EXCERPTED FROM

STEPHEN WOLFRAM
A NEW KIND OF SCIENCE

SECTION 7.8

The Problem of Satisfying Constraints

So even if a system at some level follows continuous rules it is still possible for the system to exhibit discrete overall behavior. And in fact it is quite common for such behavior to be one of the most obvious features of a system—which is why discrete systems like cellular automata end up often being the most appropriate models.

The Problem of Satisfying Constraints

One feature of programs is that they immediately provide explicit rules that can be followed to determine how a system will behave. But in traditional science it is common to try to work instead with constraints that are merely supposed implicitly to force certain behavior to occur.

At the end of Chapter 5 I gave some examples of constraints, and I showed that constraints do exist that can force quite complex behavior to occur. But despite this, my strong suspicion is that of all the examples of complex behavior that we see in nature almost none can in the end best be explained in terms of constraints.

The basic reason for this is that to work out what pattern of behavior will satisfy a given constraint usually seems far too difficult for it to be something that happens routinely in nature.

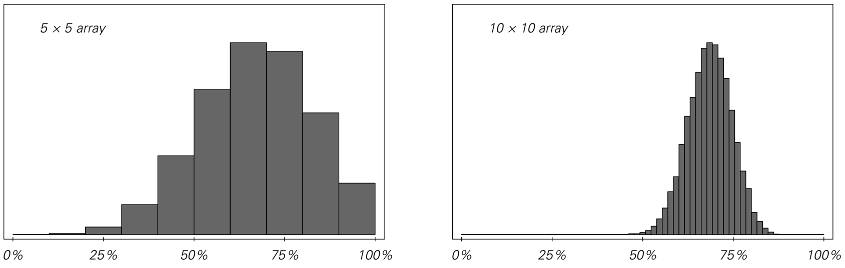
Many types of constraints—including those in Chapter 5—have the property that given a specific pattern it is fairly easy to check whether the pattern satisfies the constraints. But the crucial point is that this fact by no means implies that it is necessarily easy to go from the constraints to find a pattern that satisfies them.

The situation is quite different from what happens with explicit evolution rules. For if one knows such rules then these rules immediately yield a procedure for working out what behavior will occur. Yet if one only knows constraints then such constraints do not on their own immediately yield any specific procedure for working out what behavior will occur.

In principle one could imagine looking at every possible pattern, and then picking out the ones that satisfy the constraints. But even with a 10×10 array of black and white squares, the number of possible patterns is already 1,267,650,600,228,229,401,496,703,205,376. And with a

20 × 20 array this number is larger than the total number of particles in the universe. So it seems quite inconceivable that systems in nature could ever carry out such an exhaustive search.

One might imagine, however, that if such systems were just to try patterns at random, then even though incredibly few of these patterns would satisfy any given constraint exactly, a reasonable number might at least still come close. But typically it turns out that even this is not the case. And as an example, the pictures below show what fraction of patterns chosen at random have a given percentage of squares that violate the constraints described on page 211.



The fraction of all possible patterns in which a certain percentage of squares violate the constraints discussed on page 211. Only a handful of patterns satisfy the constraints exactly (so that 0% of the squares are wrong). For large arrays, the vast majority of possible patterns have about 70% of the squares wrong.

For the majority of patterns around 70% of the squares turn out to violate the constraints. And in a 10 × 10 array the chance of finding a pattern where the fraction of squares that violate the constraints is even less than 50% is only one in a thousand, while the chance of finding a pattern where the fraction is less than 25% is one in four trillion.

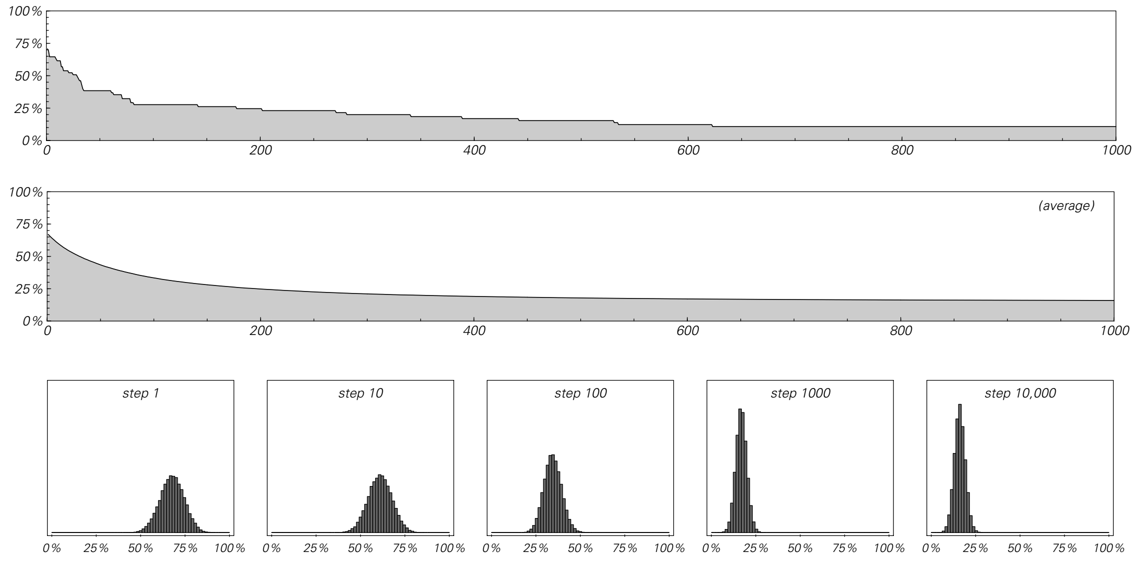
And what this means is that a process based on picking patterns at random will be incredibly unlikely to yield results that are even close to satisfying the constraints.

So how can one do better? A common approach used both in natural systems and in practical computing is to have some form of iterative procedure, in which one starts from a pattern chosen at random, then progressively modifies the pattern so as to make it closer to satisfying the constraints.

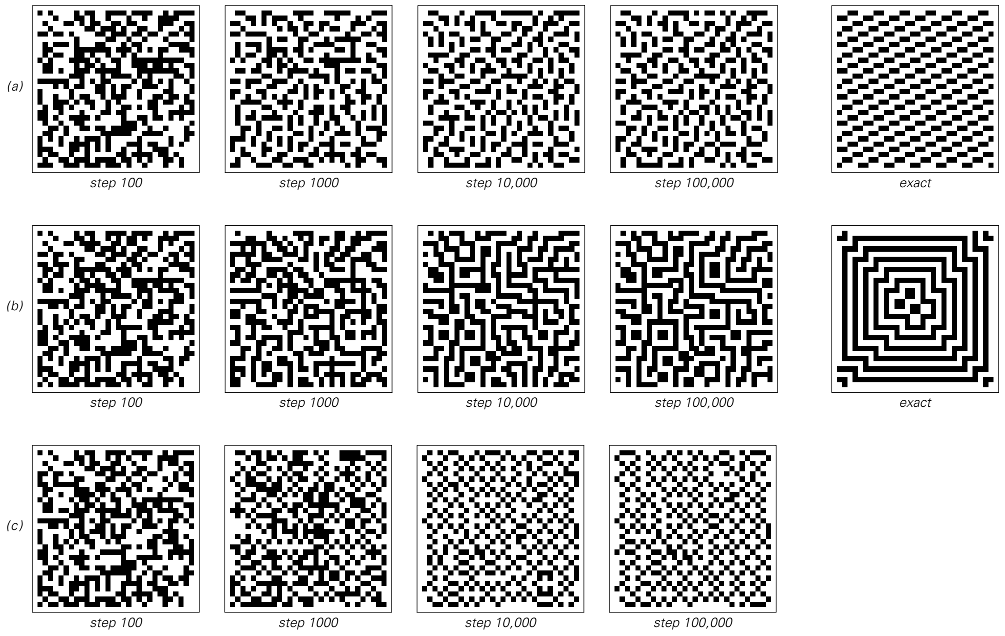
As a specific example consider taking a series of steps, and at each step picking a square in the array discussed above at random, then reversing the color of this square whenever doing so will not increase the total number of squares in the array that violate the constraints.

The picture below shows results obtained with this procedure. For the first few steps, there is rapid improvement. But as one goes on, one sees that the rate of improvement gets slower and slower. And even after a million steps, it turns out that 15% of the squares in a 10 × 10 array will on average still not satisfy the constraints.

In practical situations this kind of approximate result can sometimes be useful, but the pictures at the top of the facing page show that the actual patterns obtained do not look much at all like the exact results that we saw for this system in Chapter 5.



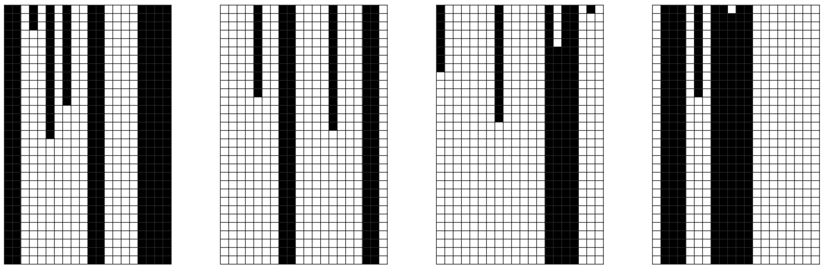
The results of a procedure intended to produce patterns that get progressively closer to satisfying the constraints described on page 211. The procedure starts with a randomly chosen pattern, then at each step picks a square in the pattern at random, and reverses the color of this square whenever doing so does not increase the total number of squares in the pattern that violate the constraints. The top picture shows one particular run of this procedure. The second picture shows the average behavior obtained from many runs. And finally, the bottom picture shows how the fraction of patterns with different percentages of squares violating the constraints changes as the procedure progresses. In all cases 10 × 10 patterns are used.



Patterns generated by using the same procedure as in the previous picture but with three different sets of constraints. Case (a) uses the same constraints as in the previous picture, (b) requires every black square and every white square to have exactly two adjacent black squares, and (c) requires every black square to have 3 adjacent black squares and 1 white square, and every white square to have 4 adjacent white squares. In cases (a) and (b) it is possible to satisfy the constraints exactly; in case (c) it is not. The pictures show the evolution of a 30 × 30 array, which is nearly 10 times the area of the array shown in the previous picture. Although the fraction of squares that violate the constraints is less than 20% after 100,000 steps, the overall patterns still do not look much like the exact results.

So why does the procedure not work better? The problem turns out to be a rather general one. And as a simple example, consider a line of black and white squares, together with the constraint that each square should have the same color as its right-hand neighbor. This constraint will be satisfied only if every square has the same color—either black or white. But to what extent will an iterative procedure succeed in finding this solution?

As a first example, consider a procedure that at each step picks a square at random, then reverses its color whenever doing so reduces the total number of squares that violate the constraint. The pictures at the top of the next page show what happens in this case. The results are

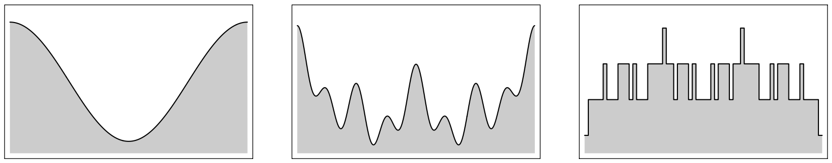


Results of four tries at applying an iterative procedure to find configurations which satisfy the simple constraint that every square should be the same color as the square to its right. (The squares are assumed to be arranged cyclically, so that the right neighbor of the rightmost square is the leftmost square.) The procedure starts from a random configuration of squares, and then at each step picks a square at random, then reverses the color of this square whenever doing so reduces the total number of squares that violate the constraint. The only configurations that ultimately satisfy the constraints are all white and all black. But the procedure gets stuck long before it reaches these configurations. The problem is that for any block more than one square across changing the color of a square at either end will not reduce the total number of squares that violate the constraint. And as a result, such blocks remain fixed and cannot disappear.

remarkably poor: instead of steadily evolving to all black or all white, the system quickly gets stuck in a state that contains regions of different colors.

And as it turns out, this kind of behavior is not uncommon among iterative procedures; indeed it is even seen in such simple cases as trying to find the lowest point on a curve. The most obvious iterative procedure to use for such a problem involves taking a series of small steps, with the direction of each step being chosen so as locally to go downhill.

And indeed for the first curve shown below, this procedure works just fine, and quickly leads to the lowest point. But for the second



Three examples of curves. In the first case, the most obvious mechanical or mathematical procedure of continually going downhill will successfully lead one to the lowest point. But in the other two cases, this procedure will usually end up getting stuck at a local minimum. This is the basic phenomenon which makes it difficult to find patterns that satisfy constraints exactly using a procedure that is based on progressive improvement. The third picture above is a representation of the kind of curve that arises in almost all discrete systems based on constraints.

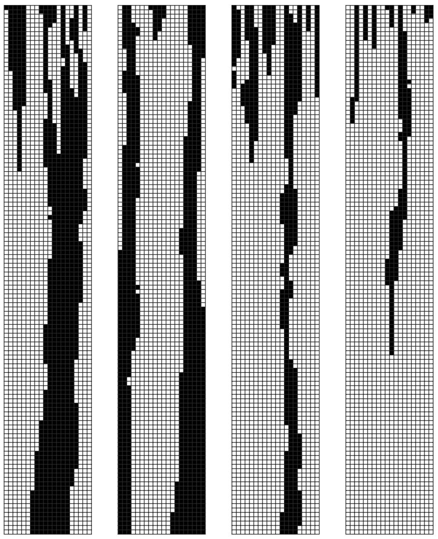
curve, the procedure will already typically not work; it will usually get stuck in one of the local minima and never reach a global minimum.

And for discrete systems involving, say, just black and white squares, it turns out to be almost inevitable that the curves which arise have the kind of jagged form shown in the third picture at the bottom of the facing page. So this has the consequence that a simple iterative procedure that always tries to go downhill will almost invariably get stuck.

How can one avoid this? One general strategy is to add randomness, so that in essence one continually shakes the system to prevent it from getting stuck. But the details of how one does this tend to have a great effect on the results one gets.

The procedure at the top of the facing page already in a sense involved randomness, for it picked a square at random at each step. But as we saw, with this particular procedure the system can still get stuck.

Modifying the procedure slightly, however, can avoid this. And as an example the pictures below show what happens if at each step one reverses the color of a random square not only if this will decrease the total number of squares violating the constraints, but also if it leaves this number the same. In this case the system never gets permanently stuck, and instead will always eventually evolve to satisfy the constraints.



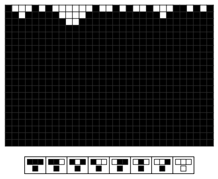
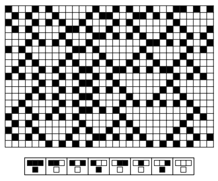
Results from a slight modification to the procedure used in the picture at the top of the facing page. A random square is again picked at each step. But now the color of that square is reversed not only if doing so actually changes the total number of squares that violate the constraint, but also if it leaves this number the same. With this procedure, evolution from any initial condition can visit every possible configuration, so that the configurations which satisfy the constraints will at least eventually be reached.

But this process may still take a very long time. And indeed in the two-dimensional case discussed earlier in this section, the number of steps required can be quite astronomically long.

So can one speed this up? The more one knows about a particular system, the more one can invent tricks that work for that system. But usually these turn out to lead only to modest speedups, and despite various hopes over the years there seem in the end to be no techniques that work well across any very broad range of systems.

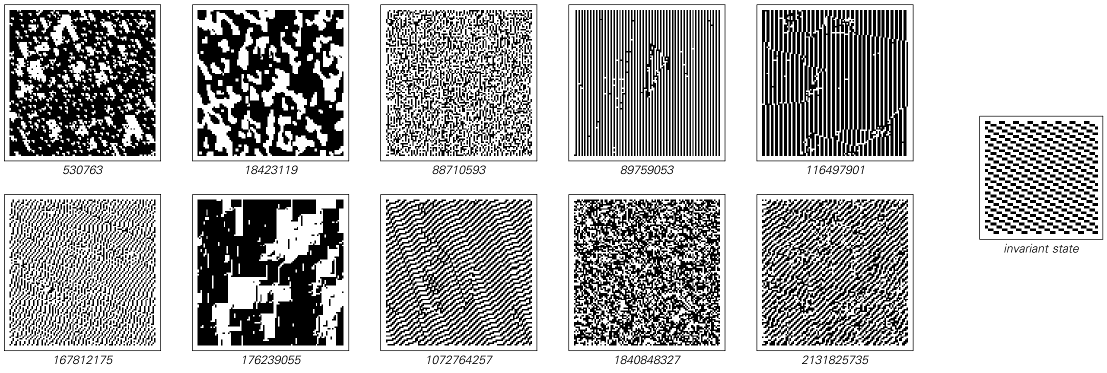
So what this suggests is that even if in some idealized sense a system in nature might be expected to satisfy certain constraints, it is likely that in practice the system will actually not have a way to come even close to doing this.

In traditional science the notion of constraints is often introduced in an attempt to summarize the effects of evolution rules. Typically the idea is that after a sufficiently long time a system should be found only in states that are invariant under the application of its evolution rules. And quite often it turns out that one can show that any states that are invariant in this way must satisfy fairly simple constraints. But the problem is that except in cases where the behavior as a whole is very simple it tends not to be true that systems in fact evolve to strictly invariant states.



Two of the 28 elementary cellular automata whose only invariant states are uniform in color. In the first case one of these invariant states is always reached; in the second it is not.

The two cellular automata on the left both have all white and all black as invariant states. And in the first case, starting from random initial conditions, the system quickly settles down to the all black invariant state. But in the second case, nothing like this happens, and instead the system continues to exhibit complicated and seemingly random behavior forever.

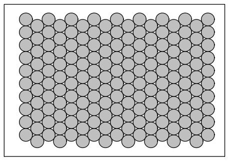
The two-dimensional patterns that arise from the constraints at the end of Chapter 5 all turn out to correspond to invariant states of various two-dimensional cellular automata. And so for example the pattern of page 211 is found to be the unique invariant state for 572,522 of the 4,294,967,296 possible five-neighbor cellular automaton rules. But if one starts these rules from random initial conditions, one typically never gets the pattern of page 211. Instead, as the pictures at the top of the facing page show, one sees a variety of patterns that very



Typical behavior of two-dimensional cellular automata that leave only the pattern on the right invariant. The results shown come from 500 steps of evolution starting from random initial conditions. In no case does the global behavior seen come even close to satisfying the simple constraints that determine the invariant state.

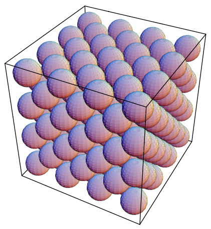
much more reflect explicit rules of evolution than the constraint associated with the invariant state.

So what about actual systems in physics? Do they behave any differently? As one example, consider a large number of circular coins pushed together on a table. One can think of such a system as having an invariant state that satisfies the constraint that the coins should be packed as densely as possible. For identical coins this constraint is satisfied by the simple repetitive pattern shown on the right. And it turns out that in this particular case this pattern is quickly produced if one actually pushes coins together on a table.



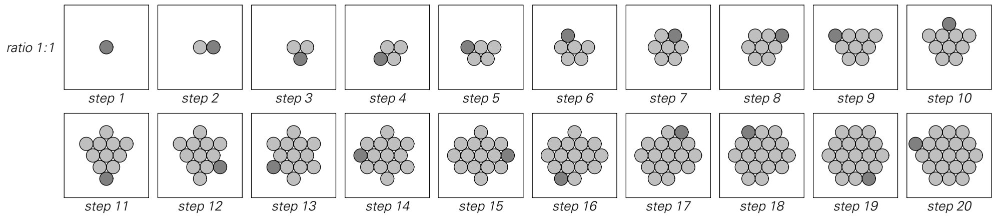
The densest packing of identical circles in the plane. Each circle is surrounded by six others.

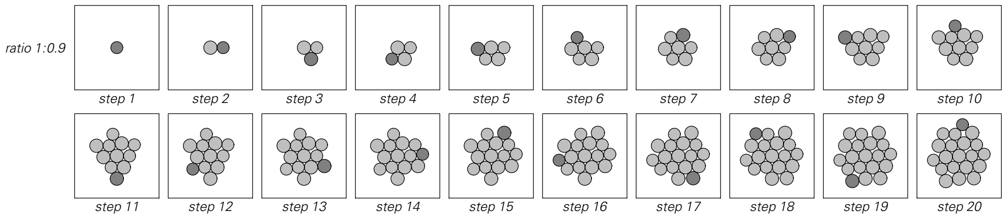
But with balls in three dimensions the situation is quite different. In this case the constraint of densest packing is known to be satisfied when the balls are laid out in the simple repetitive way shown on the right. But if one just tries pushing balls together they almost always get stuck, and never take on anything like the arrangement shown. And if one jiggles the balls around one still essentially never gets this arrangement. Indeed, the only way to do it seems to be to lay the balls down carefully one after another.

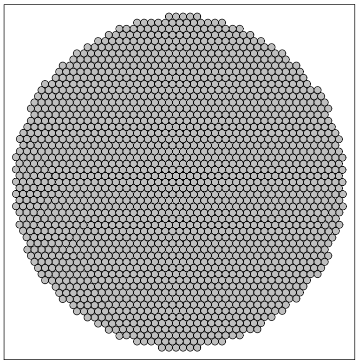


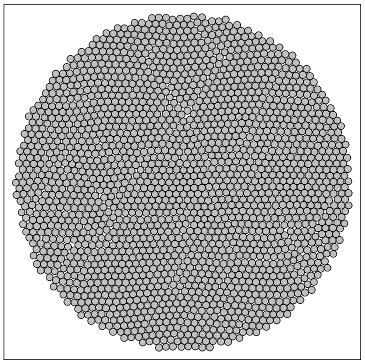
The densest packing of identical spheres in three-dimensional space. Each sphere is surrounded by 12 others.

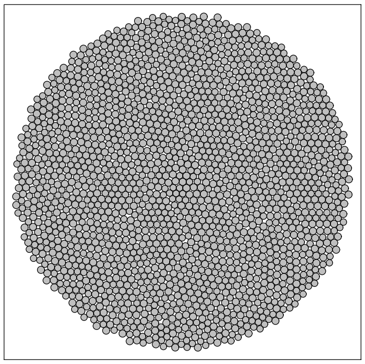
In two dimensions similar issues arise as soon as one has coins of more than one size. Indeed, even with just two sizes, working out how to satisfy the constraint of densest packing is already so difficult that in most cases it is still not known what configuration does it.

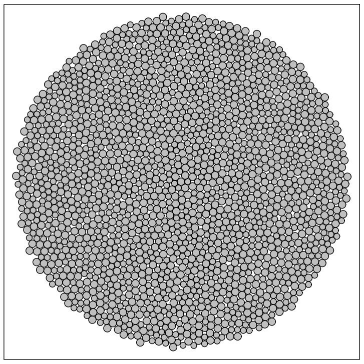


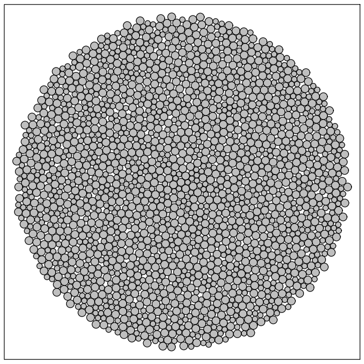


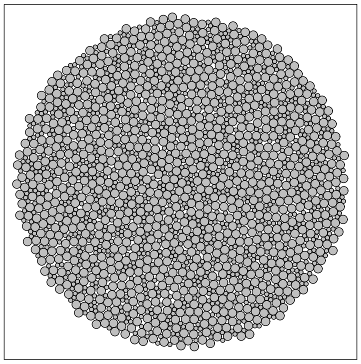

ratio 1:1


ratio 1:0.95


ratio 1:0.9


ratio 1:0.8


ratio 1:0.7


ratio 1:0.5

Patterns obtained by successively laying down circles in such a way that the center of each new circle is as close as possible to the center of the first circle. Except in the very first case, the extent to which these represent the densest possible packings is not clear, and indeed it is quite possible that in most such actual packings circles of different sizes are just separated into several uniform regions.

The pictures on the facing page show what happens if one starts with a single circle, then successively adds new circles in such a way that the center of each one is as close to the center of the first circle as possible. When all circles are the same size, this procedure yields a simple repetitive pattern. But as soon as the circles have significantly different sizes, the pictures on the facing page show that this procedure tends to produce much more complicated patterns—which in the end may or may not have much to do with the constraint of densest packing.

One can look at all sorts of other physical systems, but so far as I can tell the story is always more or less the same: whenever there is behavior of significant complexity its most plausible explanation tends to be some explicit process of evolution, not the implicit satisfaction of constraints.

One might still suppose, however, that the situation could be different in biological systems, and that somehow the process of natural selection might produce forms that are successfully determined by the satisfaction of constraints.

But what I strongly believe, as I discuss in the next chapter, is that in the end, much as in physical systems, only rather simple forms can actually be obtained in this way, and that when more complex forms are seen they once again tend to be associated not with constraints but rather with the effects of explicit evolution rules—mostly those governing the growth of an individual organism.

Origins of Simple Behavior

There are many systems in nature that show highly complex behavior. But there are also many systems that show rather simple behavior—most often either complete uniformity, or repetition, or nesting.

And what we have found in this book is that programs are very much the same: some show highly complex behavior, while others show only rather simple behavior.

Traditional intuition might have made one assume that there must be a direct correspondence between the complexity of observed behavior and the complexity of underlying rules. But one of the central discoveries of this book is that in fact there is not.