1

THE CYCLIC PRISONERS

Peter Winkler

In the world of mathematical puzzles, prisoners are faced with a fascinating variety of tasks, some of which can be re-cast as serious problems in mathematics or computer science. Here we consider two recent prisoner puzzles (really problems in an area of computer science known as "distributed computing"), in which communication is limited to passing bits in an ever-changing cyclic permutation.

1 Bit-Passing in Prison

The following marvelous puzzle was passed to me by Boris Bukh, of Carnegie-Mellon University, but was given to him by Imre Leader of Cambridge University, and to him by the composer, Nathan Bowler of Universität Hamburg.

> You are the leader of an unknown number of prisoners. The warden explains to you that every night, each prisoner (including you) will write down a bit (that is, a 0 or 1). The warden will then collect the bits, look at them, and redistribute them to the prisoners according to some cyclic permutation, which could be different every night. The prisoners are lodged in individual cells and have no way to communicate other than by passing these bits.
>
> You will all be freed if after some point, every prisoner knows, with certainty, how many prisoners there are. Before the bit-passing begins, you have the opportunity to broadcast to your compatriots (and the warden) your instructions. Can you design a protocol that will succeed no matter what the warden does?

It is my contention that despite its fanciful assumptions, the puzzle is a legitimate, serious problem in distributed computing—because it gets at the question: *what is the minimum amount of communication required to make a nontrivial discovery?* In distributed computing, a network of processors face

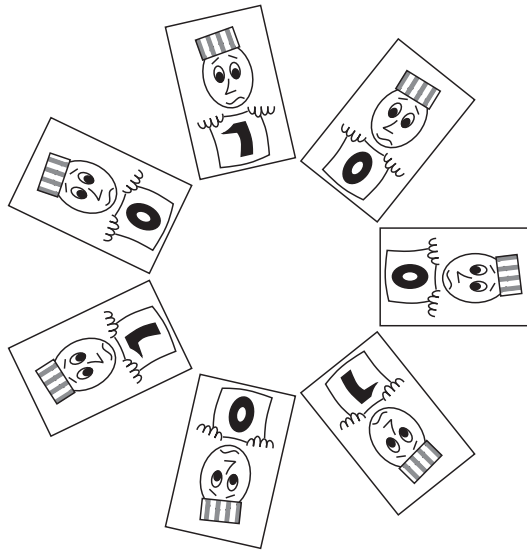

Figure 1.1. Some prisoners and their bits

some cooperative task—here, counting its members—and must accomplish this despite communication constraints.

In distributed computing the network is usually fixed, so that (for example) the message passed in a particular direction by a particular processor reaches the same target processor every time. It must be assumed that the network is *connected*, otherwise some part of the network would never be able to reach some other part.

Here, connectivity is ensured by the constraint that the messages be permuted in one large cycle. But the messages are only single bits, and the permutation is not only variable—it is also controlled by an adversary who sees the messages. That anything at all can be accomplished in such a setting is remarkable.

The solution presented below is my own. A somewhat similar solution was posted by Zilin Jiang [2], then a graduate student at Carnegie Mellon University. The composer Nathan Bowler has sent me his own write-up, which includes another, similar, solution plus one with a very different second phase suggested by Attila Joó (described briefly in Section 4).

The basic mechanism for all the solutions is what I call a "poll." Let P be some property that any prisoner knows whether or not he possesses. A P-poll enables all the prisoners to find out whether they *all* have property P. This works as follows. On the first night of the poll, each prisoner without property P writes "0" as his bit, while those with P write "1." On each subsequent night, each prisoner who has ever sent a 0 in this poll continues to do so; each

prisoner who has been sending 1s continues to do so until he receives a 0, after which he starts sending 0s. In other words, the 0s proliferate during a poll—if there were any to begin with!

The poll lasts for k nights, where k is a known bound on the number $|P|$ of prisoners with property P. Then, if there is at least one prisoner without property P, the number of prisoners sending 0s will increase by at least one every night until, after k nights, all prisoners with property P will be getting 0s. Of course, if all prisoners have property P, then no 0s will ever be passed, and all prisoners will still be getting 1s after k nights. Thus, all prisoners will know at the end of the poll whether they all have property P.

It is critical that each prisoner knows when a poll is being conducted, what property P is being queried in the poll, and the number k of nights the poll will take. Then, on the last night of the poll, everyone will have received the same bit. If prisoner "George" receives a 1 on the last night of the poll, and he himself has property P, he knows everyone had property P, and he knows that everyone else knows, too. If George receives a 0 on the last night, or if he did not have property P, he knows that not everyone had property P—and, again, he knows that everyone else knows, too.

The first phase of the protocol is devoted to getting a bound b on n, the total number of prisoners. (After that, all subsequent polls can be run for b nights.) You, the leader, begin the first "probe" by sending out a 1, while every other prisoner is sending (by instruction) a 0. After a probe, you, and any prisoner (just one, in this case) that has received a 1, are deemed to have been "reached." Since two prisoners have been reached, a two-night poll will suffice to determine whether all prisoners have been reached. If they have, there are just two prisoners and they both now know it.

Otherwise a second probe is initiated, in which all reached prisoners send out 1s while all others send out 0s. This is again followed by a poll, but this time we take $k = 4$, since as many as four (and as few as three) prisoners may have been reached.

Each probe is a one-night affair in which every prisoner that has been reached (that is, has received a 1 during some previous probe) sends out a 1, while every other prisoner sends out a 0. Since the permutation of bits is cyclic, the 1s sent out during a probe cannot all remain in the community of "reached" prisoners unless all prisoners have been reached, in which case the poll following the probe will reveal this fact and the prisoners will all know to go on to the next phase.

Until that happens, probes and polls continue, with a poll of duration $k = 2^m$ following the mth probe. Eventually, at the end of (say) the mth poll, all prisoners discover simultaneously that every prisoner has been reached. Moreover, each prisoner now knows that there are at most $b = 2^m$ prisoners in all, since the number of reached prisoners cannot more than double during a probe.

Furthermore, each prisoner knowingly belongs to a "group" G_i, of size g_i, with $0 \leq i \leq m$, where i is the number of the first probe that reached him. (Your group, as the leader, is the singleton G_0). The group sizes are mostly unknown at this point, except that each $g_i \geq 1$, since some new prisoner was reached with each probe.

Notice that once the prisoners get your initial broadcast with all the instructions for Phase 1 (and also Phase 2, described below), they know what's going on at all times. They know, for instance, that there will be a probe on the first night, followed by a poll of length 2, followed by another probe on night 4, then a poll of length 4, a probe on night 9, a poll of length 8, and so forth, until one of the polls finds that everyone has been reached.

As an example, suppose there are just three prisoners, you, Bob, and Carl. In the first probe, you send out a 1, while Bob and Carl send out 0s. Say your 1 goes to Carl. There is now a two-night poll. On the poll's first night, poor unreached Bob sends out a 0, while you and Carl send out 1s. On the second night, Bob and whoever got Bob's previous 0 both send out 0s, and as a result, all three of you now know that there was an unreached prisoner.

Another probe is thus run, where you and Carl send out 1s—one of which must get to Bob. A three-night poll now follows, in which everyone sends and receives only 1s; after the third night, all three of you know that no unreached prisoners remain. Moreover, Carl knows that he is in group G_1 and Bob in group G_2. At this point, however, no one knows whether there are one or two prisoners in group G_2; so n could be either 3 or 4. Figuring out which is the mission of the second phase of the protocol.

In the second phase, the prisoners seek to refine the groups and, when they are as refined as they are going to get, determine the groups' sizes.

Let $M = \{0, 1, \ldots, m\}$. For any subset $X \subset M$, denote by G_X the union of the prisoners in the groups G_i for $i \in X$, and put $g_X := |G_X|$. The 2^m subsets $X \subset M$ that do not contain 0 are now considered one at a time, in some order—say, lexicographic—that you, as leader, have announced in advance for any possible m.

For each such X, an "X-probe" is launched in which every prisoner in G_X sends out a 1 while the rest send out 0s. This probe is followed by a series of $2m$ polls in which it is determined whether any two people in the same group received different bits on the night of the probe. (For example, suppose $X = \{2, 3\}$. The third poll after the X-probe might ask whether anyone in G_2 itself got a 0, and then the fourth poll would ask whether anyone in G_2 got a 1. The fifth and sixth polls would query G_3, the seventh and eighth G_4, and so forth.)

If the polls uncover any groups whose members received both 0s and 1s, those groups are split according to the bit received. The groups are then renumbered, and the second phase is restarted with a new, larger value of m.

If no two members of any group received different bits on the probe night, we note which groups got 1s and which 0s; since the permutation is cyclic, it cannot be that all the 1s fell back into G_i. Let U be the set of is in X for which the members of G_i received 0s, and V the is in $M \setminus X$ for which the members of G_i received 1s. Then U and V are disjoint and nonempty, and $g_U = g_V$, since both sides of the equation count the 1s that exited G_X.

The next X in line then launches a new probe, and the same procedure is followed. If polls uncover any groups whose members received different bits, those groups are subdivided, and the whole second phase is restarted with higher m.

Since the groups cannot be subdivided forever, eventually every eligible X will have launched a probe in which no two members of the same group got different bits. At this point, we have, for every subset $X \subset M$, not containing 0, a pair of nonempty sets U and V with $U \subset X$, $V \subset M - X$, and $g_U = g_V$.

The claim is that we are now done: the g_is are now all determined, so every prisoner can compute them and add them up to get n. To prove the claim, assume that there are two different solutions, $g_0, \ldots, g_m$ and $h_0, \ldots, h_m$, with (say) $g_j < h_j$ for some fixed j. Let $X = \{i : g_i < h_i\}$; then X is nonempty, and $0 \notin X$, since $g_0 = h_0 = 1$. Thus there are nonempty sets $U \subset X$ and $V \subset M \setminus X$ with $g_U = g_V$ and $h_U = h_V$, giving $h_U - g_U = h_V - g_V$. But this is impossible, because $h_U - g_U$ is positive while $g_V - g_V$ is at most zero, so the claim is proved.

2 Firing Squad

We now consider a variation of the above problem in which the prisoners have a bound on their number but do not know when the protocol begins. In anticipation of capture, you must provide a protocol to your compatriots, before anyone is captured, to handle the following situation.

As prisoners are being rounded up and sent to the notorious Cyclic Prison, those already there are already sending out bits every night. When all are present (including you), you only, the leader, are notified. Your objective now is to get synchronized; once you do, you could use the second phase of the previous protocol to determine the precise number of prisoners. But your actual plan is to overwhelm the guards, and to do this, all prisoners must agree on the date of insurgency.

(In distributed computing this problem is sometimes known as the "firing squad" problem, but it is understandable that you and your troops might not want to use this term.)

In this problem everyone knows that there are 100 troops in your regiment and therefore that 100 is an upper bound on the number of prisoners. Can you design an algorithm that will bring every prisoner to action on the same day?

If you solved the first puzzle, you might find this one relatively easy. The idea is that until he receives his first 1, every prisoner will send out only 0s. You, as leader, will do the same until notified that all prisoners are present, at which time you will initiate the first probe by sending out a 1. The first and every later probe is followed by a poll of length exactly 100, then another probe and another poll, and so forth, until some poll reports that all prisoners have been reached. The revolt begins the next morning.

When a prisoner is reached (by a 1) for the first time, he knows it is a probe but does not know *which* probe. Fortunately, since all polls are the same length, he does not need to know; he just cooperates with the poll, sending out 1s until he gets a 0, for 100 nights. If he gets a 1 on the hundredth night, he knows all prisoners have been reached and that next morning they will all make their move. Otherwise, he sends out a 1 as part of the next night's probe, and then endures another 100 nights of polling.

3 Discussion

It seems that the prisoners require not only a bound, but a *common* bound on their number to solve the firing squad problem. Without that, the prisoners will not know when polls end. But can you prove that the common bound is necessary?

The firing squad protocol is (relatively) efficient, at least, taking at most $b^2 - 1$ nights to complete once it gets started, where b is the common bound. But as stated above, both phases of the proposed counting problem have length exponential in the number of prisoners.

We do not really care how each prisoner actually computes the g_is; any method for solving a set of linear equations will do. There are 2^m equations, which seems like a gross overabundance for m unknowns; but if there are no "on-the-fly" adjustments, no X can be skipped in the second phase of the protocol without risk that the g_is will no longer be determined. The reason is that during any probe other than the one for the skipped X, it is conceivable that all 1s sent from G_X return to G_X and all 1s sent from $M - X$ return to $M - X$. If this happens every time, the g_is for $i \in X$ will be determined only up to a multiplicative factor.

However, by choosing the sets X adaptively, there is a way for the prisoners, given a bound N, to count themselves in time polynomial in N. When it is time to pick a set X of groups, the prisoners take the first one (in their agreed-on ordering of the sets) for which they cannot already *deduce* an answer; that is, the first X for which the equations they know so far determine no pair U, V with $u \subset X$, $V \subset Y$, and $g_U = g_V$.

Then the equation that arises from each new X-probe is guaranteed to be independent of the previous ones, and therefore m probes are enough to

determine the g_is, where $m < b$ is the number of groups of unknown size after subdivision has run its course.

The polling after an X-probe takes at most time $2bm$ (each group has to be asked whether anyone in it got a 0 during the probe, then whether anyone got a 1). The groups cannot be subdivided more than b times, so altogether the procedure takes at most $2b^4$ nights.

We still have the problem that b itself may be exponential in the actual number of prisoners, since the first phase of the protocol must allow for the number of reached prisoners doubling after each probe, when it might only increase by one. No one knows any way around this problem, but maybe the warden will concede when he sees that your protocol is bound to work. Otherwise, you and your prisoners might all be dead of old age before the protocol terminates.

If you can find a protocol for the counting problem that terminates in time polynomial in the number of prisoners—or a proof that none exists—let me know!

4 A Different Solution

As mentioned, another solution to the second phase of the counting problem has been suggested by Attila Joó. In it, each prisoner except you (the leader) is given b^2 "virtual coins"; his stack of coins goes up by one every time he receives a 1, and down by one whenever he sends a 1. You start with no coins with the idea of collecting enough to know how many prisoners there are. The rules are as follows.

Let the maximum number of coins held by any prisoner other than you be denoted by m. A prisoner's stack of k coins is deemed to be "large" (at a given point in the protocol) if every stack size between k and m is held by at least one prisoner. Prisoners (but not you) with large stacks send out 1s, while the others send out 0s. Between these events, polls enable everyone to ascertain which stack sizes exist, so the prisoners can determine whether their own stacks are large.

If c_i is the numbers of prisoners with a stack of size i, then the sequence $(c_1, \ldots, c_b)$ will descend lexicographically after every night until no prisoner other than you has a stack of size exceeding b. At that point, you can determine from the size of your own stack the number of prisoners (and then, if needed, pass that information to the others by using polls.)

Protocols such as this one using virtual coins have been used before in distributed computing—and even in prisoner problems—see, for examples, "The Two-Bulb Room" on p. 122 of Winkler [1]. Joó's version is quite clever, I think, but it does appear to require exponentially many nights in the worst case.

Of course, better solutions than anything discussed here may well exist!

Acknowledgment

My research is supported by NSF grant DMS-0901475.

References

[1] P. Winkler. *Mathematical Puzzles: A Connoisseur's Collection*. A. K. Peters, New York, 2004.

[2] Z. Zhang. The ultimate prisoner puzzle, in *Roy and His Friends*. `http://www.libragold.com/blog/2013/05/the-ultimate-prisoner-puzzle`, last accessed June 26, 2016.