EuroComb 2005 DMTCS proc. **AE**, 2005, 73–76

Deterministic Random Walks on the Integers

Joshua Cooper[1], Benjamin Doerr[2], Joel Spencer[1] and Gábor Tardos[3]

[1]*Courant Institute of Mathematical Sciences, New York*
[2]*Max-Planck-Institut für Informatik, Saarbrücken*
[3]*Rényi Institute of the Hungarian Academy of Sciences, Budapest*

We analyze the one-dimensional version of Jim Propp's P-machine, a simple deterministic process that simulates a random walk on $\mathbb{Z}$. The "output" of the machine is astonishingly close to the expected behavior of a random walk, even on long intervals of space and time.

Keywords: random walks, chip firing games.

1 The Propp Machine

In Cooper and Spencer (2005), the authors consider the following "Propp machine", also known under the name "rotor router model": Chips are placed at even integers. Each integer is assigned a direction – left or right. Then, at each step of time, all integers d simultaneously "fire", i.e., they send their chips to locations $d-1$ and $d+1$. If an integer has an even number of chips, it sends them equally in each direction. If an integer has an odd number of chips, it splits the pile evenly, except for one, which it sends in the direction that d is currently assigned. Then, d's direction is flipped, i.e., left to right or right to left.

Alternatively, one can imagine that each integer has a two-state "rotor" sitting on it, which, when the clock ticks, flips back and forth, depositing one chip in the direction it points until there are no chips left. All rotors act simultaneously and in sync.

The primary reason that this process is interesting is that it closely resembles a random walk. Chips are sent evenly in each direction at each time step, and we ensure that "odd" chips are distributed as evenly as possible by alternating which direction to send them. If the chips did a true random walk instead, one could reasonably guess that the expected number of chips at a given location, after a given amount of time, would be quite close to the number of chips deposited there by Propp machine, were it run in parallel.

In fact, the expected number of chips in the random walk is determined by a similar process (which we call "linear machine"), except that chips are split in fractions as necessary to ensure that d sends the exact same number of chips to $d-1$ and $d+1$. Hence, there are no rotors, which ensure integrality in the Propp machine. Despite this difference, however, the discrepancy between the two processes does not accumulate: in Cooper and Spencer (2005) it was shown that there is a constant c_1 such that, given any initial configuration of chips and rotors, and any amount of time, no chip-pile differs between the two processes by more than c_1. In fact, the authors show a generalization of this to $\mathbb{Z}^d$, but here we are concerned only with the one-dimensional case, as it is already surprisingly rich.

2 Our Results

We obtain the following results: Fix any starting configuration, that is, the number of chips on each vertex and the position of the rotor on each vertex. Now run both the Propp machine and the linear machine for a fixed number of time steps. Looking at the resulting chip configurations, we have the following:

- On each vertex, the number of chips in both models deviates by at most a constant $c_1 \approx 2.29$. We may interpret this as that the Propp machine simulates a random walk extremely well. In some sense, it is even better than the random walk. Recall that in a random walk a vertex holding n chips only in expectation sends $n/2$ chips to the left and the right. With high probability, the actual numbers deviate from this by $\Omega(\sqrt{n})$.
- In each interval of length L, the number of chips that are in this interval in the Propp model deviates from that in the linear model by only $O(\log L)$ (instead of, e.g., $2.29L$).
- If we average this over all length L interval in some larger interval of $\mathbb{Z}$, things become even better. The average squared discrepancy in the length L intervals also is only $O(\log L)$.

We may as well average over time. In the setting just fixed, denote by $p(x,T)$ the sum of the numbers of chips on vertex x in the last T time steps in the Propp model, and by $\ell(x,T)$ the corresponding number for the linear model. Then we have the following discrepancy bounds:

- The discrepancy on a single vertex over a time interval of length T is at most $|p(x,T) - \ell(x,T)| = O(T^{1/2})$. Hence a vertex cannot have too few or too many chips for a long time (it may, however, alternate having too few and too many chips and thus have an average $\Omega(1)$ discrepancy over time).
- We may extend this to discrepancies in intervals in space and time: Let I be some interval in $\mathbb{Z}$ having length L. Then the discrepancy in I over a time interval of length T is at most

$$\Big|\sum_{x \in I} p(x,T) - \sum_{x \in I} \ell(x,T)\Big| = \begin{cases} O(LT^{1/2}) & \text{if } L \leq T^{1/2}, \\ O(T \log(LT^{-1/2})) & \text{otherwise.} \end{cases}$$

  Hence if L is small compared to $T^{1/2}$, we get L times the single vertex discrepancy in a time interval of length T (no significant cancellation in space); if L is of larger order than $T^{1/2}$, we get T times the $O(\log L)$ bound for intervals of length L (no cancellation in time, the discrepancy cannot leave the large interval in short time).

All boundS stated above are sharp, that is, for each bound there is a starting configuration such that after suitable run-time of the machines we find the claimed discrepancy on a suitable vertex, in a suitable interval, etc.

A technicality: There is one limitation, which we only briefly mentioned, but without which our results are not valid. Note that since the integers form a bipartite graphs, the chips that start on even vertices never mix with those which start on odd positions. It looks as if we would play two games in one. This is not true, however. The even chips and the odd ones may interfere with each other through the rotors. Even worse, we may use the odd chips to reset the arrows and thus mess up the even chips. Note that the odd chips are not visible if we look at an even position after an even run-time. An extension of the arrow-forcing theorem presented below shows that indeed we use the odd chips to arbitrarily reset the rotors.

This is equivalent to running the Propp machine in an adversarial setting, where an adversary may decide each time where the extra odd chips on a position is sent to. It is clear that in this setting, the results above cannot be expected. We therefore assume that *the starting configuration has chips only on even positions* ("even starting configuration").

3 Some Proof Ideas

Due to space limitations, it is impossible even to sketch the proofs of the results stated above. We therefore focus on one particular aspect.

It is clear from the definition of the two machines that all discrepancies are caused by having an odd number of chips on vertices. If there is an even number of chips on some vertex, both machineS work identically here: They send half the chips to the left and half of the chips to the right. In fact, this means that only odd numbers of chips are important. Consider two starting configurations such that the initial rotor positions are identical and such that at each time on each vertex the parity of the number of chips is identical (in both configurations). Then all discrepancies (on vertices, and thus in intervals, etc.) are identical.

Conversely, we see that to prove lower bounds, it is crucial that we have some control over the parity. The following theorem tells us that we do. Roughly speaking, we can prescribe the parity on all vertices at all times.

We use $f(x,t)$ to denote the number of chips at time t at position x and $\mathrm{ARR}(x,t)$ to denote the value of the arrow at time t and position x, i.e., $+1$ if it points to the right, and -1 if it points to the left.

We consider the machine to be started at time $t = 0$. Being a deterministic process, the initial configuration (i.e., the values $f(x,0)$ and $\mathrm{ARR}(x,0)$, $x \in \mathbb{Z}$) determines the configuration at any time $t > 0$ (i.e., the values $f(x,t)$ and $\mathrm{ARR}(x,t)$, $x \in \mathbb{Z}$). The totality of all configurations for $t > 0$ we term a *game*.

Theorem 1 (Arrow-forcing Theorem) *Let $\rho(x,t) \in \{-1,+1\}$ be arbitrarily defined for $x \in \mathbb{Z}, t \in \mathbb{N}_0$ such that $x - t$ is even. Then there exists an even initial configuration that results in a game with $\mathrm{ARR}(x,t) = \rho(x,t)$ for all such x and t.*

Proof: Assume the functions f and ARR describe the game following an even initial configuration, and for some $T \geq 0$, we have $\mathrm{ARR}(x,t) = \rho(x,t)$ for all $0 \leq t \leq T+1$ and $x - t$ even. We modify the initial position by defining $f'(x,0) = f(x,0) + \epsilon_x 2^T$ for even x, while we have $f'(x,0) = 0$ for odd x and $\mathrm{ARR}'(x,0) = \mathrm{ARR}(x,0)$ for all x. Here, $\epsilon_x \in \{0,1\}$ are to be determined.

Observe that a pile of 2^T chips will split evenly T times so that the arrows at time $t \leq T$ remain the same. Our goal is to choose the values ϵ_x so that $\mathrm{ARR}'(x,t) = \rho(x,t)$ for $0 \leq t \leq T+2$ and $x - t$ even. As stated above, this holds automatically for $t \leq T$ as $\mathrm{ARR}'(x,t) = \mathrm{ARR}(x,t) = \rho(x,t)$ in this case. For $t = T+1$ and $x - T - 1$ even we have $\mathrm{ARR}'(x,T+1) = \mathrm{ARR}'(x,T) = \mathrm{ARR}(x,T) = \mathrm{ARR}(x,T+1) = \rho(x,T+1)$ since we start with an even configuration. To make sure the equality also holds for $t = T+2$ we need to ensure that the parities of the piles $f'(x,T)$ are right. Observe that $\mathrm{ARR}'(x,T+2) = \mathrm{ARR}'(x,T)$ if $f'(x,T)$ is even, otherwise $\mathrm{ARR}'(x,T+2) = -\mathrm{ARR}'(x,T)$. So for $x - T$ even we must make $f'(x,T)$ even if and only if $\rho(x,T+2) = \rho(x,T)$. At time T the "extra" groups of 2^T chips have spread as in Pascal's Triangle and we have

$$f'(x,T) = f(x,T) + \sum_y \epsilon_y \binom{T}{\frac{T+x-y}{2}}$$

where $x - T$ is even and the sum is over the even values of y with $y - x \leq T$. As $f(x, T)$ are already given it suffices to set the parity of the sum arbitrarily. For $T = 0$ the sum is ϵ_x so this is possible. For $T > 0$ we express

$$\sum_y \epsilon_y \binom{T}{\frac{T+x-y}{2}} = \epsilon_{x+T} + h + \epsilon_{x-T}$$

where h depends only on ϵ_y with $x-T < y < x+T$. We now determine the ϵ_y sequentially. We initialize by setting $\epsilon_y = 0$ for $-T < y \leq T$. The values ϵ_y for $y > T$ are set in increasing order. The value of ϵ_y is set so that the sum at $x = y - T$ (and thus $f'(y - T, T)$) will have the correct parity. Similarly, the values ϵ_y for $y \leq -T$ are set in decreasing order. The value of ϵ_y is set so that the sum at $x = y + T$ (and thus the $f'(y + T, T)$) will have the correct parity.

Note that the above procedure changes an even initial configuration that matches the prescription in ρ for times $0 \leq t \leq T+1$ into another even initial configuration that matches the prescription in ρ for times $0 \leq t \leq T + 2$. We start by defining $f(x, 0) = 0$ for all x (no chips anywhere) and $\mathrm{ARR}(x, 0) = \rho(x, 0)$ for even x, while $\mathrm{ARR}(x, 0) = \rho(x, 1)$ for odd x. We now have $\mathrm{ARR}(x, t) = \rho(x, t)$ for $0 \leq t \leq 1$ and $x - t$ even. We can apply the above procedure repeatedly to get an even initial configuration that satisfies the prescription in ρ for an ever increasing (but always finite) time period $0 \leq t < T$. Notice, however, that in the procedure we do not change the initial configuration of arrows $\mathrm{ARR}(x, 0)$ at all, and we change the initial number of chips $f(x, 0)$ at position x only if $|x| \geq T$. Thus at any given position x the initial number of chips will be constant after the first $|x|$ iterations. This means that the process converges to an (even) initial configuration. It is simple to check that this limit configuration satisfies the statement of the theorem. □

Acknowledgements

The authors are grateful for various support they received. This in particular includes a highly fruitful visit at the Rényi Institute in February and March this year, as well as a three month's stay of the second author at the Courant Institute last year.

References

J. Cooper and J. Spencer. Simulating a random walk with constant error. *Combinatorics, Probability and Computing*, 2005. To appear.