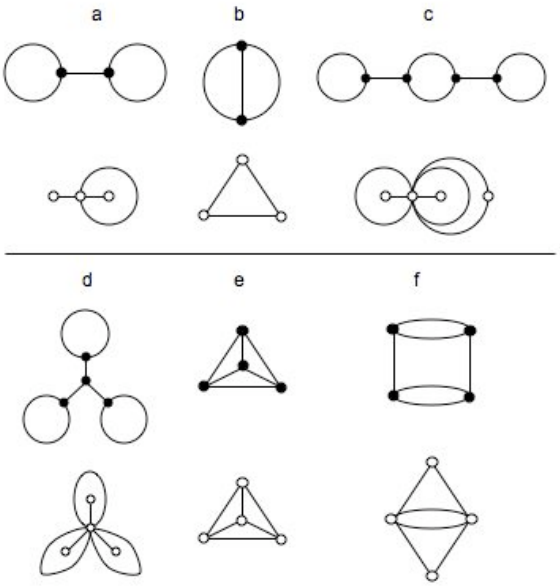

Fig. 2. Simple trinets (black nodes) and their duals (white nodes)

This proves that n is even. □

Proposition 2. *The dual D of a simple, planar trinet is a planar graph that satisfies the following properties:*

- *D is simple;*
- *each node has degree $d \geq 3$;*
- *each face is a triangle.*

Proof. Trivial.

Note that the dual of a simple planar graph which is *not* a trinet is not necessarily a simple graph, as illustrated by case (b) in Figure 2: the dual of the triangular graph, shown above it, has double edges.

The choice of using only *planar* trinets is not excessively restrictive. First, planar trinets may well yield dimensionalities other than 2; for example, two trinets with fractal dimensionalities $\log_2 3$ and $\log_3 7$ are shown in [3] (p. 509). Second, planar trinet rewriting has maximum computing power, as established by the following proposition.

Proposition 3. *Planar trivalent network rewriting is Turing-complete*

Proof. It is sufficient to exhibit a planar trinet rewrite system that simulates a universal Turing machine (TM). Thus we consider the smallest known universal

Table 1. State table for smallest universal Turing machine

	1	2	3	4	5
s_1	s_1, 2, +1	s_1, 1, +1	s_1, 1, +1	s_2, 5, +1	s_2, 4, -1
s_2	s_1, 4, -1	s_1, 1, +1	s_1, 5, +1	s_2, 5, +1	s_2, 3, -1

TM, which was introduced in [3] (p. 707), and uses a tape alphabet of 5 symbols {1,2,3,4,5} and two states {s1, s2}; Table 1 provides its transitions. Figure 3 shows a coding scheme for two-state, five-symbol TMs (which can be directly extended to any other TM). The idea is to code each cell, including a tape symbol and possibly control state information, by an *elementary* sub-trinet. The trinet in (a), shown in two possible layouts, illustrates the overall structure

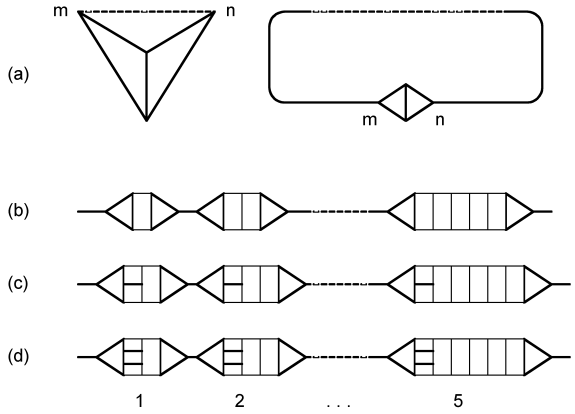

Fig. 3. Building blocks for the trinet coding a two-state, five-symbol Turing machine

of a coded TM configuration. The dotted edge stands for a sequence of *at least one* of the elementary sub-trinets further detailed below, which code the cells of the TM that are explicitly assigned an initial value, or are written during the computation. The two triangles sharing one edge code the infinite portion of the tape not yet reached by the TM head, whose cells are occupied by the 'background' symbol 1. In (b) we show the coding of the pure tape symbols. In (c) (resp. (d)) we enrich this coding for representing the presence, on the tape cell, of the TM head in state s_1 (resp. s_2). Note that the elementary sub-trinets in (c) and (d) are linked only for visual clarity; such sequences may not appear in any computation, due to the presence of multiple instances of control state information!

Defining trinet rewrite rules that implement Table 1 is easy. For example, the six rewrite rules corresponding to the lower-left slot in the table, with the

head in state s_2 positioned on a cell containing a '1', are depicted in Figure 4. The rewriting schema on the left, where the dots denote the coding of some of

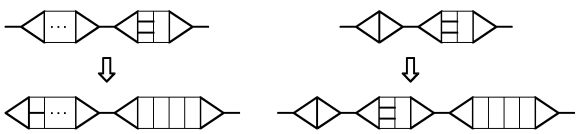

Fig. 4. Trinet rewrite rules coding the lower-left slot of table 1

the tape symbols, accounts for five different rewrite rules; the rewriting on the right handles the case of the head moving to a portion of the tape not visited before. In conclusion, we need a total of 60 rewrite rules.

In devising the coding, one has to make sure that the rewrite rules match only the intended sub-trinets. One way to check this is to reason in terms of patterns of polygonal faces. The impossibility to apply a rewrite rule to the wrong sub-net is then guaranteed by the observation that:

- the sub-net coding a plain cell must only include polygons of arities 3 and 4;
- the sub-net coding a cell with control state $s1$ must only include polygons of arities 3, 4, 5;
- the sub-net coding a cell with control state $s2$ must only include polygons of arities 3, 4, 6;
- the net coding the whole TM configuration must always include a polygon of arity at least 8 (and the patient reader may check that the simplest initial configuration, with the tape uniformly filled by 1's, is coded by a planar trinet that already includes two octagonal faces).

□

Note that the trinets used for the simulation above are simple (no loops or double edges).

4 A fully deterministic planar trinet growth algorithm

The system of 60 rewrite rules described in Section 3 is fully deterministic, once assuming legal initial conditions, since at each step the trinet includes only one of the sub-trinets that code the current (unique) cell with control information, and there is only one rule that matches this sub-trinet and its appropriate (left or right) neighboring elementary sub-trinet. But the price for obtaining full determinism has been to define rewrite rules that manipulate relatively complex, ad-hoc sub-trinets. Defining a rewrite system that handles simpler sub-trinets while preserving full determinism would be much harder.

On the other hand, in line with the NKS style of investigation, we are interested in exploring the computational universe created by *very* simple rewrite

systems, and in this paper, in particular, we take the extreme approach of choosing just *one* rule.

The rule we choose is node tripartition; since our algorithm is conveniently formulated in terms of trinet duals, we show in Figure 5 how this rule operates on them, assuming *simple* trinets (in grey), thus *simple* trinet duals (in black). In terms of duals, the rule consist in placing a new node inside a face (which must be a triangle, by Proposition 2) and splitting the latter into three triangular faces.

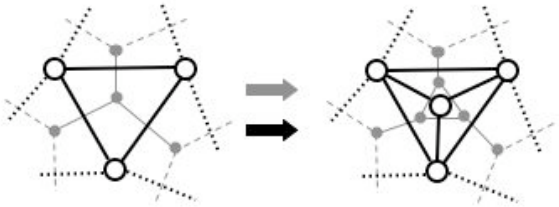

Fig. 5. Node tripartition for trinets (grey) is face tripartition for trinet duals (black)

Used without further algorithmic control, the tripartition rule induces the maximum possible nondeterminism, since it can be applied, by definition, to every node of any trinet, or, equivalently, to any face of the dual! Thus, we need a way, as simple as possible, to introduce full determinism in the rewriting process, that is, to decide where to apply the rule at each step.

Informal description of the algorithm. The idea, illustrated in Figure 6, is to move across the dual graph by unit steps, from a node to some adjacent node, e.g. from S to T_0, without jumping, in a sort of (deterministic!) brownian motion, and to use a simple criterion for selecting the face to split, based on a minimum amount of information – less information than that provided, for example, by TM state transition tables.

The selection criterion is as follows. When node T_0 is reached, coming from node S, the cycle of radius one centered in T_0 is considered, and the nodes $T1$ and $T2$ at distances k and $k+1$ from S, moving counterclockwise, are selected, with T_0 itself, as the vertices of the triangle to be split.

We have explored two variants of the algorithm, corresponding to different ways to define k:

- k is a (small), constant nonnegative integer, or
- k is the length of the cycle (partially) visited in the previous step of the algorithm.

Note that k may well determine more than one rotation around T_0.

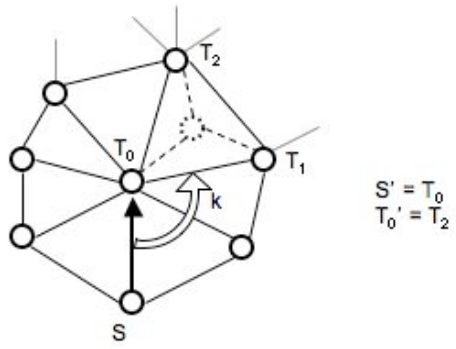

Fig. 6. Finding the triangle (T_1, T_2, T_3) for the tripartition, coming from node S

Initial conditions. The computation step assumes and returns graphs that are *simple*. Therefore, as an initial condition we choose the simplest trinet dual which is simple: this is the triangular graph reproduced, again, in the lower-left corner of Figure 7. Note that the trinet corresponding to this triangle, shown above it, is not simple. Yet, any application of the node tripartition rule to this trinet yields the tetrahedron trinet, whose dual is the tetrahedron graph itself. Regardless of

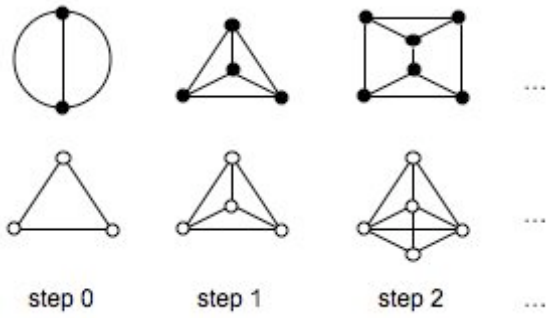

Fig. 7. Trinets and trinet duals for the common prefix of any deterministic computation based on the tripartition rule

the different choices for k, all computations share the initial sequence of three steps illustrated in Figure 7, and all the trinet duals that one gets, as well as all the corresponding trinets, are simple, except for the first trinet.

Code. The *Mathematica* code for the function that implements the computation step, in case the k parameter is a constant, is:

```
step[{TND_, S_, T0_, k_, n_}] :=
  With[{T0Adj = TND[[T0, 2]]},
    With[{T1 = rotax[T0Adj, S, k], T2 = rotax[T0Adj, S, k + 1]},
      {Replace[Append[TND, {n, {T0, T1, T2}}],
          {{T0, adjT0_} :> {T0, adjT0 /. {T1 -> Sequence[T1, n]}},
            {T1, adjT1_} :> {T1, adjT1 /. {T2 -> Sequence[T2, n]}},
            {T2, adjT2_} :> {T2, adjT2 /. {T0 -> Sequence[T0, n]}}},
          1],
        T0,
        T2,
        k,
        n + 1}]]
```

Function *step* accepts a tuple of five parameters:

1. *TND*, for TriNet Dual, is the adjacency list structure representing the current graph, in which nodes are identified by progressive natural numbers;
2. S is the central node of the previous step ('from-node');
3. *T0* is the central node of the current step, reached from S ('to-node');
4. k is the parameter used for selecting nodes T_1 and T_2 and identifying the triangle to be split;
5. n is the progressive natural number to be associated with the node created in the current step – its identifier.

The function returns an updated 5-tuple $\{TND', T_0, T_2, k, n+1\}$, where *TND'* is the trinet dual increased by a new node numbered n, and parameters T_0 to T_2 play the roles, respectively, of the new from-node and to-node. Auxiliary function $rotax[aList, anElem, k]$ finds the k^{th} element of a circular list *aList*, starting from element *anElem*.

A whole computation of, say, 20,000 steps, based on constant parameter $k = 2$, starting from the triangular graph, assuming to move initially from node 1 to node 2, with the counter for new node identifiers initialized to 4, is obtained by the convenient *Mathematica* predefined function *NestList*, which iterates function *step* for the desired number of steps:

```
initTND = {{1, {2, 3}}, {2, {3, 1}}, {3, {1, 2}}};
k = 2;
steps = 20000;
myComputation = NestList[step, {initTND, 1, 2, k, 4}, steps]
```

In the next two sections we analyze the computations of the two variants of the algorithm corresponding to different ways to define parameter k.

5 Visual indicators and computations with constant *k*

One of the key lessons from [3] is that important clues about the computing power of a given formal model can be obtained by inspecting the 'shape' of its computations, besides using it for simulating another reference model of known

power. However, different models lend themselves to different visualization techniques, not all of which are as effective as the ECA diagrams thoroughly studied in [3]. Furthermore, when computations involve complex data structures, and complete visualization is unmanageable, the problem arises of which aspects to expose, as potential *indicators* of computational complexity. (A taxonomy of visual complexity indicators is proposed in [6].)

Trinet computation precisely suffers from this difficulty, since periodicity, nesting, pseudo-randomness, or other emergent properties are hardly detected by directly inspecting sequences of graphs with thousands of nodes. But a computation of our algorithm is actually a sequence of *quintuples*, of type

```
(TND, fromNode, toNode, k, n)
```

and this provides us with further parameters, simpler than the complex graph structure TND, for visualizing its character. In particular, we choose two indicators:

Face size - this is the number of neighbors of *fromNode* in the current trinet dual *TND*, that corresponds to the number of edges in the current polygonal face of the trinet. By looking at this parameter one takes a very localized view at the computation, which only retains information, step by step, about the shape of one of the three polygons affected by the node tripartition rule.

Face id - this is the progressive number of the current face. Again, this parameter provides very partial information, but it is quite useful for revealing the extent to which the computation may go back to previously created/visited regions of the growing network.

Note that each time a polygonal face is visited, its size is increased by one. Figures 8 and 9 show the dynamics of the two indicators above for computations obtained by using different, constant values for parameter k (0 to 4, and 5 to 9, respectively). The two figures also include the plot of the final graph of each computation – a trinet dual; recall that this graph and the corresponding trinet are always planar. [3] All computations start from the triangular trinet dual (Figure 7), thus the initial value for *Face size* is always 2, which is the number of neighbors of the initial node (and the number of sides of the initial trinet face).

When $k = 0$ the computation is very regular: parameter *Face size* indicates that the current face stabilizes in three steps to a pentagon, and *Face id* indicates that the algorithm never returns to an already visited face. The graph in the right column reveals the linear overall structure of the trinet dual, which implies a linear structure for the trinet itself. The case of $k = 1$ is similar, with the current face stabilizing now to an hexagon.

When $k = 2$ the computation is still regular, but more interesting. There is no bound to the size of the visited polygon, and each face, except for face 1, is visited infinitely often, so that its size grows unboundedly too. More precisely, it turns out that node h, with $h = 2, 3, \ldots$ is the *from-node* in steps $(2h-3)2^n$, for

[3] The plot is obtained by *Mathematica* function *GraphPlot*, using the *SpringModel* layout.

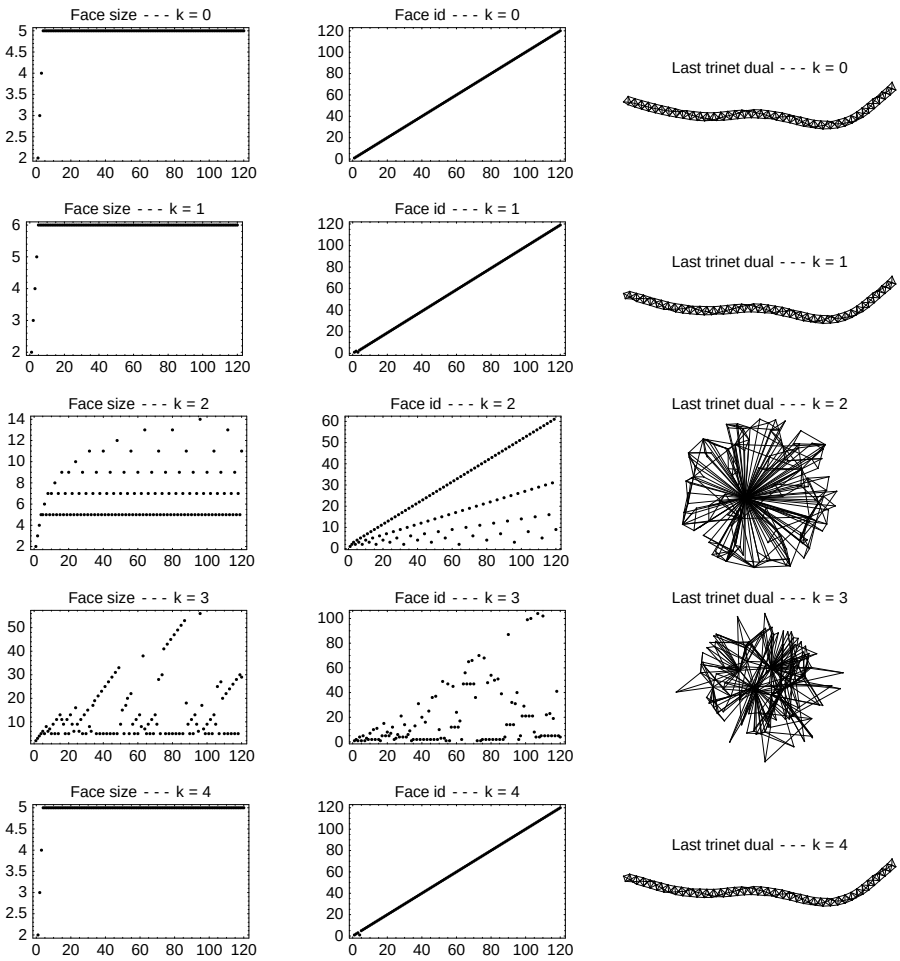

Fig. 8. Dynamics of indicators 'Face size' and 'Face id', and last trinet dual, for 120-step computations with fixed k (0 - 4)

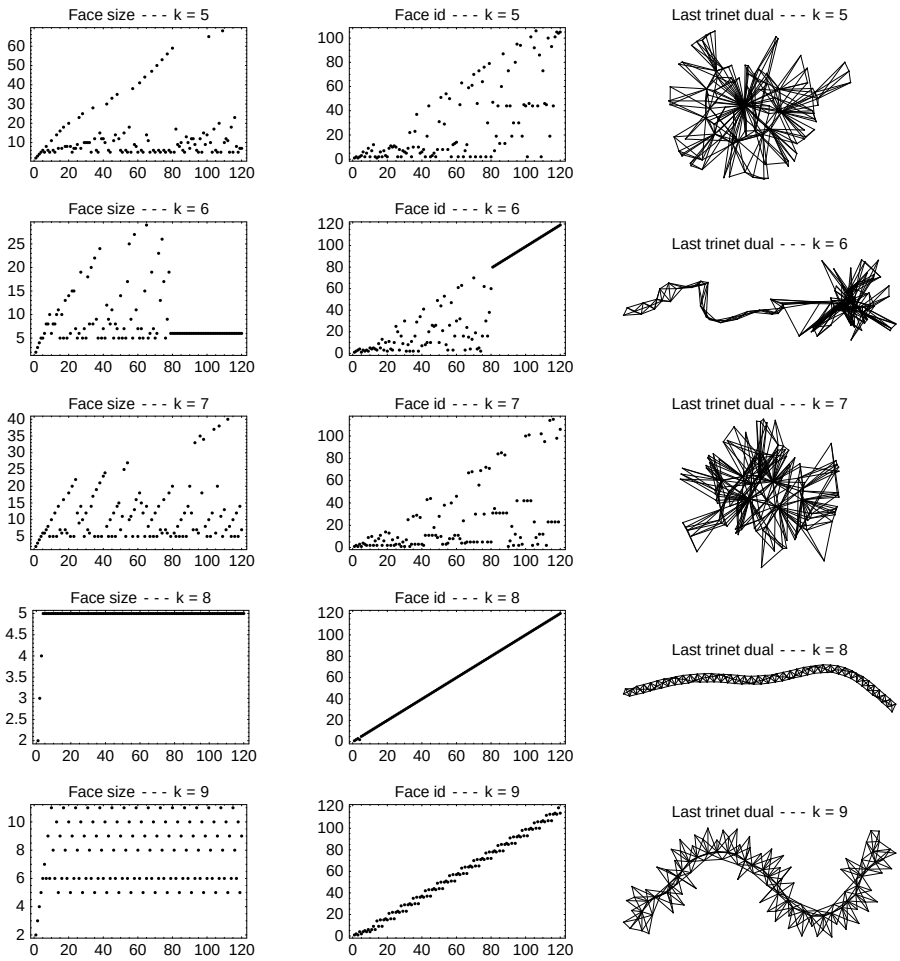

Fig. 9. Dynamics of indicators 'Face size' and 'Face id', and last trinet dual, for 120-step computations with fixed k (5 - 9)

$n = 1, 2, \ldots$, except that node 1, not node 2, is the *from-node* in step 1. Node 1 is visited only once, as a *from-node*, yet its degree is increased by 1 at each step, so that it is always the node with highest degree.

Cases $k = 4$ and $k = 8$ are basically the same as case $k = 0$. In case $k = 6$ the dynamics of the two indicators behaves randomly for 79 steps and then stabilizes to visiting hexagons, while never returning to already visited faces. The random and the regular parts are clearly visible in the final graph. In case $k = 9$ both indicators soon stabilize to a regular dynamics, with period 7. Faces are visited more than once, but all of them are eventually left forever. The trinet dual shows a linear structure.

We are left with cases $k = 3, 5, 7$, in which the plots appear largely random. However, 2000-step computations reveal eventual periodicity in the first two cases, as shown in Figure 10. In case $k = 3$ we get a computation which

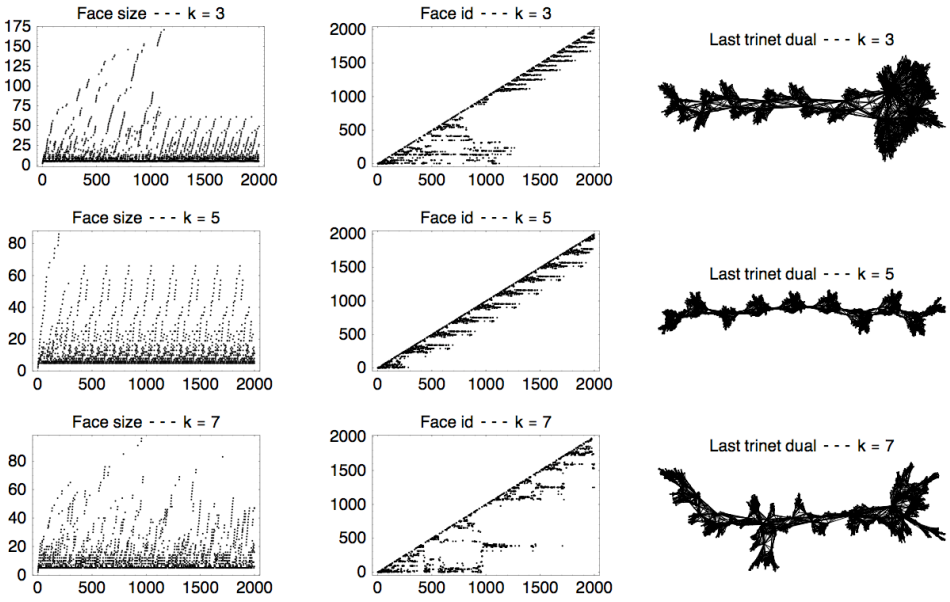

Fig. 10. Dynamics of indicators 'Face size' and 'Face id', and last trinet dual, for 2000-step computations with fixed k (3, 5, 9)

stabilizes after 1229 steps to a cyclic behaviour with period 140, written, for short, 1229/140*; in case $k = 5$ we get a 441/205* computation. Case $k = 7$ is more complex, and only by looking at about 30000 steps we discover a periodic, 27450/850* computation (Figure 11).

In conclusion, all computations illustrated so far, except for one, have the property of permanently leaving the region created in the initial, transient seg-

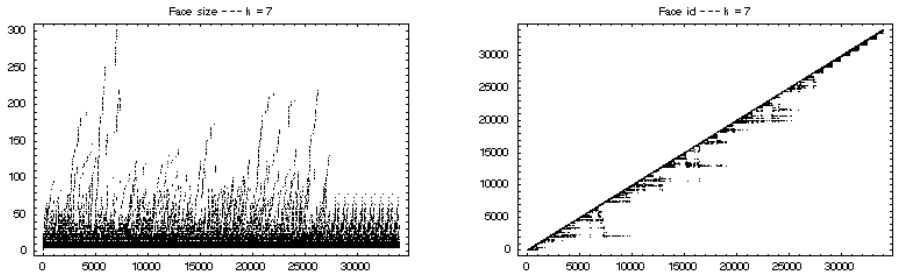

Fig. 11. Dynamics of indicators 'Face size' and 'Face id', for a 34000-step computation with fixed k = 7

ment of the computation, when the periodic behavior is established. The remarkable exception is the case of $k = 2$ shown in Figure 8, where control goes back infinitely often to infinitely many faces.

As a variant of the algorithm, and referring to Figure 6, we have used node T_1 in place of T_2 as the next current node. Two regular computations are illustrated in Figure 12 [4], corresponding to $k = 1$ and $k = 5$. In the first case the same three

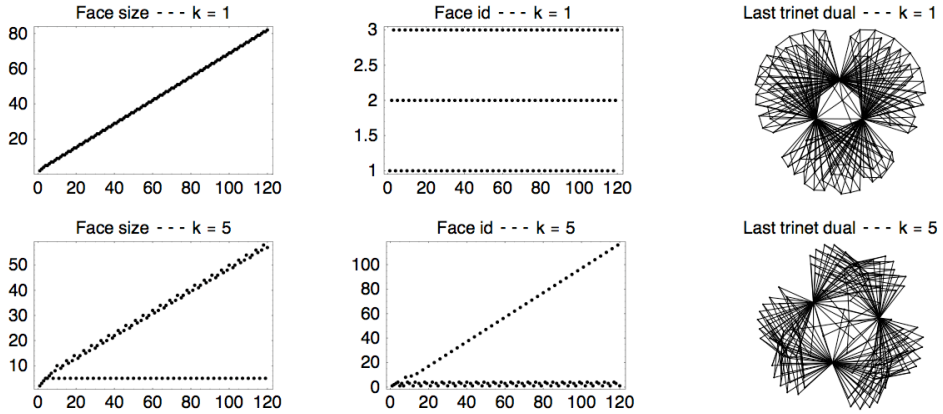

Fig. 12. Computations with T_1 in place of T_2 as next current node, and fixed k (1, 5)

faces are cyclically visited, and three nodes with high degrees are readily spotted in the final trinet dual graph; in the second case control alternates between

[4] The plot is obtained by *Mathematica* function *GraphPlot*, using now the *Radial-Drawing* layout.

visiting the same three faces and a new one. Thus, the new feature exhibited by these computations is that they can go back infinitely often to previously visited faces, but only to a finite number of them.

Figure 13 illustrates the case $k = 9$, corresponding to a computation that settles to regular behavior after over 20,000 steps.

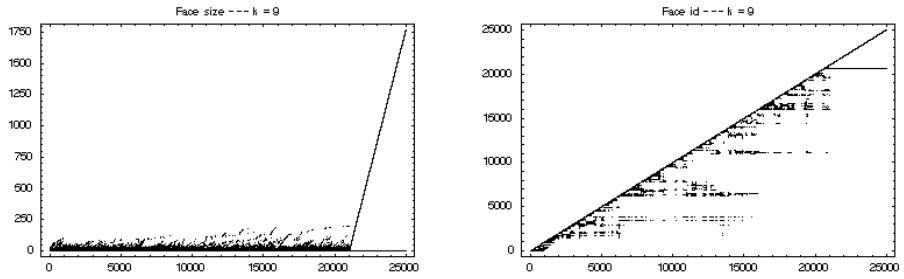

Fig. 13. Computation with T_1 in place of T_2 as next current node, and $k = 9$

6 Computations with variable k

In this section we consider a variant of our deterministic trinet growth algorithm in which parameter k is no longer a constant, but depends on the number n of neighbors of the node that was central in the previous step. In particular, at step i, with $i = 1, 2, \ldots$, we use $k_i = n_{i-1} + h$, where h is a constant. The initial condition is the triangular trinet dual, as before.

Figures 14 and 15 show computations for various values of h. No new, qualitatively different feature seems to emerge from this variant of the algorithm, and the property of going back infinitely often to infinitely many faces is not found again. The case of $h = 2$, illustrated by the 9000-step computation in Figure 16, achieves the longest period (1992/3769*) among those shown in the paper.

7 Emergent dimensionality

What is the dimensionality of the spaces – the trinet dual and, more importantly, the trinet itself - created by the various computations of the tripartition algorithm?

Let us first consider the first group of computations, shown in Figures 8, 9, 10, 11, corresponding to the choice of T_2 as the central node of the next step of the algorithm, and to $k = 0, 1, \ldots, 9$. In the cases when the periodic structure of the last trined dual graph is simple and clearly visible, it would seem natural to conclude that the graph has dimensionality one. The informal argument in

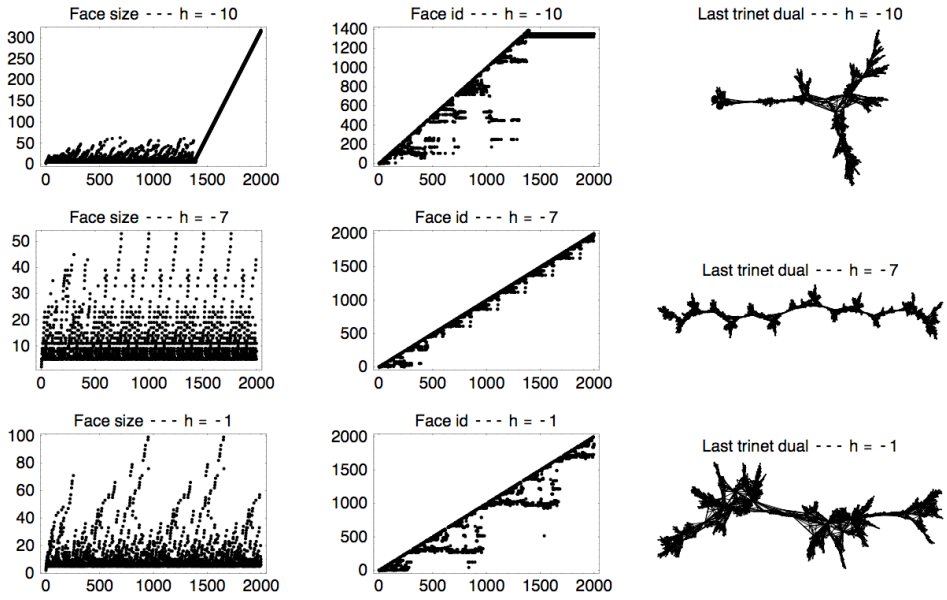

Fig. 14. Computations with $k_i = n_{i-1} + h$, where n_{i-1} is the number of neighbors of the central node from the previous step, and $h = -10, -7, -1$

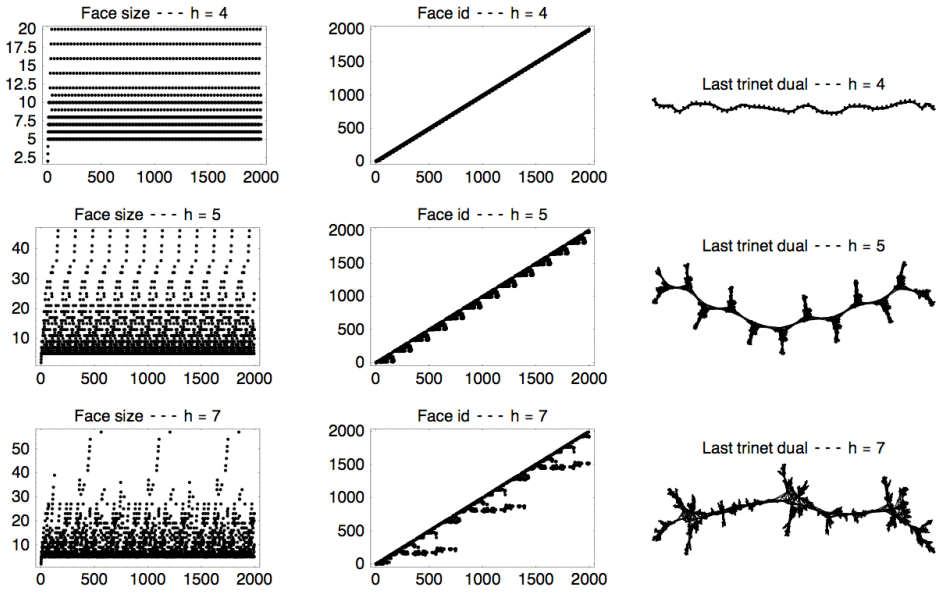

Fig. 15. Computations with $k_i = n_{i-1} + h$, where n_{i-1} is the number of neighbors of the central node from the previous step, and $h = 4, 5, 7$

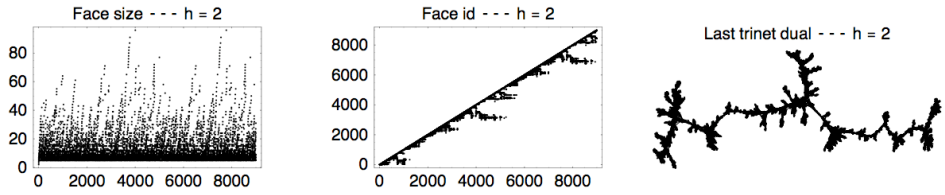

Fig. 16. Computation with $k_i = n_{i-1} + h$, where n_{i-1} is the number of neighbors of the central node from the previous step, and $h = 2$

support of this claim is as follows. These periodic and planar networks, whose nodes have bounded degree, appear as embedded on a cylindrical surface with uniform node density, yielding the long and thin graph structures observed in some of the plots. As a consequence, their duals are structurally equivalent to them: they are periodic, with bounded-degree nodes, and embedded on the same cylindrical surface, with uniform node density. Thus, the original trinets are one-dimensional too.

The reasoning above can be made more rigorous, also in light of the fact that, in some cases, we may still have doubts about the precise structure of the graphs, whose appearance largely depends on the specific visualization method. Plots like those for the 'Face size' and 'Face id' indicators, though presenting less information (and, indeed, due to that) are more clear and effective in exposing regularity. Let us then consider a variant of the 'Face id' plots that visualizes, for each step of the algorithm, not only the identifier of the current node T_0, but also those of the other two nodes T_1 and T_2 of the triangle being split, and the identifier X of the new node being created. Figure 17 shows such plot for $k = 9$. Each node on the diagonal is a new node being created, while the three nodes

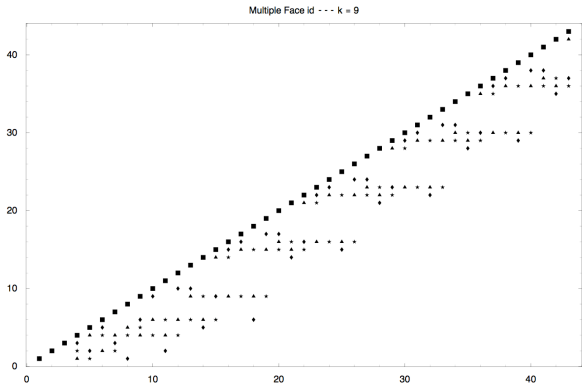

Fig. 17. Plot 'Multiple Face id' for a computation with $k = 9$

below it, to which the new node is connected by three newly introduced edges, indentify the triangle being split. The advantage of this plot is revealed by the following proposition.

Proposition 4. *If, for a given computation c, all points of the (infinite) 'Multiple face id' plot fall in the region between two parallel lines, then the (infinite) trinet of c is at most one-dimensional.*

Proof. The points on the diagonal of the plot represent the integer ids of *all* nodes in the trinet dual, and all the other points correspond, one to one, to its edges: the point with coordinates (i, j) in the plot can be taken to represent, by

construction, the edge between nodes i and j. Let $y = x$ and $y = x - k$, for some positive integer k, be the equations of the two lines of the proposition. Keeping in mind the definition of distance between nodes, and taking advantage from the node numbering performed by the algorithm, the following inequality holds for the distance between two nodes i and j:

$$d(i,j) \geq \frac{|i-j|}{k}, \tag{1}$$

where, with a small notational abuse, i and j represent nodes in the l.h.s, and integers in the r.h.s.. Thus, denoting by $N_i(r)$ the number of nodes found at distance at most r from i, we have:

$$\begin{aligned} N_i(r) &= |\{j | d(i,j) \leq r\}| \\ &\leq |\{j | \frac{|i-j|}{k} \leq r\}| \quad \text{[by (1)]} \\ &= |\{j \mid \ |i-j| \leq rk\}|, \end{aligned}$$

which proves that $N_i(r) \in O(r)$. □

It turns out that all the computations of the considered group, except for the case $k = 2$, satisfy the bounding condition of Proposition 4.

The second group of computations was illustrated in Figures 12 and 13. Figure 18 shows the 'Multiple face id' plot for $k = 1$; this is the interesting case

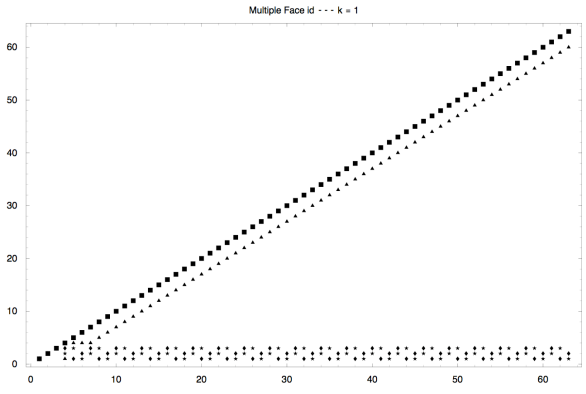

Fig. 18. Plot 'Multiple Face id' for a computation with $k = 1$

of finitely many nodes (three) incremented in their degree infinitely often while infinite nodes are incremented a finite number of times (one). All new nodes are neighbors of two of the three initial nodes, so that the diameter is a constant, and the dimensionality is infinite. But infinite dimensionality for the trinet dual

does not imply infinite dimensionality for the primal – the original trinet –, essentially because a small diameter in a planar graph does not imply a small diameter in its dual. For example, the undirected, planar graph $G(n)$ with nodes $N = \{-1, 0, 1, \ldots, n\}$ and edges

$$\begin{aligned} E = &\{(-1, i) \mid i \in [0, n-1]\} \\ &\cup \{(n, i) \mid i \in [0, n-1]\} \\ &\cup \{(i, i \oplus 1) \mid i \in [0, n-1]\} \end{aligned}$$

where $\oplus$ is summation mod n, has constant diameter 2, while its dual is a planar trinet whose diameter grows linearly with n. The study of the trinet dimensionality for this set of computations is left for further investigation.

The third set of computations, with variable k, appears to differ from the previous cases only quantitatively, and needs no special attention.

Let us finally consider the last case left (see Figure 8, for $k = 2$). In the graph for the last trinet dual, a central node with high degree is clearly visible (see also the upper graphs in Figure 19): this is node 1, whose degree is incremented at every step. We have pointed out earlier that all the other nodes have their

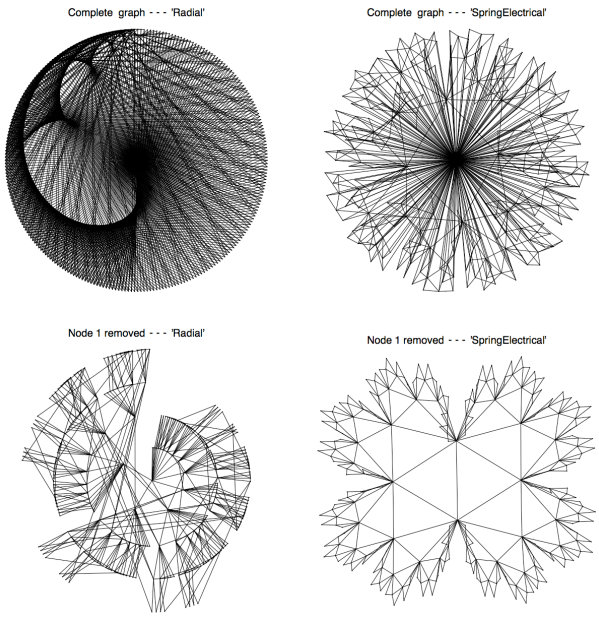

Fig. 19. 'Radial' and 'SpringElectrical' layouts for last trinet dual of a 256-step computation, with $k = 2$, and with node 1 and associated edges included or removed

degrees increased infinitely often, although at a rapidly decreasing rate. This circumstance is perhaps better visualized by the 'radial' layout of Figure 19 (upper-left), which hints at some nested structure.

If we now remove the 'spurious' node 1, and the edges that connect it to all the other nodes, the nested, fractal structure of the graph clearly emerges, at least when using the 'SpringElectrical' layout (lower-right). By connecting back node 1 to all the nodes in this layout, and taking the dual of the resulting graph, we readily get the original trinet, which is a double binary tree with leaves connected in circle, as depicted in Figure 20. The node count of these

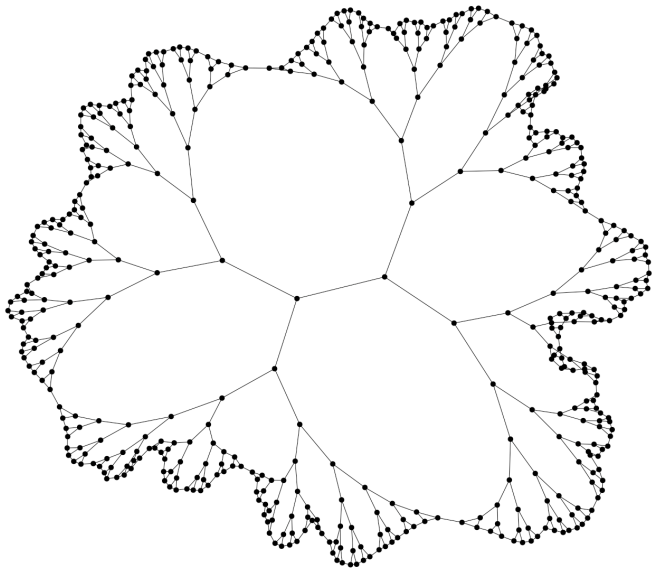

Fig. 20. The nested trinet for $k = 2$ ('SpringElectrical' layout)

trinets grows exponentially with their diameter; and, given any specific (large) trinet, and a specific reference node r in it, the size of the concentric node layers around r grows exponentially too, with minor deviations as r is picked closer to the tree boundary. In this respect, the above computation is useful because it proves that our algorithm can produce trinet-based spaces that go well beyond one dimension.

8 Conclusions

In this paper we have investigated the computing power of planar trivalent networks ('trinets'). On one hand we have proved that they are powerful in a traditional sense, since, when manipulated by appropriate rewrite rules, they are

Turing-complete, and can perform any computation. On the other hand, we have explored their power by the non traditional, experimental type of investigation advocated by Wolfram in [3], and have visualized, by means of ad-hoc indicators, a variety of computational 'shapes' and features. Let us emphasize that this was done by using just *one* graph rewrite rule, called node-tripartition, a very simple deterministic algorithm, and a very simple initial condition, namely a triangle: the contrast between this simplicity and the complexity of shapes that we have exposed is perhaps as remarkable as that observed with elementary cellular automata.

However, the crucial advantage of our trinet-based approach over cellular automata is that space is not assumed, but *created*: while cellular automata can only be regarded as a (precious) metaphor, there seems to be no obvious reasons to exclude that some trinet manipulation algorithm could provide the perfectly accurate, ultimate law of physics.

We believe that an interesting contribution of our paper consists in having introduced an original way both to generate and to observe evolving planar trinets, and their duals, thus achieving some progress in the exploration of a computational space that, according to some researchers, might offer important insights to fundamental physics. Our simple indicators have been quite useful for spotting regularity in several computations, for which plain representation of complete graphs is computationally more costly and visually less effective.

In particular, we point out two results.

1. For parameter $k = 2$ our algorithm yields a specific, regular computation in which infinitely many faces of the growing trinet are visited infinitely often, each time having their polygonal degree incremented by one. In other words, a dynamic network is obtained in which no part is permanently abandoned by the updating process. This property was spotted only once, and is intuitively appealing; it would be interesting to observe it also in random-like computations. It is remarkable that the nested structure of this trinet (Figure 20) be obtained directly by an algorithm that operates only locally, and with control moving on the growing structure only by unit steps.
2. We have found some complex computations that involve transient phases as well as periods in the order of thousands of steps. Just for the sake of comparison, the computation of the well known elementary cellular automaton 110, started from the elementary initial condition of one black cell in an infinite white tape, has a transient of about 2500 steps, and stabilizes to a shifted-period of 240 steps. In this respect we may classify some of our computations as class 4, according to Wolfram's empirical classification scheme.

One highly desirable result that we have not yet obtained is to find class 3 planar trinet computations whose random-like behavior never stabilizes, similar to elementary cellular automaton 30.

In an early section of the paper we have discussed the NKS approach to trivalent network rewriting, and have already pointed out how our work differs from it. In summary, while with pure rewrite systems the location where a

change occurs is implicitly determined by the applicability of one or the other rule, and jumps unpredictably, in our algorithm it is determined explicitly, and moves by unit steps, although the rule is applicable anywhere. In this respect, our approach is more similar to two-dimensional Turing machines, sometimes called 'turmites', which move by unit steps on an infinite grid (see Pegg, Ed Jr. Turmites. http://mathworld.wolfram.com/Turmite.html).

But with turmites, again, space is given, and its static structure is made of stateful cells. In our approach space is created, its topology varies, and its locations – trinet faces, or nodes in the trinet dual – are stateless, although one could perhaps view the arity of these polygons as a local states (of unbounded value!). The most famous turmite is Langton's ant, a 4-state machine moving on an infinite square grid of black and white cells; when placed on a completely white grid, it moves chaotically for about 10,000 steps, and then enters a periodic behavior called 'highway', with shifted-period 104.

A side result of our paper, that might possibly have some practical application still to be investigated, is that every periodic portion of our computations corresponds to a regular tiling of the plane; some of these involve the usage of a finite but large set of different polygons.

In [3] it is implicitly assumed that expansion rules such as node tripartition should be complemented with rewrite rules that decrease the number of nodes in the network. [5] An obvious extension of our work would consist in considering further rewrite rules, while still driving their application by deterministic procedures along the lines we have described.

It may be useful to study variants of our algorithm that can handle trinets with loops and double edges. By dropping the geometric, trinet-based interpretation, and by looking at the algorithm as just an abstract list manipulation procedure, one might perhaps achieve further simplifications, and make it possible to start from even simpler initial conditions.

Acknowledgment. I wish to thank Marco Tarini, at CNR-ISTI, for several lively discussions on the topics covered in this paper.

References

1. K. Zuse, Rechnender raum, Elektronische Datenverarbeitung 8 (1967) 336–344.
2. K. Zuse, Calculating space (rechnender raum), Tech. rep., MIT, Cambridge, Mass., technical Translation AZT-70-164-GEMIT (1970).
3. S. Wolfram, A New Kind of Science, Wolfram Media, Inc., 2002.
4. M. Cook, Universality in elementary cellular automata, Complex Systems 15 (1) (2004) 1–40.
5. L. Smolin, Atoms of space and time, Scientific American (2004) 56–65.
6. T. Bolognesi, Behavioural complexity indicators for process algebra: the NKS approach, Journal of Logic and Algebraic Programming (to appear).

[5] Node-eating rules have been conjectured to be helpful in explaining gravitational attraction.