Self-Organizing Switchbox Routing on an Adaptive Resistor Grid

Allen L. Barker
Technical Report CS-97-28
University of Virginia
Computer Science Department

Dec. 13, 1997

Abstract

The switchbox routing problem arises in the fabrication of computer chips, and is representative of a general class of routing problems. We describe a self-organizing algorithm for solving certain instances of the switchbox routing problem. The method is based on path formation via a positive feedback process, with competitive interactions. We define such a process on a grid of adaptive, variable resistors and simulate its dynamics. The method is applied to several problem instances.

1 Introduction

In this paper we present a self-organizing algorithm for solving certain instances of the switchbox routing problem. A global solution to the switchbox routing problem is computed by sets of nodes operating locally with fairly simple, but well-tuned, rules. The solution is an emergent property of these local interactions.

The method is based on path formation via a positive feedback process. Consider an analogy to soil erosion. The height of the soil at a particular location decreases as the the flow of water over it increases, i.e., as the water carries away soil particles. At the same time, the flow of water at a point tends to increase as the level of the soil decreases there. That is, water tends to flow toward lower points, and the more water that flows at a point the more that point is lowered by erosion. The positive feedback interaction between these properties tends to cause channels to be cut into the soil, resulting in the erosion patterns of rivers, ravines, canyons, etc. Another analogy one can make is to the formation of deer paths. As more deer follow a particular path, it tends to become packed down and freer of vegetation than the surrounding woods. As the path becomes easier to traverse, more deer tend to follow it, and so forth.

These general sorts of processes occur in many situations, and can be fairly robust with regard to the exact properties of the interacting processes. We have implemented such a feedback process on a (simulated) resistor grid with current flowing through it. As more current flows through a resistor, its resistance value decreases. One reason for choosing this electrical analogy is that it facilitates an intuitive understanding of what is going on. It also allows us to apply the well-known current and voltage laws and to make use of properties such as the conservation of current through a network. It might also allow for efficient, partially analog implementations using special hardware.

2 The Switchbox Routing Problem

The switchbox routing problem arises in the fabrication of computer chips, where certain components need to be connected to other components. That is, the connections between input and output pins of electronic components are specified by a circuit de-

sign. The switchbox routing problem arises as part of an automated system to fabricate these connections onto two (or possibly more) conductive layers. The two layers are chip layers onto which conductive paths are etched. The problem is representative of a more general class of routing problems which occur in widely varied applications.

In the basic switchbox routing problem, one is given an $n \times m$ box, with **terminals** specified along the outside edges of the box. In this paper we assume the box has two layers. The terminals are numbered (or lettered) from 1 to some maximum number of terminal types. All terminals of the same type, i.e., with the same number, are said to belong to the same **net**. Looking ahead to Figure 3 gives an idea of the type of box structure (and graph topology) we are talking about, though the terminals around the edges are not all shown.

For a more formal description we refer the reader to, e.g., [CH88]. A simple example should make the idea clear. Consider Figure 1. This is a representation of a very simple instance of the switchbox routing problem. The pairs of letters around the outside of the box indicate the terminal types, or nets. (The plus signs are points around the edge with no terminals.) The points labeled 1 inside the box represent all the nodes on the first layer. Similarly, the points labeled 2 represent all the nodes on the second layer. A node on layer 1 is connected to all the points to its north, south, east, and west. It is also connected to the point on layer two that is directly "above" it, and this type of connection is called a **via**. The nodes on layer 2 are similarly connected. Again, looking ahead to Figure 3 shows the type of graph structure we are dealing with.

The goal is to label all the nodes of the graph with at most one terminal type. This labeling should be such that each terminal is connected by a path through the grid, labeled with its own type, to all other terminals of its same type. That is, all terminals of a common net are connected through the switchbox.

Figure 2 shows a solution to the switchbox problem of Figure 1. Nodes which are not used are still labeled 1 or 2, and the others are labeled with terminal letters. Notice that each terminal on one of the

```
   aa ++ bb ++ cc
++ 12 12 12 12 12 ++
++ 12 12 12 12 12 bb
++ 12 12 12 12 12 ++
++ 12 12 12 12 12 ++
   bb cc ++ aa ++
```

Figure 1: An instance of a switchbox routing problem.

```
   aa ++ bb ++ cc
++ 1a 12 1b 12 c2 ++
++ 1a 12 1b 1b cb bb
++ 1a aa ab aa c2 ++
++ 1b cb cb ca c2 ++
   bb cc ++ aa ++
```

Figure 2: A solution to the switchbox routing problem above.

the outside edges has a path, through the grid, to all other external terminals of its same type (connecting to either of the pair of edge terminals is sufficient here). For example, terminal **a** on the north of the switchbox first gets routed three nodes south on layer 2. It is then routed one node east on layer 2. At this point it is routed down to layer 1, then two nodes east on layer 1, back up to layer 2, and finally two nodes south to connect up with the **a** terminals on the southern edge of the switchbox.

A solution does not necessarily exist for a given instance of the switchbox routing problem. When multiple solutions exist there are a variety of criteria, such as total path length and the number of vias, for preference-ordering the solutions. Our method is based on "paths of least resistance," and so should perform well relative to total path length. Vias can also be given additional resistive weighting, if desired. In this paper we focus on finding valid solutions.

See [CH88] for more information and references on traditional switchbox routing algorithms. See, e.g., [VM90] for work on switchbox routing using specially constructed hardware. See [SCF91] and [SF91] for neural network approaches to routing problems. An approach to module placement based on a resistive network analogy is described in [CK84].

3 Adaptive Resistor Grids (Slabs)

The basic building block of our self-organizing router is a grid of resistors, which we call a **slab**. An example is shown in Figure 3. A fixed amount of current, I, is pumped into the slab at a current source, and the same amount of current is drawn out of the slab in equal portions from every current sink. The resistor grid has the same $n \times m \times 2$ structure as the particular problem instance. The current sources and current sinks correspond to the terminals of a net. For example, the slab of Figure 3 corresponds to the net labeled **b** in the example of Figure 1. This will be described in more detail in the next section.

The points of the grid where three or more wires meet are called **nodes**. All the resistors grouped around a node in the grid are associated with a common resistance value, though resistance values in certain directions may be weighted differently. The common resistance value at a node is assumed to be modifiable.

An effective heuristic for switchbox routing in two layers is to route most north-south connections on layer 1, and most east-west connections on layer 2. To implement a variant of this heuristic, we weight all east-west resistances on layer 1, and north-south resistances on layer 2, by an extra factor of 8. For example, a node on layer 1 with a node resistance value of 3 would have resistors of resistance 3 in the north, south, and up (via) directions, and resistors of resistance 24 in the east and west directions.

To simulate the current flow and voltages in a resistor grid we iteratively update Kirchoff's current law at all the nodes until a given accuracy is obtained. In our simulations we set the current to $I = 1$. We repeatedly iterate the nodal current laws over all the nodes of a slab until the largest change in the summed current into some node, on an iteration, is less than 0.01. These updates consume by far the bulk of the computation time in our simulations. Sparse matrix techniques could be applied, but since the weights will be modified only slightly between solutions relaxation methods may be preferable. More efficient relaxation techniques could be employed.

The basic intuitive notion of a slab is that a fixed amount of current is pumped in, and that current must flow out in equal parts from each current sink. The current will be maximum along some path, and the highest current paths will have their resistance values decreased more than others. When resistance is defined to decrease as current increases through a node, path formation results. A slab operating in isolation according to these rules (to be detailed later) will tend to converge to a state where a single path (tree) connects all its terminals.

4 Combining Slabs Into a Switchbox Router

We now describe how slabs are combined to form a complete router. The main constraint we need to incorporate is that, in a valid solution, only one net can occupy any node of the grid.

In Figure 4 we show how slabs are assigned to terminals. Each terminal has a slab associated with it, and these slabs are grouped according to the particular net the terminal belongs to. The illustration of Figure 4 is for the example switchbox of Figure 1.

This stacking of the slabs is significant because nodes of each slab will compete with other nodes occupying the same position in other slabs of the stack. This "vertical" collection of nodes will be called a **column**. For example, all the nodes on layer 2 and in the northeast corner of some slab (ignoring sources and sinks) form a column.

In order to incorporate the constraints, we implement a winner-take-all competition between the nets in each column. That is, the slabs of at most one net can "control" the column in a valid solution. To do this we need to a single value for each net within a column. These values will be called **weight values**. Note that there is a weight value associated with each net, in each column.

The weight values are fundamental to our algorithm in several ways. First, they define how we produce a trial solution from a given configuration. To produce a trial solution from a set of slabs we assign each grid node of the trial solution to the net with

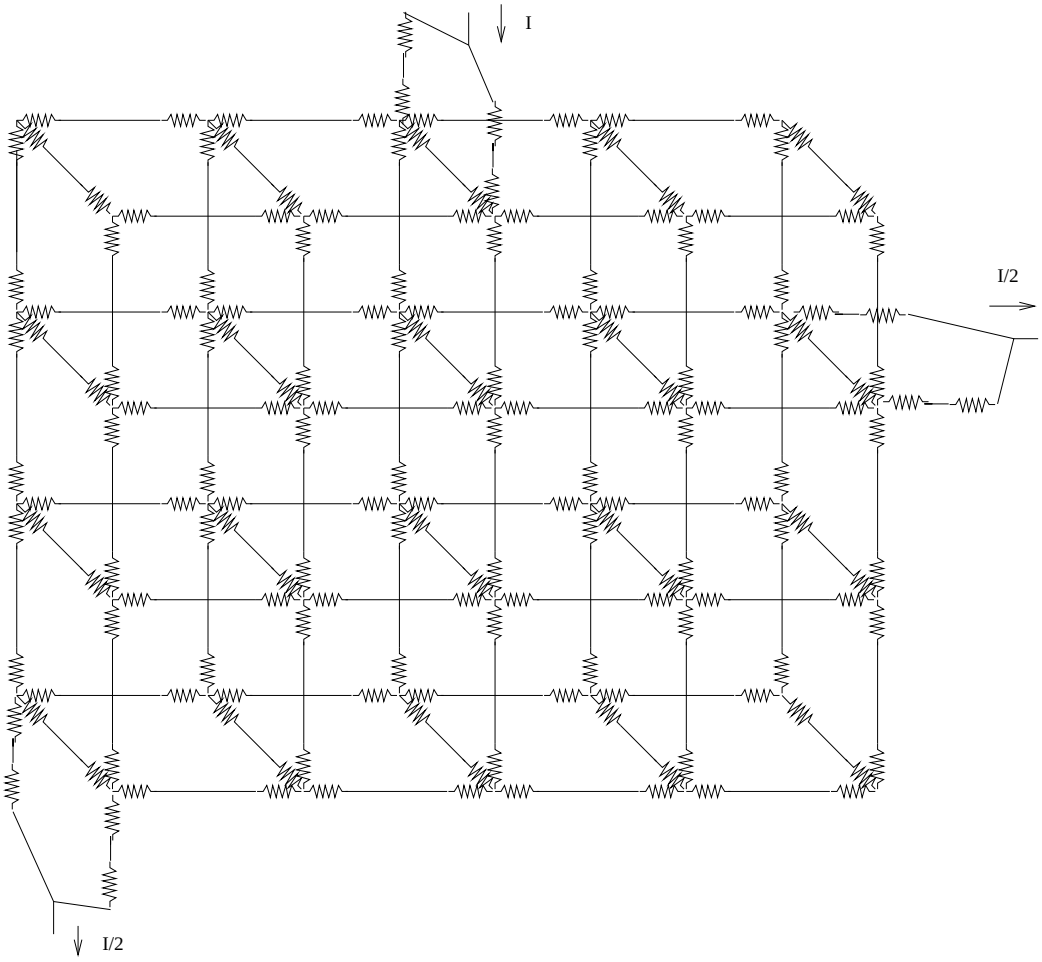

Figure 3: A slab of resistors (for the north terminal of net **b** in the example switchbox in Figure 1).

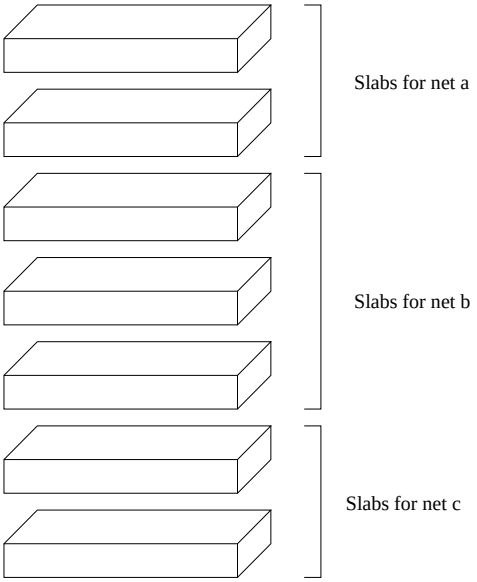

Figure 4: A stack of slabs for the example switchbox.

the highest corresponding weight value.

Secondly, weight values at a node are directly related to resistance values at a node according to

$$r_{ij} = -log(1 - w_{ij}), \tag{1}$$

where w_{ij} is the weight value for net i at node j in the grid, and r_{ij} is the corresponding resistance value. All resistance values at nodes in column j corresponding to net i are assigned this same value, r_{ij}. The curve is shown in Figure 5. Thus when the weight value approaches 1, the resistance value approaches 0, and when the weight value approaches 0, the resistance value approaches infinity.

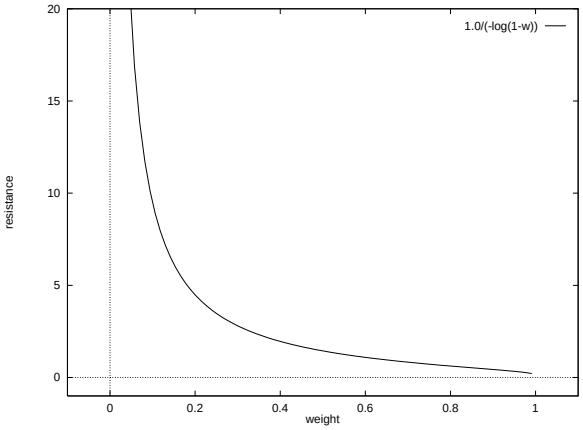

Figure 5: The function converting weight values to resistance values.

Finally, the weight values are made to compete against each other to incorporate the problem constraints. There are many possible ways to set up this weight competition. In the following paragraphs we first describe how the weight values are calculated and dynamically updated. We then describe the particular winner-take-all formulation we use.

4.1 Calculating and Updating the Weight Values

The weight calculations and updates are based on computing the sum of the absolute current values into each node. These values are then summed over all nodes of the same net in a column. This value will be called Δ_{ij}, where i specifies a net whose slabs are being summed over and j specifies a particular node position in the slabs (column) that is being summed over.

Figure 6 illustrates the summation process for computing Δ values and how it works for paths which have converged into single paths. The idea is to allow nets with different numbers of terminals to compete fairly against each other — especially near convergence. We always take the current I into each slab's source to be set to 1, and draw equal amounts of current from each sink.[1] To see how this tends to allow nets with different numbers of terminals to compete fairly against each other, consider the cases for two terminals, four terminals, etc. The sums tend to be 4 on converged trees (recall that the absolute sum of all current into and out of a node is being calculated, so a node on a path carrying a current of 1 gets a score of 2 for current both into and out of the node). Any values above 4 are then set to 4 (to deal with any special cases that do not follow the rule) and the values are divided by 4 to yield a Δ value between 0 and 1.

The delta values are thus computed as

$$\Delta_{ij} = \left(\frac{1}{4} \sum_{\substack{nodes\ of \\ net\ i\ in \\ column\ j}} \sum_{\substack{wires \\ into \\ node}} \text{abs(wire current)} \right)^{power}, \tag{2}$$

where we have not explicitly notated the "floor" at 4. In this paper we have used $power = 1$, but it is included for generality.

The weight values are updated as

$$w_{ij} = (1 - \epsilon)\ w_{ij} + \epsilon\ \Delta_{ij}. \tag{3}$$

Note that if a Δ_{ij} value is constant then the exponential averaging will gradually cause w_{ij} to approach Δ_{ij}. The rate of the convergence is specified by a value ϵ between 0 and 1. After each weight update the weights are normalized.

4.2 Normalizing the Weight Values

There are many different types of winner-take-all networks and methods, and different ones are appropriate in different contexts. The winner-take-all normalization we use is performed as

$$w_{ij} = \frac{w_{ij}}{\left(\sum_i w_{ij}^{\alpha}\right)^{1/\alpha}}. \tag{4}$$

That is, the Minkowski norm is fixed to 1 across column weight values for different nets.[2] (For example, choosing $\alpha = 2$ fixes the Euclidean norm, and $\alpha = 1$ fixes the L1 norm since all weights are positive.) Weight values are initially all set to 0.5 and then normalized. They are renormalized after each weight update.

The weight values of all nodes in a column belonging to a common net are identical by definition. Thus, for example, in Figure 4, all of the slabs for net **a** have identical weight values. While the weight values, and hence the resistances, are identical, the currents and voltages are not.

An exception to the weight normalization occurs for sources and sinks. In this case, the normalization occurs only within a slab. The weights on layer 1 and layer 2 are normalized for each pair of source or sink weights. (The Minkowski norm for sources and sinks is fixed at 3×10^6 rather than 1 because this seems to improve the performance of the simple iterative update algorithm on slab currents and voltages.)

4.3 Selecting Parameters

We now need to select a set of parameters. We chose parameters that would work on the difficult switchbox example, to be discussed later. Here we give a

[1] If a single sink can draw all the current then disconnected nets can result. For example with four terminals we might get $1 \rightarrow 2$ and $3 \rightarrow 4$ without 1 being connected to 3.

[2] Another variant is to renormalize if the norm is greater than 1, but leave it alone otherwise. This strengthens the tendency to form individual paths, since more current must be committed to "capture" a node.

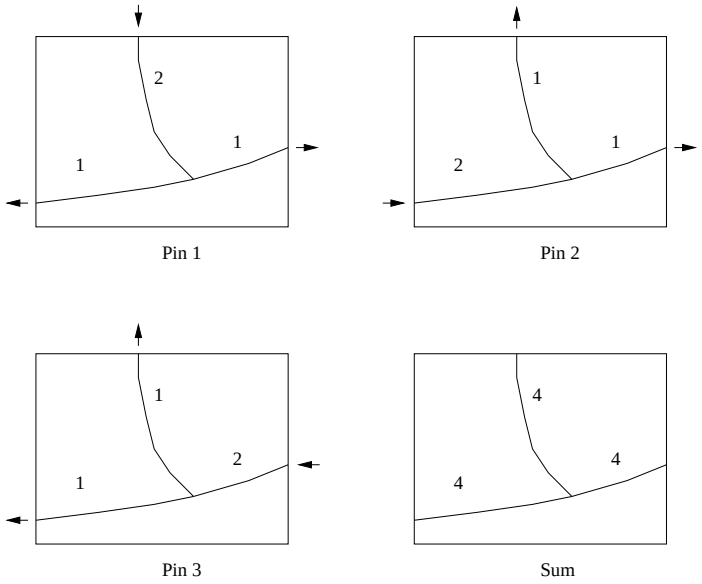

Figure 6: Current on three slabs after path formation, and their sums.

brief description of the subjective effects of the different parameters, and the values we have used in this paper. (These descriptions should be clearer after reading the Examples section.)

The power α in the Minkowski norm is set to 0.5. The idea here is to discourage routing paths for different nets from sharing nodes. For example, with $\alpha = 0.5$, the vectors $(0.25, 0.25)$ and $(0, 1)$ both have Minkowski norm of 1. Thus, unlike in the L1 norm and Euclidean norm, the total amount of weight available decreases when the weight is "shared" across vector components. Intuitively, this tends to cause the current associated with a net to spread out and "search" for paths where nodes need not be shared. When set too low, though, there is a tendency for firm paths not to coalesce.

The *power* parameter in the Δ computation influences the tendency to path formation. Values greater than 1 tend to increase path formation by increasing the advantage given to the nets with the highest current values. Setting this parameter too high, though, can cause too rapid a "crystallization" into an invalid solution. We have used a value $power = 1$.

As previously described, we weight the resistances in non-preferred directions by a factor of 8. Decreasing this value tends to weaken the tendency to path formation. Increasing it tends to straighten out paths, but may interfere in cases where this is not the required behavior.

5 Examples

In this section we present some examples of running the routing algorithm on some standard instances of the switchbox routing problem. The basic simulation cycle is to calculate the currents and voltages for all slabs, update and normalize the weights, and then compute the new resistance values from these weight values. The examples are all taken from [CH88], and that paper cites their original appearance in the literature.

The **sample switchbox** is shown in Figure 7. It is a 7×8 switchbox with 6 nets. The trial solution after 40 resistance update cycles is shown in Figure 8. Recall that a trial solution is produced by assigning each node to the net with the largest corresponding weight value. After 40 cycles the trial solution is

```
   ++ ++ ++ cc ff ee cc
aa 12 12 12 12 12 12 12 ++
++ 12 12 12 12 12 12 12 ++
++ 12 12 12 12 12 12 12 dd
dd 12 12 12 12 12 12 12 ++
++ 12 12 12 12 12 12 12 ++
++ 12 12 12 12 12 12 12 bb
++ 12 12 12 12 12 12 12 aa
   ++ bb cc ff ++ ee ++
```

Figure 7: The sample switchbox problem instance.

```
   ++ ++ cc ff ee cc ++
aa aa aa cc fc ec cc ee ++
++ aa aa cc fc ec cc ad ++
++ ad ad cd fd ed ed dd dd
dd dd ad cd fd ed ed ad ++
++ ad bb cb fb eb eb ab ++
++ aa bb cb fb eb eb ab bb
++ aa bb cb fb ea ea aa aa
   ++ bb cc ff ++ ee ++
```

Figure 8: The sample switchbox after 40 update cycles.

not yet valid. In Figure 9 we show the weight values (multiplied by 100) for the **a**-net slabs after 40 cycles. The indented rows are weight values on layer 2, and the unindented rows are the weight values on layer 1. This figure illustrates how various different paths are being explored on a "subconscious" level, though they are not all reflected in the trial solution.

A valid solution to the sample switchbox is found by 80 cycles. In Figure 10 we show the solution found after 200 cycles, and in Figure 11 we show the corresponding weight values for the **a**-slabs. In Figure 12 those same weight values are shown after 800 cycles, and can be seen to be converging to distinct paths.

The **difficult switchbox** is shown in Figure 13, along with the invalid trial solution found after 2000 cycles. Notice that nets **d** and **g** are not yet routed correctly. A valid solution, found after 4200 cycles, is shown in Figure 14.

The **more difficult switchbox** is identical to the difficult switchbox, but with the rightmost column removed. The self-organizing switchbox router with the current parameter settings fails to converge to a

```
  31   18   09   03   02   01   01   02
09   10   05   01   01   01   00   02
       03   03   03   02   02   01   02
     07   04   01   01   01   01   02
       02   02   02   02   01   01   01
     05   03   01   01   01   01   02
       01   02   02   02   02   02   02
     04   03   01   01   01   01   03
       02   02   02   02   02   02   02
     03   02   01   01   02   02   05
       02   02   02   02   01   01   01
     02   01   01   01   02   02   06
       02   02   02   03   04   07   15   31
     02   01   01   01   02   02   08   09
```

Figure 9: Weight values for net **a** after 40 update cycles.

```
   ++ ++ cc ff ee cc ++
aa aa aa cc fc ec cc aa ++
++ aa aa cc fc ec cc aa ++
++ ad ad cd fd ed ed ad dd
dd ad ad cd fd ed ed ad ++
++ aa aa ca fa ea ea aa ++
++ aa bb cb fb eb eb ab bb
++ aa ba ca fa ea ea aa aa
   ++ bb cc ff ++ ee ++
```

Figure 10: The solution after 200 update cycles.

```
  65   56   29   05   04   03   02   07
01   37   19   01   00   00   00   06
       09   11   06   06   05   04   09
     29   18   01   01   01   01   12
       03   04   03   02   02   01   01
     19   14   01   01   01   01   10
       01   03   03   03   03   03   04
     17   13   01   01   02   01   12
       08   11   10   10   08   07   10
     21   11   01   01   03   02   21
       07   03   02   02   02   01   01
     15   02   01   01   04   02   24
       13   10   11   13   14   19   42   65
     12   01   01   01   05   02   28   01
```

Figure 11: Weight values for net **a** after 200 update cycles.

```
   pp ++ cc ee mm hh gg jj ff ii nn pp oo pp ++ vv uu bb cc tt bb ss ++
++ pp cp cp ep mp hp gp jp fp ip np pp op pp pp vp up bp cp tp bp sp sp pp
++ pp cc cc ec mc hc gc jc fc ic nc pc oc pc pc vc uc bb cb tb bb sd dd dd
oo oo co co eo mo ho go jo fo io no po oo pc pc vc uc bc cc tt dt st dt tt
nn cn cn cn en mn hn gn jn fn in nn pp pp aa pa va ua ba ca da da sa da aa
ll ll cl cl el ml hl gl jl fl il ll ll pa aa pa va ua ba cu dd dd sd dd ++
aa aa ca ca ea ma ha ga ja fa ia la la pa aa pa va uu bu cu du du su uu uu
bb ab cb cb eb mb hb gb jb fb ib lb lb pb ab pb vb ub bb cu du du ss us ss
jj aj cj cj ej mj hj gj jj fj ij jj ll pl al pl vl ul bl cu du du su uu uu
cc ac cc cc ec mc hc gc jc fc ic jj ll pl al pl vl vl bl cl dl dl sl sl ll
rr ar rr mm em mm hg gc jc fc ic jc pc pc ac pc vv vv bv cv dv dv sv sv vv
mm am rm mm em mm hm gm jm fm im jm mm pd ad pd cd dd bd cd dd dd ss ss ss
qq aq rq qq ed hh hh gf jf ff id jd md pd ad pd cd xd bx cx xx xx sx sx xx
ee ae re qe ee hd gd ff jd id id jd md pd ad pd cc xc bc cc xc xc sc sc cc
kk ak rk qk ek hk gk fk jk ik ik jk mk pk ak pk kk xx bx xx xx ww sw sw ww
dd ad rd qd ed hd gd fd jd id dd jd md pp ap pp kd xd bd dd dd ww ss ss ss
   aa rr qq ee hh gg ff jj ii ++ jj mm pp aa pp kk xx bb ++ ++ ww ss ++
```

Figure 13: An invalid solution to the difficult switchbox after 2000 cycles.

```
   pp ++ cc ee mm hh gg jj ff ii nn pp oo pp ++ vv uu bb cc tt bb ss ++
++ pp cp cp ep mp hp gp jp fp ip np pp op pp pp vp up bp cp tp bp sp sp pp
++ pp cc cc ec mc hc gc jc fc ic nc pc oc pc pc vc uc bb cb tb bb sd dd dd
oo co co co eo mo ho go jo fo io no po oo pc pc vc uc bc cc tt bt st dt tt
nn cn cn cn en mn hn gn jn fn in nn pp pp aa pa va ua ba ca da ba sa da aa
ll ll cl cl el ml hl gl jl fl il ll ll pa aa pa va ua ba ca dd dd sd dd ++
aa aa ca ca ea ma ha ga ja fa ia la la pa aa pa va uu bu cu du du su uu uu
bb ab cb cb eb mb hb gb jb fb ib lb lb pb ab pb vb ub bb cu du du ss us ss
jj aj cj cj ej mj hj gj jj fj ij jj ll pl al pl vl uu bu cu du du su uu uu
cc ac cc cc ec mh hh gc jc fc ij jj ll pl al pl vl vl bl cl dl dl sl sl ll
rr ar rr mm em mm hg gg jf ff ic jc pc pc ac pc vv vv bv cv dv dv sv sv vv
mm am rm mm em hm hm gm jm fm im jm mm pd ad pd cd dd bd cd dd dd ss ss ss
qq aq rq qq ed hh gh gf jf ff ii jd md pd ad pd cd xx bx cx dx xx sx sx xx
ee ae re qe ee hd gg ff jf if id jd md pp ap pp cc xc bc cc dc xc sc sc cc
kk ak rk qk ek hk gk fk jk ik ik jk mk pk ak pk kk xx bx dx dx ww sw sw ww
dd ad rd qd ed hd gd fd jd id dd jd md pd ad pd kd xd bd dd dd ww ss ss ss
   aa rr qq ee hh gg ff jj ii ++ jj mm pp aa pp kk xx bb ++ ++ ww ss ++
```

Figure 14: A valid solution to the difficult switchbox after 4200 cycles.

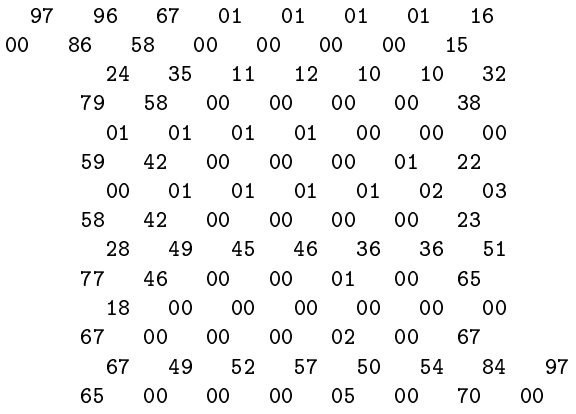

Figure 12: Weight values for net **a** after 800 update cycles.

valid solution for this problem instance. The trial solution after 6000 cycles is shown in Figure 15. Notice that the problem is with the **b** and **d** nets in the northeast corner. This is a difficult instance of the switchbox routing problem. By analyzing such cases one can look for ways to modify the basic dynamics so that these final states no longer constitute fixed points — though the accuracy of the simulation of the current flow, and other such factors, must also be considered.

6 Conclusions

We have presented a method for solving the switchbox routing problem on a grid of adaptive resistors. While the basic switchbox problem maps naturally into a grid, our basic method is applicable to a wide variety of graph topologies. The method can easily be extended to switchboxes with more than two layers or with arbitrary obstacles.

The method illustrates how local, adaptive processes can provide solutions to problems usually associated with search algorithms — or at least provides an alternative style of search algorithm. This may allow for the creation of fast, special-purpose hardware, or may help in improving traditional routing algorithms. It can also provide insight into natural phenomena, such as the ability of neural tissue to adaptively form routing connections across layers of locally acting neurons. The "winner-take-all" structure across local positive-feedback path formation slabs may bear some resemblance to the columnar formations found in the brain [Koh84].

Future research should analyze more rigorously the interaction of parameters and their effects in different routing situations. A global "energy function" formulation would also be desirable. The method might be extended to problems other than switchbox routing, for example to other types of routing problems or to Steiner tree approximation problems.

The feedback signals we have used are fairly low-level, though higher-level feedback can also be incorporated. For example, the current into a slab might be dynamically increased over time if that slab's net were found, by a higher-level routine, to not be getting routed properly. Certain parameters might be adjusted according to the number of incompletely routed nets. One might also consider the effects of injecting random noise into the system to eliminate unstable fixed points, or the dynamics of fluids with momentum.

References

[CH88] James P. Cohoon and Patrick L. Heck. Beaver: A computational-geometry-based tool for switchbox routing. *IEEE Transactions on Computer-Aided Design*, 7(6):684–697, June 1988.

[CK84] Chung-Kuan Cheng and Ernest S. Kuh. Module placement based on resistive network optimization. *IEEE Transactions on Computer Aided Design*, 3(3):218–225, July 1984.

[Koh84] Teuvo Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, 1984.

[SCF91] P-H Shih, K-E Chang, and W-S Feng. Neural computation network for global routing. *Computer-Aided Design*, 23(8):539–547, October 1991.

```
   pp ++ cc ee mm hh gg jj ff ii nn pp oo pp ++ vv uu bb cc tt bb ss
++ pp cp cp ep mp hp gp jp fp ip np pp op pp pp vp up bp cp tp bp sp pp
++ pp cc cc ec mc hc gc jc fc ic nc pc oc pc pc vc uc bb cb tb bd sd dd
oo co co co eo mo ho go jo fo io no po oo pc pc vc uc bc cc tt dt st tt
nn cn cn cn en mn hn gn jn fn in nn pp pp aa pa va ua ba ca da da sa aa
ll ll cl cl el ml hl gl jl fl il ll ll pa aa pa va ua bb cb db db sd ++
aa aa ca ca ea ma ha ga ja fa ia la la pa aa pa va uu bu cu du du su uu
bb ab cb cb eb mb hb gb jb fb ib lb lb pb ab pb vb ub bb cu du du ss ss
jj aj cj cj ej mj hj gj jj fj ij jj ll pl al pl vl uu bu cu du du su uu
cc ac cc cc ec mc hc gc jc fc ic jj ll pl al pl vl vl bl cl dl dl sl ll
rr ar rr mm em mm hh gg jc fc ic jc dc pc ac pc vv vv bv cv dv dv sv vv
mm am rm mm em hm hm gm jm fm im jm mm pc ac pc cc dc bc cc dd dd ss ss
qq aq rq qq ed hh gh gg jf ff ii jd md pd ad pd cd xx bx cx dx cx sx xx
ee ae re qe ee hh gg ff jf if ii jd md pp ap pp cc xc bc cc dc cc sc cc
kk ak rk qk ek hk gk fk jk ik ik jk mk pk ak pk kk xx bx dx dd ww sw ww
dd ad rd qd ed hd gd fd jd id id jd md pd ad pd kd xd bd dd dd ww ss ss
   aa rr qq ee hh gg ff jj ii ++ jj mm pp aa pp kk xx bb ++ ++ ww ss
```

Figure 15: An invalid solution to the more difficult switchbox after 6000 cycles.

[SF91] P-H Shih and W-S Feng. An application of neural networks on channel routing problem. *Parallel Computing*, 17:229–240, 1991.

[VM90] R. Venkateswaran and Pinaki Mazumder. A hexagonal array machine for multilayer wire routing. *IEEE Transactions on Computer Aided Design*, 9(10):1096–1112, October 1990.