The Eight Rotors Puzzle

written by Jim Propp
for the Eighth Gathering for Gardner
(April 10, 2008 version)

Here's a puzzle I know how to solve, but not how to build! If I tell you all how to solve it, maybe one of you will have some clever ideas for how to build it.

It's one of those "How do you get this physical object from state A to state Z (by means of moves of a specific sort)?" puzzles, but I don't know a good way to build the object so as to enforce the rules mechanically; I'm hoping someone can think of a smart way to do this. (It would also be possible to embody the rules in computer code; maybe one of you will create a Java applet or a Mathematica Demonstrations program. You can find existing Java applets that implement variants of this puzzle at `http://www.cs.uml.edu/~jpropp/rotor-router-model/`.)

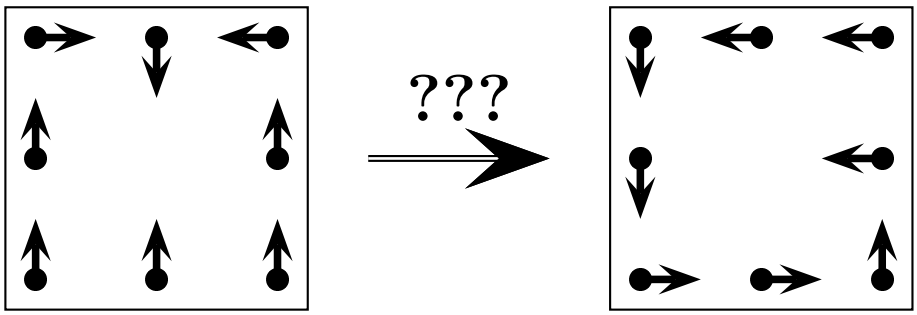

Figure 1: How do we get from **M** to **G**?

The puzzle is based on work that I and others have done on something we call the "rotor-router model" (see [**1**], available on the web at `http://arxiv.org/abs/0801.3306`). Rotor-routing is a very general scheme for removing the randomness from random systems while retaining some of their random-like properties. I'll say a bit more about that in the final version of this article (the one that I hope will be published in the next Gathering for Gardner book). That article, as befits an article written in honor of Martin, will also contain comments from readers about the puzzle and its variants. But for now I'll just treat the basic puzzle itself, and only mention aspects

of the general theory to the extent that they can help us solve our particular puzzle.

The prototype for the puzzle that is being distributed at the Eighth Gathering for Gardner (made by Walt and Christine Hoppe, owners of Laser Perfect, Inc., and attached to this document) involves a simple arrangement of eight arrows on a square board, with each arrow free to rotate about its tail. Let's call such an arrow a "rotor". Four of the rotors are at the corners of the board, and the other four are at the middles of the four sides of the board. Each of the rotors thus has two neighboring rotors. Each corner rotor is allowed to point at either of its two neighbors, so we'll call the corner rotors "2-way rotors". The other four rotors (the ones at the middles of the sides) can point to the empty space in the middle of the board in addition to pointing to either of the two neighboring rotors, so we'll call the non-corner rotors "3-way rotors". (I'll sometimes use the words north, south, east, and west to describe the layout; thus, the upper left corner may also be called the northwest corner, etc.)

In this puzzle, the goal is to get from one configuration of the rotors to another. Specifically, the goal is to get from the **M** (for Martin) of the left part of Figure 1 to the **G** (for Gardner) of the right half of Figure 1, though you could vary the puzzle by having a different starting configuration and/or a different ending configuration.

Of course you aren't allowed to just turn the rotors to point where you want them to point (you didn't really expect a puzzle created in honor of Martin Gardner to be *that* easy, did you?). Instead, you only get to turn the rotors by moving chips around on the board in accordance with a certain rule. The chips can be any sort of tokens you like (poker chips, coins, pebbles, disk magnets, whatever) as long as they're small enough to sit on a single rotor, and the rule is that whenever you move a chip that is sitting on a particular rotor, you must follow the "rotor-router rule": you rotate that rotor clockwise to its next legal position and then route the chip in the direction that the rotor is pointing (either to a neighboring rotor or to the hole in the middle of the board).

Note that you can't rotate a rotor unless there's a chip sitting on it; that you must rotate such a rotor clockwise to its next legal position; and that you must then move only one chip sitting at that rotor. (This last condition is relevant when there are several chips on a rotor.)

Figures 2 and 3 illustrate two successive legal moves. On the left side of Figure 2(a), we see a configuration with two chips, one at the northwest 2-

way rotor and one at the western 3-way rotor. We decide to move the chip at the upper left first. So we rotate the upper left rotor and then slide the chip in the direction in which the rotor now points, obtaining the configuration on the right side Figure 2 (which is also shown on the left side of Figure 3). Now there are two chips occupying the western 3-way rotor. We decide to move one of them (it doesn't matter which, since in this puzzle chips might as well be identical). So we rotate the rotor, and then slide the chip in the direction in which the rotor now points, obtaining the configuration shown on the right side of Figure 3. Note that a chip that moves to the middle of the board is immediately removed, as it is no longer part of the action.

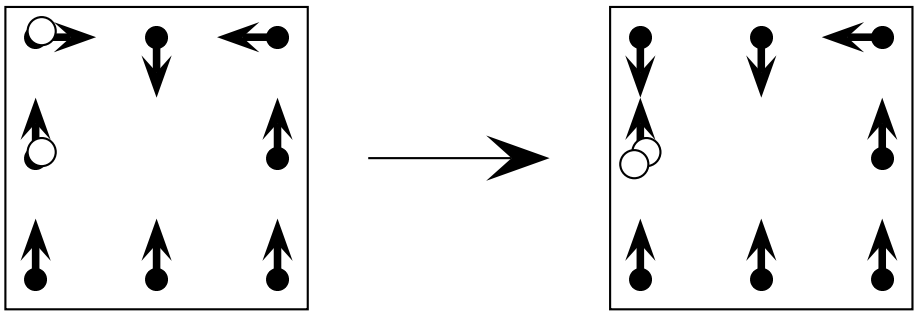

Figure 2: A legal rotor-router move.

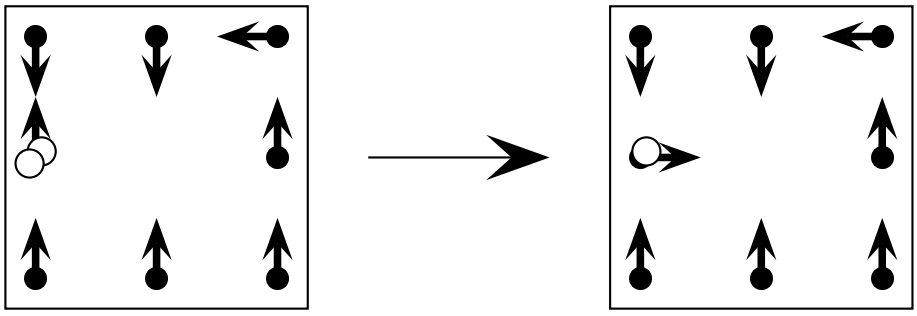

Figure 3: Another legal rotor-router move.

(To get comfortable with using rotors, you may wish to check that if you continue to move the remaining chip in accordance with the rotor-router rule, the chip takes eleven steps, moving south, east, east, west, west, north, north, east, west, south, and then finally east. Remember, when you move a chip, you must turn the associated rotor clockwise to its next *legal* position; this may be a 90 degree, 180 degree, or 270 degree turn.)

At any time you can add a chip to any rotor you like and move any chip you like (as long as you follow the rotor-router rule), but you aren't allowed to remove a chip whenever you like; chips can only be removed as a side-effect of rotor-routing. You must start with the rotors forming the **M** configuration, with no chips on the board; and you must end with the rotors forming the **G** configuration, with no chips on the board.

The first challenge is, can you do it? The second challenge is, can you do it with as efficiently as possible? There are different ways to measure efficiency; for this puzzle, a good way to measure efficiency is to count the number of times you added a chip.

Already there is a natural non-trivial question that presents itself: how do you know you can't get into an infinite loop in which chips keeps moving around on the board, never disappearing into the middle? It's not hard to show that this can't happen. For, when a chip leaves a 2-way rotor, it moves to a 3-way rotor; and a 3-way rotor can't be visited more than 3 times without routing a chip into the middle (and out of the game). So chips must keep disappearing from the board at a steady rate.

If you're the sort of person who likes to figure things out for yourself, you should put the article down now and play with the puzzle for a while, because a major spoiler is coming up in the very next sentence.

An important and surprising property of the game is that all that really matters is how many chips you add at each rotor — NOT when you add them, nor in what order you move the chips (when you have a choice about which chip to move). For instance, suppose you find a way to get from one rotor configuration C to another rotor configuration C' with a sequence of moves that involves adding a single chip at the upper left, rotor-routing it through the board until it exits, then adding a chip at the lower right, and rotor-routing *it* through the board until it exits. Then I claim that if instead you start with C and add the chip at the lower right *first*, router-route it through the board until it exits, then add the chip at the upper left, and router-route it through the board until it exits, you must obtain the same end-state C'. Or you could add both chips to the board at the very start, and use a coin-toss to decide which of the two chips to rotor-route at each step until one of them leaves the board, and then move the remaining chip until it too has exited. No matter what you do, you'll end up with C'.

This is sometimes called the abelian property, but I prefer to call it the convergence property.

You should test the convergence property now on a few examples, both

to convince yourself that I'm telling you the truth and to give yourself some practice with the kinesthetics of rotor-routing. Don't forget that you can only move one chip at a time! If you violate this part of the rotor-router rule, you are working on a different puzzle than the one I have in mind, and the convergence property is not guaranteed to hold.

So, this puzzle is really asking you to come up with eight numbers, one for each rotor, such that if you add the appropriate number of chips at each rotor and allow the chips to rotor-route through the board, you'll get to the desired end-state. Once you've chosen how many chips you'll add at each rotor, no other choice you make matters.

Let's write these eight numbers in a 3-by-3 array with a hole in the middle, and call it a "chip configuration". For instance, the chip configuration

$$\begin{array}{ccc} 1 & 0 & 0 \\ 1 & & 0 \\ 0 & 0 & 0 \end{array}$$

corresponds to a chip at the upper left and a chip at the middle left; this is the chip configuration that we added to the rotor configuration $\mathbf{M}$ in the left half of Figure 2.

I'm going to give you a recipe for finding the chip configuration that solves the problem of getting from rotor configuration C to rotor configuration C', for any two acyclic rotor configurations C and C'. But first I'll need to explain what "acyclic" means, and before I do that, I'll give examples of rotor configurations that aren't acyclic.

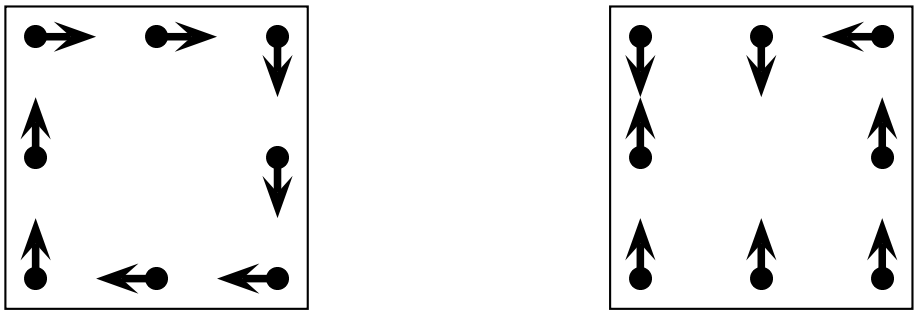

Figure 4: Rotor configurations with cycles.

The left part of Figure 4 shows a rotor configuration in which each rotor points clockwise around the board, forming a cycle of length 8. The right

part of Figure 3 shows a rotor configuration in which two rotors point at each other, forming a cycle of length 2. Neither of these configurations is acyclic.

On the other hand, the **M** and **G** configurations shown in Figure 1 are acyclic: each contains no cycles, or equivalently, if you start at any rotor and trace your eye from that rotor to the rotor it's pointing to, and then trace your eye to the rotor that *that* rotor is pointing to, and so on, your eye will eventually be led to the middle of the board, rather than going into an infinite loop. (Note that no rotors are turned in this mental tracing process.)

I now claim (without giving the proof) that the chip configuration

$$\begin{array}{ccc} 2 & 2 & 2 \\ 2 & & 2 \\ 2 & 2 & 2 \end{array}$$

has a magical property: starting from any acyclic configuration of the rotors, if you add two chips at every rotor and pass them through the system, you end up with the same rotor configuration you started with. This holds true no matter what your original acyclic configuration was.

You now have enough clues to enable you to figure out on your own how to get from **M** to **G**, so try to puzzle this out before reading further!

The northwest rotor in the **M** configuration is pointing east, and we want it to end up pointing south (the way it does in the **G** configuration). We can force that rotor to point south right away if we add a chip at the northwest corner and move that chip *just once*. Likewise for all the other rotors: by adding 0, 1, or 2 chips, as appropriate, and moving each chip just once, we can make each rotor advance by 0, 1, or 2 positions, causing the rotors to point in the desired directions. You can check that if we add the chip configuration

$$\begin{array}{ccc} 1 & 1 & 0 \\ 2 & & 2 \\ 1 & 1 & 0 \end{array} \qquad (1)$$

to the **M** configuration of the rotors, and move each chip just once, the rotors will be in the **G** configuration. Of course this will leave some chips on the board. Specifically, you get the **G** configuration of the rotors plus the chip configuration

$$\begin{array}{ccc} 1 & 0 & 0 \\ 1 & & 0 \\ 1 & 1 & 2 \end{array} \qquad (2)$$

If you were allowed to remove chips whenever you wanted, you could just sweep all six chips from the board and declare victory. Now of course you can't just remove chips from the board, but you can do something that's almost as good: you can add ten *more* chips to the board so that the end result (when all the chips have left the board) is as if you'd just swept the six chips from the board!

Specifically, if you add

$$\begin{matrix} 1 & 2 & 2 \\ 1 & & 2 \\ 1 & 1 & 0 \end{matrix} \tag{3}$$

chips at the respective sites, you'll get the **G** configuration of the rotors plus the chip configuration

$$\begin{matrix} 2 & 2 & 2 \\ 2 & & 2 \\ 2 & 2 & 2 \end{matrix} \tag{4}$$

which (after all chips have exited) will give you the desired state **G**, because of the magical property of the all-2's chip configuration that I mentioned before. (There's no mystery to how I chose (3) to ensure that (2) plus (3) would be (4): I just let (3) be (4) minus (4).)

So, we've found one solution to the puzzle: Starting from **M**, if you add

$$\begin{matrix} 1 & 1 & 0 \\ 2 & & 2 \\ 1 & 1 & 0 \end{matrix}$$

chips (that's (1)), move each chip once, then add

$$\begin{matrix} 1 & 2 & 2 \\ 1 & & 2 \\ 1 & 1 & 0 \end{matrix}$$

chips (that's (3)), and then move the chips as you like until all of them have exited, you'll end up with **G**.

By the convergence property, you can put all the chips on the board at the start and get the same outcome. That is, you can add

$$\begin{matrix} 2 & 3 & 2 \\ 3 & & 4 \\ 2 & 2 & 0 \end{matrix} \tag{5}$$

(that's (1) plus (3)) to **M** and rotor-route the eighteen chips off the board, and you'll get **G**. Or, at the other extreme, you can process the chips one at a time, so that there's never more than a single chip on the board. You can use just a single chip: when it leaves the board, add it back at a suitable place. If you add the chip eighteen times, taking care that it gets added the appropriate number of times at each rotor, you'll end up with **G**.

This answers the challenge "Can you get from **M** to **G**?", so now let's move on to the challenge of doing it more efficiently.

It turns out that that some chip configurations are equivalent to others, in the sense that they have the same impact on any rotor configurations that you add them to. For instance, I claim that adding

$$\begin{array}{ccc} \underline{2} & \underline{3} & 2 \\ \underline{3} & & 4 \\ 2 & 2 & 0 \end{array}$$

has the exact same effect on a rotor configuration as adding

$$\begin{array}{ccc} \underline{0} & \underline{4} & 2 \\ \underline{4} & & 4 \\ 2 & 2 & 0 \end{array}$$

(I've underlined the numbers that differ in the two chip configurations.) To see this, notice that if you move the two chips at the upper left by rotor-routing them each a single step, one of them moves east and the other moves south, and the rotor ends up pointing whichever way it started out pointing.

Similarly, I claim that adding

$$\begin{array}{ccc} \underline{0} & \underline{4} & \underline{2} \\ 4 & & 4 \\ 2 & 2 & 0 \end{array}$$

has the exact same effect on a rotor configuration as adding

$$\begin{array}{ccc} \underline{1} & \underline{1} & \underline{3} \\ 4 & & 4 \\ 2 & 2 & 0 \end{array}$$

To see this, notice that if you move the three of the chips at the northern 3-way rotor by rotor-routing them each a single step, one of them moves east,

one of them moves west, and one of them moves south (and out of the game), and the rotor ends up pointing whichever way it started out pointing.

We can generalize these observations with the notion of *chip-firing*. When there are 2 or more chips at a corner location, we can redistribute two chips among the two neighboring locations. When there are 3 or more chips at a non-corner location, we can remove one from the board and redistribute two others among the two neighboring locations. Either way, we get a new chip configuration that has the same effect on any rotor configuration we add it to. And, in the second case (firing chips from a non-corner location), the number of chips in the configuration goes down by 1, which means progress: for instance, that last configuration

$$\begin{array}{ccc} 1 & 1 & 3 \\ 4 & & 4 \\ 2 & 2 & 0 \end{array}$$

is a seventeen-chip solution to the **M**-to-**G** puzzle.

If you try to improve

$$\begin{array}{ccc} 1 & 1 & 3 \\ 4 & & 4 \\ 2 & 2 & 0 \end{array}$$

further by doing more rounds of chip-firing (which can clearly be done in lots of ways), you will eventually wind up with the chip configuration that cannot be reduced further by chip-firing. In fact, chip-firing is governed by a convergence theorem of its own, which guarantees that if you fire until no more firing is possible, you'll arrive at an end state that doesn't depend on the choices you made along the way. In this case, that chip configuration is

$$\begin{array}{ccc} 1 & 0 & 1 \\ 2 & & 2 \\ 1 & 2 & 1 \end{array}$$

So, chip-firing gives a ten-chip solution to the **M**-to-**G** puzzle.

But can we do better?

We can, using a generalization of the chip-firing operation called cluster-firing. This is like chip-firing several locations at once, except that a location is allowed to temporarily borrow chips from the universe as long as it pays them back at the end of the operation, or equivalently, as long as we allow a location to briefly have a negative number of chips, provided that it has

a non-negative number of chips when the operation is over. Let's apply cluster-firing to

$$\begin{array}{ccc} 1 & 0 & 1 \\ 2 & & 2 \\ 1 & 2 & 1 \end{array} \tag{6}$$

by firing all the locations in the bottom row. First, we fire the southwest location:

$$\begin{array}{ccc} 1 & 0 & 1 \\ 3 & & 2 \\ -1 & 3 & 1 \end{array}$$

Next we fire the central southern location:

$$\begin{array}{ccc} 1 & 0 & 1 \\ 3 & & 2 \\ 0 & 0 & 2 \end{array}$$

Last we fire the southeast location:

$$\begin{array}{ccc} 1 & 0 & 1 \\ 3 & & 3 \\ 0 & 1 & 0 \end{array} \tag{7}$$

Note that there are no negative numbers here, so our operation is a legal example of cluster-firing. That is, we can cluster-fire the bottom row of (6), and the result is (7).

Now, it's not true that adding (7) to *any* rotor configuration has the same effect as adding (6), but it *is* true if we restrict ourselves to acyclic rotor configurations. Since the **M** and **G** configurations are acyclic, that means that if you add

$$\begin{array}{ccc} 1 & 0 & 1 \\ 3 & & 3 \\ 0 & 1 & 0 \end{array}$$

chips, you can turn **M** into **G**. So now we've got nine-chip solution.

By doing more cluster-firing, we eventually arrive at a chip configuration in which no more cluster-firing is possible. As you might be expecting by now, there is a convergence theorem for cluster-firing as well, and it guarantees in this case that no matter how you cluster-fire, you'll end up with

$$\begin{array}{ccc} 0 & 2 & 0 \\ 2 & & 2 \\ 0 & 0 & 0 \end{array}$$

This is a six-chip solution to the **M**-to-**G** puzzle, and it is optimal.

If you've followed everything I've told you, then you should be able to find a five-chip solution to the inverse puzzle of turning **G** back into **M**.

You'll notice that turning an **M** into a **G** took more chips than turning a **G** into an **M**. So some variants of the **M**-to-**G** puzzle are easier, and it's also true that some are harder. One question that I don't know the answer to, and which perhaps one or more of you will solve, is: If you let your worst enemy choose the starting configuration and the target configuration, and his goal is to make the puzzle as hard for you as possible (where hardness is measured by the number of chips you'll need to add to the board to get from the starting configuration to the target configuration, assuming chips aren't re-used), how many chips can he force you to use? Clearly the answer to this question must be at least six, since that's how many chips the **M**-into-**G** puzzle forces you to use.

One could also ask this question using the number of chip-moves, rather than the number of chips, as the measure of difficulty of a puzzle.

How many such puzzles does your enemy have to choose among? It turns out that there are exactly 192 acyclic rotor configurations. So there are $192 \times 191 = 36,672$ non-trivial Eight Rotors puzzles (plus 192 trivial puzzles in which the starting configuration and target configuration are one and the same). The number 192 is also the number of "recurrent" chip configurations; the article [**1**] explains what this phrase means, and explains why it is no coincidence that the same number occurs in both contexts. Indeed, the Eight Rotors Puzzle belongs to a special class of puzzles (including Lights Out and Rubik's Clock) in which an abelian group is acting behind the scenes; the abelian group for the eight rotors puzzle has exactly 192 elements.

See `http://jamespropp.org/quasirandom.html` for more on rotor-routing (including some discussion of its links to probability theory).

Comments on this article are welcome! I have a gmail account under the name `JamesPropp`.

References

[**1**] *Chip-Firing and Rotor-Routing on Directed Graphs*, A. Holroyd, L. Levine, K. Mészáros, Y. Peres, J. Propp, and D. Wilson; to appear in "In and out of Equilibrium II", eds. V. Sidoravicius and M.E. Vares, Birkhauser (2008).