DIMACS Technical Report 2005-22

The Nagger-Mover Game

by

Z. Nedev[1] S. Muthukrishnan[2,3]

[1]Univ. of Victoria, znedev@math.uvic.ca. Part of the work was done while visiting DIMACS.
[2]DIMACS Permanent Member
[3]Rutgers Univ., muthu@cs.rutgers.edu.

ABSTRACT

We introduce a new two-persons game. Nagger starts from position 0 in a round table with positions marked $0, 1, \ldots, n-1$ and repeatedly calls number of places to move, and Mover responds to each move with which direction—clockwise or anticlockwise—that the Nagger should move. What is the maximum number of positions that Nagger may occupy? We provide the exact bound for this question, provide algorithmic strategies for the Nagger and the Mover to meet this bound, and pose other variants of the Nagger-Mover game for future study.

1 Introduction

We introduce a new class of two-persons games that we call the *Nagger-Mover* games. We study its following basic version here.

There is a circular arrangement of chairs consecutively numbered $0, 1, \ldots, n-1$ around a circular table, known to both the Mover and the Nagger. The Nagger sits at chair $p_0 = 0$ at the beginning. The Nagger and Mover take turns with the Nagger leading the game. After i moves (initially $i = 0$), the Nagger is at some chair p_i. In the $i + 1$th move, Nagger calls the size of the next move d_{i+1}. The Mover decides which way—Clockwise (ie., $p_{i+1} = p_i + d_{i+1} \bmod n$) or Anti-Clockwise (ie., $p_{i+1} = p_i - d_{i+1} \bmod n$), denoted **+** and **–** respectively—the Nagger should move. At the end of this move, Nagger is in chair p_{i+1}, and the game continues. The Nagger may visit the same chair many times during the course of the game, that is, p_i's are not necessarily distinct. Notice that only one player, i.e., the Nagger, "sits" at various positions around the table. In what follows, we will refer to the chairs by their positions.

Informally, the Nagger wishes to maximize the number of chairs visited eventually, and the Mover wishes to minimize it. Formally, let A denote the algorithm used by the Nagger and B that of the Mover. A and B are both *adaptive*, that is, a player's move depends on the history of the moves of *both* players. Let $f_{A,B}(n)$ be the number of chairs occupied by the Nagger while the Nagger and the Mover use algorithms A and B respectively., Notice that if Nagger revisits the same position more than once, that does not affect f's. We define $f^*(n) = \max_A \min_B f_{A,B}(n)$ where the Mover and the Nagger have no bounds on the computation, A and B are *adaptive*, that is, dependent on each other's previous moves, and B is adversarial. Our problems are threefold:

- What is $f^*(n)$ for a given n? Clearly $f^*(n) \leq n$.
- What is an algorithm A^* for the Nagger such that $f_{A^*,B}(n) \geq f^*(n)$, irrespective of B. We will also be interested in bounding the number of moves in the game as well as the running time of the Nagger for computing A^* and would like both these parameters to be as small as possible.
- What is an algorithm B^* for the Mover that ensures that $f_{A,B^*}(n) \leq f^*(n)$, irrespective of A. We will be interested in minimizing the running time of the Mover for computing B^*.

Thus, our focus is not only on the mathematical structure that determines the value of $f^*(n)$ but also the algorithmics of realizing such a solution for the Nagger as well as the Mover. Our main result solves all three problems above. We leave similar problems for other variants of the Nagger-Mover games open, and describe the variants in Section 6.

To puzzlers, this problem may be reminiscent of the famous Josephus problem [6] where, starting from a circle of n positions, every mth chair is removed continually while going around the circle as many times as needed until only one chair remains. Our problem is obviously different from the Josephus problem of course: Nagger strategizes to pick the moves, the visited positions are not removed, etc. The Nagger-Mover game is an *adversarial* formulation of certain *occupancy* problems. Such a scenario arises when one hashes without chaining by probing a *series* of array locations p_i's determined by a modular, random hash function to store a given item [2]. Another such scenario is a random walk on a circular array where *both* the number of steps as well as the directions are randomly chosen, hence, d_i's are random and **+**, **-** are equally likely. The Nagger-mover game arises when we study such occupancy problems as a game between an adversary (Mover) who is trying to prevent the algorithm (Nagger), no matter how sophisticated, from occupying all the positions. Under the adversarial formulation, the question is whether the Nagger can even occupy all the positions eventually no matter how many moves—cover time—it takes (as we show, depending on n, this may be impossible); this is the dual of the question studied in occupancy problems which is how many moves are needed to occupy all the positions.

2 The Main Result

Theorem 1 *We have:*

- *If $n = 2^k$ for some k, then $f^*(n) = n$ and there is an $O(n)$ time algorithm A^* for the Nagger that produces a series of $O(n)$ moves achieving $f_{A^*,B}(n) = f^*(n)$ independent of B.*
- *If n is a prime, then $f^*(n) = n - 1$, there is an $O(n^2)$ time algorithm A^* for the Nagger that produces an $O(n^2)$ sequence of moves achieving $f_{A^*,B}(n) = f^*(n)$ independent of B, and there exists an $O(n)$ time algorithm for the Mover to insure that $f_{A^*,B}(n)$ is not larger.*
- *Else, $n = pm$ where p is the smallest odd prime factor of n. We have*

$$f^*(n) = \frac{(p-1)n}{p}.$$

There exist algorithms for the Nagger and the Mover that run in $O(n^2/p)$ time and realize $f^*(n)$ in $O(n^2/p)$ moves.

The main result gives a closed form expression for $f^*(n)$ as well as algorithms to realize it. As n increases from 1 to ∞, $f^*(n)$ is $1, 2, 2, 4, 4, 4, 6, 8, 6, 8, 10, 8, 12, 12, 10, 16, \ldots$. This sequence has not previously appeared in the Encyclopedia of Sequences [4].

3 Preliminaries

We will use $C(s, d, o)$ to denote $\{o + i \times d \mid 0 \leq i < s\}$, that is, it is a set of positions that start at o and are successively d apart for total size s.

We will dispense with the $n = 2^k$ case first. We need to show that the Nagger visits all chairs. The strategy (and the proof) can be described inductively. Consider $n = 2$ and $p_0 = 0$. Nagger's move is $d_1 = 1$: irrespective of **+** or **-** determined by the Mover, $p_1 = 1$. Hence $\{p_0, p_1\} = \{0, 1\}$. Assume $f^*(n/2 = 2^{k-1}) = 2^{k-1}$, that is, there exists a sequence $d_1, d_2, \ldots$ of Nagger's moves such that the set of p_i's visited is $\{0, \ldots, n/2 - 1\}$. We divide $C(n, 1, 0)$ into $C(n/2, 2, 0)$ and $C(n/2, 2, 1)$. Nagger first uses $2d_1, 2d_2, \ldots$ to visit all positions in $C(n/2, 2, 0)$; then, no matter what the resulting position is, the next $d_i = 1$ puts the Nagger at some position j in $C(n/2, 2, 1)$. Thereafter, Nagger's moves would be $2d_1, 2d_2, \ldots$ as before visiting all positions in $C(n/2, 2, 1)$. Therefore, all n positions are visited by the Nagger and the total number of moves is $n - 1$. The strategy for the Nagger is easy to generate in $O(n)$ time.

Example. For $n = 32$, Nagger's moves are (in order):

$$16; 8; 16; 4; 16; 8; 16; 2; 16; 8; 16; 4; 16; 8; 16; 1; 16; 8; 16; 4; 16; 8; 16; 2; 16; 8; 16; 4; 16; 8; 16.$$

More generally, let $n = 2^k$ for some k. Nagger calls the following sequence of moves:

$$(\frac{n}{2^1}; \frac{n}{2^2}; \frac{n}{2^1}); \frac{n}{2^3}; (\frac{n}{2^1}; \frac{n}{2^2}; \frac{n}{2^1}); \frac{n}{2^4}; (:::); ...; \frac{n}{2^k}; (:::);$$

Here $(:::)$ means a repetition of all the moves starting from the beginning of the sequence until, but not including, the move before the current $(:::)$. ∎

Henceforth, we will focus on other values of n.

4 Mover's Strategy

This is the easier part of Theorem 1.

Lemma 2 *Let $n = pm$ where p is the smallest odd prime of n, $p > 1$. There exists an $O(n)$ time algorithm for the Mover such that $f^*(n) \leq (p-1)n/p$.*

Proof. Recall that $p_0 = 0$. The algorithm for the Mover first involves determining p (this takes $O(n)$ time by the simple sieve method), and then choosing a k where $1 \leq k \leq p-1$. At the beginning, $p_0 \neq k \bmod p$. Mover's algorithm will insure that no matter what algorithm Nagger uses, Nagger never sits in any position in $C(n/p, p, k)$, that is, for all i, $p_i \neq k \bmod p$; this will immediately imply the lemma.

Quite simply, the algorithm is as follows. In the $i+1$st move of the Mover, if $p_i + d_{i+1} \bmod p = k$, then return **–**, else return **+**. Hence, $p_{i+1} = p_i + d_{i+1} \bmod p$ or $p_{i+1} = p_i - d_{i+1} \bmod p$. Observe that if $p_i + d_{i+1} = k \bmod p$ then $p_i - d_{i+1} \neq k \bmod p$ because otherwise, adding the two equations we have $2p_i = 2k \bmod p$, or $p_i = k \bmod p$ which is a contradiction. ∎

Rather than showing that there exist *some* n/p positions possibly dependent on the strategy of the Mover that Nagger can not reach, the proof above in fact shows something stronger: the Mover can prespecify a set of positions ($C(n/p, p, k)$ for a suitable k) irrespective of the strategy used by the Nagger, and insure that the Nagger never reaches those positions.

5 Nagger's Strategy

We will first prove the simple case.

Lemma 3 *Say n is prime. Then there exists an algorithm taking $O(n^2)$ time and producing a series of $O(n^2)$ moves for the Nagger such that $f^*(n) \geq n-1$ irrespective of the Mover's strategy.*

Proof. We prove that for any two different positions α and β, $\beta > \alpha$, Nagger can always sit in at least one of them. Let $\Delta = \beta - \alpha \bmod n$ and say Nagger is at position γ currently. Define

$$\mu_0 = (\gamma - \alpha)/\Delta \bmod n$$

where $/$ is performed on prime field $[n]$. Nagger proceeds in rounds visiting chair $q_i = \alpha + \mu_i 2^i \Delta \bmod n$ after i rounds, for $i = 0, 1, \ldots, n-1$. For even μ_i, let $\mu_i \leftarrow \mu_{i-1}/2$ and no move is performed. For odd μ_i, Nagger calls $\Delta 2^{i-1} \bmod n$ and moves to $q_i = \alpha + \mu_{i-1}2^{i-1}\Delta + \Delta 2^{i-1} \bmod n$ if Mover responds **+** and to $q_i = \alpha + \mu_{i-1}2^{i-1}\Delta - \Delta 2^{i-1} \bmod n$ if Mover responds **–**; hence $q_i = \alpha + \frac{\mu_{i-1}+1}{2}2^i\Delta \bmod n$ or $q_i = \alpha + \frac{\mu_{i-1}-1}{2}2^i\Delta \bmod n$. Setting $\mu_i = \frac{\mu_{i-1}+1}{2}$ if Mover responds **+** and $\mu_i = \frac{\mu_{i-1}-1}{2}$ otherwise, Nagger maintains the invariant visiting

$$q_i = \alpha + \mu_i 2^i \Delta \bmod n$$

after the i rounds. After $n-1$ iterations, $\mu_{n-1} = 0$ or 1 since $\mu_0 \leq n-1$ and $\mu_i < \mu_{i-1}$ unless they were 0 or 1. Hence,

$$q_{n-1} = \alpha + \mu_{n-1}2^{n-1}\Delta \bmod n.$$

By Fermat's little theorem [3], we know that $2^{n-1} = 1 \bmod n$ for prime n. Hence, the Nagger sits in either α or $\alpha + \Delta = \beta$.

The procedure above takes $O(n)$ moves by the Nagger and each move takes $O(1)$ time to determine by calculating all desired terms in each round using the corresponding values in the previous round. Starting with any position say $q_0 = 0$, this procedure is repeated whenever there are two or more unvisited positions for a total of $O(n)$ time, giving the lemma. ∎

Notice that the lemma above holds no matter the starting position. Also, combining Lemma 3 with Lemma 2 gives the tight value for $f^*(n)$ when n is prime. As an aside, the study of the length of the μ_i sequence with adversarial $+$ and $-$ may be of independent interest.

Example. Let $n = 11$ and all computations are $\bmod(11)$ so we will omit writing that explicitly. Initially the Nagger is at position $\gamma = 0$ and we show how the Nagger will always come to at least one of two different positions α or β. We choose $\alpha = 3$ and $\beta = 7$ for example; therefore $\Delta = 7 - 3 = 4$. Also $\mu_0 = (0-3)/4 = (8) * 3 = 2$ and $q_0 = \gamma = 0$. The first few steps are as follows.

1. Step $i = 1$: Because $\mu_0 = 2$ (even), $\mu_1 = \mu_0/2$. Nagger does not move, but we increment i and note that $q_1 = q_0 + 0 = \alpha + \mu_1 * 2^1 * \Delta$.

2. Step $i = 2$: Nagger calls $\Delta 2^1 = 8$. Therefore $q_2 = q_1 \pm 8$. The case $q_2 = q_1 - 8 = 3$ is trivial because from that point on $\mu_i = 0$ and q_i stays at α for every i. So we continue only the interesting scenario, ie., $q_2 = q_1 + 8 = 8$ and $\mu_2 = (1+1)/2 = 1$.

3. Step $i = 3$: Nagger calls $\Delta 2^2 = 8 * 2 = 5$. Therefore $q_3 = q_2 \pm 5$. The interesting scenario is $q_3 = q_2 + 5 = 2$ and $\mu_3 = (1+1)/2 = 1$. (All μ_i are 1s or 0s from now on.)

Thereafter, the moves are described below.

Step i	Nagger's Move	q_{i+1}	Interesting Case
4	$\Delta 2^3 = 10$	$q_4 = q_3 \pm 10$	$q_4 = q_3 + 10 = 1$
$i = 5$	$\Delta 2^4 = 9$	$q_5 = q_4 \pm 9$	$q_5 = q_4 + 9 = 10$
$i = 6$	$\Delta 2^5 = 7$	$q_6 = q_5 \pm 7$	$q_6 = q_5 + 7 = 6$
$i = 7$	$\Delta 2^6 = 3$	$q_7 = q_6 \pm 3$	$q_7 = q_6 + 3 = 9$
$i = 8$	$\Delta 2^7 = 6$	$q_8 = q_7 \pm 6$	$q_8 = q_7 + 6 = 4$
$i = 9$	$\Delta 2^8 = 1$	$q_9 = q_8 \pm 1$	$q_9 = q_8 + 1 = 5$
$i = 10$	$\Delta 2^9 = 2$	$q_{10} = q_9 \pm 2 = \{3, 7\}$	$\{3, 7\} = \{\alpha, \beta\}$

The Nagger is thus at either α or β. Notice that the easiest way to calculate the new $\Delta 2^{i-1}$ is to multiply the old $\Delta 2^{i-2}$ by 2. ∎

Now we turn to the general result. We will first provide a characterization result similar to above for $n \neq 2^k$, and use that to derive an algorithm for the Nagger.

Lemma 4 *Say $n \neq 2^k$, current position is K and there are two distinct positions I and J such that $J > I$ and $gcd((J-I), n) = 2^f$ for some nonnegative integer f. (Here, gcd() is the greatest common divisor.) Then there is a set of $O(n)$ moves for the Nagger such that starting from K, Nagger reaches either I or J, no matter the responses of the Mover. The moves can be found in time $O(n^2)$.*

Proof. With respect to the pair of distinct positions I and J, we will classify all positions $0, \ldots, n-1$ into classes as follows. Class C_0 contains I and J. Class C_1 all the positions $p \notin C_0$ such that there exists a move d for which both $p + d \bmod n \in C_0$ and $p - d \bmod n \in C_0$, that is, irrespective of Mover's response, Nagger ends in a position in C_0. More generally, class C_m contains all positions $p \notin C_0 \cup C_1 \cup \cdots \cup C_{m-1}$ such that there exists a move d for which both $p + d \bmod n \in C_0 \cup C_1 \cup \cdots \cup C_{m-1}$ and $p - d \bmod n \in C_0 \cup C_1 \cup \cdots \cup C_{m-1}$. We also define $C^*_m = C_0 \cup C_1 \cup \cdots \cup C_m$.

We will prove that if $|C_{m+1}| = 0$ for any m, then, all positions $0, \ldots, n-1 \in C^*_m$, that is, as we construct C_m in increasing m's, we keep adding new positions in each m or we have exhausted all positions. We will prove this by contradiction. Say, we have $|C_{m+1}| = 0$ for some m and there exist positions $\alpha \notin C^*_m$. Consider all positions in C^*_m in increasing order: $\alpha_1, \alpha_2, \ldots$.

- If $\alpha_i - \alpha_{i-1}$ is even for some i, then there are odd number of positions in-between: the position equidistant from α_{i-1} and α_i, namely, $\alpha_{i-1} + (\alpha_i + \alpha_{i-1})/2$, belongs to C_{m+1} giving the contradiction.

- Say $\alpha_i - \alpha_{i-1}$ is odd for all i. Consider any three consecutive positions $\alpha_{\ell-1}, \alpha_\ell, \alpha_{\ell+1}$ from C^*_m. We can observe that even class C^*_1 has 3 or more positions, so for larger m, we can always find three positions. $\alpha_{\ell+1} - \alpha_{\ell-1}$ must be even. Therefore, the midpoint $\alpha_{\ell-1} + \frac{\alpha_{\ell+1}+\alpha_{\ell-1}}{2}$ is either not in C^*_m in which case it belongs to C_{m+1} giving the contradiction, or it equals α_ℓ. Repeating this argument for each of the triples, if we do not obtain a contradiction, we have that $\alpha_1, \alpha_2, \ldots$ are arranged as vertices of a regular polygon of less than n vertices with sides $\alpha_i - \alpha_{i-1} = L$ for all i, L being odd, $L \geq 3$. Therefore $n = aL$ for some positive integer a. Also, since $I, J \in \{\alpha_1, \alpha_2, \ldots\}$, they are vertices of the polygon and we have $J - I = bL$ for some positive integer b. This implies that $gcd(J - I, n) = cL$ for some positive integers c. Then, $gcd(J - I, n) \neq 2^f$ which contradicts the assumption in the lemma.

We have proved that C^*_m contains all positions $0, 1, \ldots, n-1$ for some $m \leq n-1$. We will now show that we can efficiently construct the classes iteratively and find the algorithm for Nagger to reach I or J from K. Trivially, we can construct $C_0 = \{I, J\}$.

C_1 is constructed as follows. If n is odd, C_1 contains $I + (J-I)/2$ or $I + (J-I+n)/2 \bmod n$ whichever one is an integer (the position equidistant from I, J will be in C_1 and of the two possible midpoints, (only) one of them will be an integer since n is odd). If n is even, C_1 contains two positions that are diametrically opposite to I and J ($I + n/2 \bmod n$ and $J + n/2 \bmod n$ resp.)

and the two positions that are equidistant from I and J, if they exist. Their existence depends on precise values of I and J. Notice that it is not possible that I and J are diametrically opposite to each other since in that case $gcd((J-I), n) = gcd(n/2, n)$ which can not be 2^f since $n \neq 2^k$. Therefore, $|C_1^*| \geq 3$ no matter n even or odd.

Assume we have constructed classes $C_0, \ldots, C_{m-1}$. We can construct C_m as follows. If n is even, for every position $p \in C_{m-1}^*$, add its diametrically opposite position $p+n/2 \bmod n$ to C_m if it is not in C_{m-1}^*. Also, for every pair of positions in C_{m-1}^*, take any position that is equidistant from both if it exists and does not belong to C_{m-1}^* already. It is easy to see that this procedure correctly constructs C_m's from the definition. The entire construction can be efficiently implemented as follows: when we add a position p to C_m, we generate all midpoints between p and any $q \in C_m^*$ as well as p's diametrically opposite point $p + n/2 \bmod n$ whenever they exist and add it to a list l_m if they do not belong to C_m^*. In new iteration $m+1$, we add all the positions in l_m to C_{m+1} one after another and populate l_{m+1} in the process. The entire process takes $O(n^2)$ time.

Each $i = 0, \ldots, n-1$ belongs to some class $C_{h(i)}$. The construction above also shows how to determine for each i, a move d_i such that Nagger can move to a position in class $C_{h(i)-1}$ starting from i, irrespective of Mover's response. We can store this d_i for each i. Now, say K belongs to some set $C_{h(k)}$. The Nagger can choose d_k to move to a position in $C_{h(k)-1}$ and iterating this process from the new position each time, finally end in C_0; that proves the lemma. ∎

Example. Let $d_+(u, v)$ be the number of unit distances one must travel clockwise from u in order to reach v. Suppose $n = 12 = 3 * 2^2$. We take two positions I, J $(I < J)$ such that $\gcd(J-I), n = 12) = 2^0$. In this example we will construct the classes starting with $I = 3$ and $J = 8$ in the lemma above.

We have $C_0 = \{I, J\} = \{3, 8\}$. We build class C_1. Initially $C_1 = \phi$. We check all candidates positions add them to C_1 if they are not in C_0 already.

- $8 + n/2 = 8 + 6 = 2 \notin C_0 \Rightarrow C_1 = C_1 \cup \{2\}$. Similarly we add position 9 to C_1.
- Since $\frac{d_+(3,8)}{2}$ and $\frac{d_+(8,3)}{2}$ are not integers, they are not added to C_1.

We have constructed class C_1 and we also record the move sizes that led to each position being added to C_1. Thus, $C_1 = \{2(6), 9(6)\}$. Now, we build C_2. Initially $C_2 = \phi$.

- Since $8 + n/2, 3 + n/2, 9 + n/2, 2 + n/2$ are already in C_1^*, they are not added to C_2.
- Since $\frac{d_+(3,9)}{2}$ is integer and $3 + \frac{d_+(3,9)}{2} = 6 \notin C_1^*$, 6 is added to C_2. Similarly we add positions $0, 5, 11$ to get $C_2 = \{6(3), 5(3), 11(3), 0(3)\}$.

Now we build C_3. Initially $C_3 = \phi$.

- Since $8 + n/2, 3 + n/2, 9 + n/2, 2 + n/2, 6 + n/2, 5 + n/2, 0 + n/2, 11 + n/2$ are already in C_2^*, they are not added to C_3.
- Since $\frac{d_+(0,2)}{2}$ is an integer and $0 + \frac{d_+(0,2)}{2} = 1 \notin C_2^*$, we add 1 to C_3. Similarly we add positions $4, 7, 10$ to get $C_3 = \{1(1), 4(1), 7(1), 10(1)\}$.

Now we have exhausted all positions around the table. To summarize, the classes are as follows.

$$\boxed{\begin{array}{l} C_0 = \{I, J\} = \{3, 8\} \\ C_1 = \{2(6), 9(6)\} \\ C_2 = \{6(3), 5(3), 11(3), 0(3)\} \\ C_3 = \{1(1), 4(1), 7(1), 10(1)\} \end{array}}$$

We can invert this list to get $C_{h(i)}$ for each i. ∎

Using the characterization lemma above, we can prove the main result for the Nagger, except for the running time or the number of moves.

Lemma 5 *Let $n = pm \neq 2^k$ where p is the smallest odd prime factor of n and m, k are nonnegative integers. There exists an $O(n^3)$ time algorithm for the Nagger that produces a sequence of $O(n^2)$ moves such that $f^*(n) \geq (p-1)n/p$.*

Proof. From any current position K, if there exist two distinct positions I and J, $J > I$, such that $gcd(J - I, n) = 2^f$ for some integer f, then Nagger employs the algorithm in Lemma 4 to reach either I or J and repeats the procedure from the new current position. The whole procedure stops when Nagger is at position K and can not find two positions I and J satisfying $gcd(J-I, n) = 2^f$. We claim that at most n/p positions remain unvisited. Otherwise, partition all n positions into $\frac{n}{p}$ consecutive groups each with p positions, as follows: $(0, 1, 2, \ldots, p-1)$, $(p, p+1, p+2, \ldots, 2p-1)$, ..., $(n-p, n-p+1, \ldots, n-1)$. By the pigeonhole principle, there will be at least one group containing two unvisited positions, say I and J. We have $|J - I| \leq p-1$ since they both belong to the same group, and since p is the smallest odd prime factor of n, $gcd(J - I, n)$ has to be a power of 2 (including $2^0 = 1$) which is a contradiction. ∎

The strategy above for the Nagger proves the main theorem and yields insight into the structure in the problem via Lemma 4. Still, the running time is quite high. We will give an alternative solution that is more efficient.

Lemma 6 *Let $n = pm \neq 2^k$ where p is the smallest odd prime factor of n and m, k are nonnegative integers. There exists an $O(n^2/p)$ time algorithm for the Nagger that produces a sequence of $O(n^2/p)$ moves such that $f^*(n) \geq (p-1)n/p$.*

Proof. We divide $C(n, 1, 0)$ into p sets, $C(n/p = m, p, i)$, for $i = 0, 1, ...p-1$. Nagger proceeds in two phases and inductively uses the same two phase strategy on smaller problem instances.

Phase 1. Nagger applies the algorithm in Lemma 3 for $C(p, 1, 0)$ and gets into positions in different $C(n/p, p, i)$'s. Formally, if the solution to $C(p, 1, 0)$ is the sequence of positions $p_0, p_1, \ldots$ depending on the Mover's strategy, Nagger's moves will be the same—p_0, p_1 and so on. The invariant is that the positions the Nagger visits are congruent by $\mathrm{mod} p$ to the positions he would have visited if the table size were p. When Nagger gets to a position β_i in $C(n/p, p, i)$ for the first time, Nagger inductively uses the two phase algorithm for $C(n/p, p, \beta_i)$ by replacing each move d_j in the algorithm for $C(n/p, 1, 0)$ by $pd_j + \beta_j$. Thus, Nagger visits $f^*(n/p)$ positions in each of the $C(n/p, p, i)$ sets visited. Phase 1 ends after $p-1$ sets have been thus navigated.

At the end of this phase, say γ is the unexplored set $C(n/p, p, \gamma)$. There may also exist positions α_β in $C(n/p, p, \beta \neq \gamma)$ that Nagger has not visited; if no such positions exist, Phase 2 is not needed.

Phase 2. For each α_β in turn, Nagger is in some position and there are at least two unoccupied positions (α_β is unoccupied and one or more positions in $C(n/p, p, \gamma)$), and Nagger applies the algorithm in Lemma 3 on $C(p,1,0)$ to get into a position in either $C(n/p,p,\gamma)$ or $C(n/p,p,(\beta+\gamma)/2)$ where division / is performed on prime field $[p]$.

If Nagger reaches a position in $C(n/p,p,\gamma)$, Nagger explores it fully recursively by using the two phase algorithm, replacing each move d_j in the algorithm for $C(n/p,1,0)$ by pd_j. Thus, Nagger occupies $f^*(n/p)$ positions in $C(n/p,p,\gamma)$ and the phases are complete.

Else, Nagger reaches a position δ in $C(n/p,p,(\beta+\gamma)/2)$. Nagger's next move is then $\alpha_\beta-\delta$. If Mover responds **+**, Nagger reaches $\delta+(\alpha_\beta-\delta)$ which is α_β. Else, if Mover responds **–**, then Nagger reaches $\delta+(\delta-\alpha_\beta)=2\delta-\alpha_\beta$ which is a position in $C(n/p,p,\gamma)$. This is because say $\delta=(\beta+\gamma)/2+p\ell_1$ and $\alpha_\beta=\beta+p\ell_2$ for some non-negative integers ℓ_1 and ℓ_2. Then, $2\delta-\alpha_\beta=\gamma+p(2\ell_1-\ell_2)$.). Now, the Nagger finishes the phase as above. ∎

Example. Let $n=15=3*5$. We divide $C(15,1,0)$ into 3 subsets $C(n/p=5,p=3,i=0,1,2)$, ie., $C_0=\{0,3,6,9,12\}$, $C_1=\{1,4,7,10,13\}$, and $C_2=\{2,5,8,11,14\}$. Initially the Nagger navigates C_0 until at least $f^*(n/p=5)=4$ out of 5 possible positions are visited. Nagger achieves this by using the two phase algorithm inductively for $C(n/p=5,1,0)$ but replaces each move d_j by $pd_j=3d_j$. At the end, the Nagger is in some position in C_0, say 9. Then the Nagger applies the algorithm from Lemma 3 for $C(p=3,1,0)$ to get into any position from current ± 1, current $\pm 2,\ldots$, current $\pm(p-1)$, which in our case are $7,8,10$, or 11. Now the Nagger will be in either C_1 or C_2.

Suppose the Nagger is in C_2 (hence, $\gamma=1$). Again recursively applying the two phase algorithm the Nagger visits at least 4 out of 5 positions in C_2. If all positions from C_0 and C_2 have been visited, the process ends. Else, some α_β from $C(n/p=5,p=3,\beta=0\text{or }2)$ $(\beta\neq\gamma)$ has not been visited. Also, all positions in $C(n/p=5,p=3,\gamma)$ remain unvisited. For example, say $\alpha_\beta=5$ $(\beta=2)$ has not been visited yet and the Nagger is at position 11.

Using the algorithm from Lemma 3 for $C(p=3,1,0)$ the Nagger goes either into position $10\in C(n/p=5,p=3,\gamma=1)$ or to position $12\in C(n/p=5,p=3,(\beta+\gamma)/2=0)$. In the first case, after getting into position 10, the Nagger gets into at least 4 out of 5 positions from $C(n/p=5,p=3,\gamma=1)$. The process ends there. In the second case, after getting into the position $\delta=12$, the Nagger calls $\alpha_\beta-\delta=5-12=8$.

- If the Mover's response is $-$, the Nagger goes to position $12-8=4\in C(5,3,\gamma=1)$ and proceeds as above.
- If the Mover's response is $+$, the Nagger goes to position $12+8=5$ which is the previously unvisited α_β.

Continuing in this way the Nagger either visits all unvisited positions in $C(n/p=5,p=3,\beta=0,or2)$ or goes to a position in $C(n/p=5,p=3,\gamma=1)$ and then visits at least 4 out of the 5 positions in $C(n/p=5,p=3,\gamma=1)$. ∎

We can now count the number of visited positions. There are two possibilities: either all positions in $C(n/p,p,i\neq\gamma)$ are visited, or all $C(n/p,p,i)$'s are visited using the two phase strategy. Hence, we have

$$f^*(n)\geq\min\{(p-1)n/p,pf^*(n/p)\}$$

Observe that inductively we can set $f^*(n/p)\geq\frac{(p-1)(n/p)}{p}$ since p is no larger than the smallest odd prime factor of n/p. That proves $f^*(n)\geq\frac{(p-1)n}{p}$.

We can upper bound the number of moves, say $g(n)$. Nagger needs $O(p^2)$ moves in all to get into all $C(n/p, p, i \neq \gamma)$'s. Together for Phase 1 and Phase 2, the total number of moves for navigating within the various $C(n/p, p, i)$'s is upper bounded by $(p-1)g(n/p)+(n/p)(1/p)O(p)+g(n/p)$ where the first summand is for navigating within $C(n/p, p, i)$'s in Phase 1, the second summand is for the at most $1/p$ fraction of the unvisited nodes in $C(n/p, p, i \neq \gamma)$'s each taking $O(p^2)$ moves in the worst case to get to, and the third summand is for navigating $C(n/p, p, \gamma)$ if the Nagger reaches positions in it. Hence,

$$g(n) \leq O(p^2) + p * g(n/p) + O(n/p).$$

Since $p \leq \sqrt{n}$, we have the simpler recurrence, $g(n) \leq p * g(n/p) + O(n)$. This can not be solved directly as a recurrence, since some of the subproblems may lead to $g(q)$ for prime q where Lemma 3 applies and $g(q) = O(q^2)$ in that case. Still, we can verify that $g(n) = O(n^2)$ no matter what n. We can get an improved bound by using this upper bound for the right hand size so $g(n) \leq O(n^2/p)$ for n satisfying our lemma. That also gives the running time since we can determine all the primes that are needed and maintain necessary data structures in time and space linear in the number of moves and n. ∎

The running time of the algorithm for the Nagger is a nonuniform function over n; similarly, the number of moves $f^*(n)$ is also non-monotonic with increasing n. Both these factors depend on the simple number-theoretic property of n, namely, its smallest odd prime factor p.

6 Concluding Remarks

We have initiated the study of two person games we call Nagger-Mover games that may be thought of as an adversarial formulation of occupancy problems. We have proved tight existential bounds and shown matching algorithmic strategies on the basic version of the problem. An immediate open problem is if we can improve the running time (or the number of moves) of the algorithm. Also, it is of mathematical interest to understand the function n/p where p is the smallest odd prime factor of n.

There are a number of variants such as the following that are of great interest.

- *Dual problems.* Say we specify a bound m, and want to maximize $f_m^*(n)$ using only m rounds of moves by the Nagger and the Mover. What is $f_m^*(n)$ and what are associated strategies?

- *Off line strategy construction.* Given a sequence $d_1, d_2, \ldots, d_k$ of the Nagger's moves, design corresponding **+**'s or **–**'s for the Mover so that $f^*(n)$ is minimized.

- *Limiting strategies.* Limit Nagger and/or Mover to use only *non-adaptive* strategies, that is, Nagger (resp. Mover) prespecifies a sequence of moves without knowing Mover's (resp. Nagger's) responses, and evaluate the sequence when Mover (resp. Nagger) later specifies responses in order to optimize their goal. What is $f^*(n)$ and what are associated non-adaptive strategies?

- *Limit revisits.* Suppose one upper bounded the number of times the Nagger may visit a position, by say k. Then, what is $f_k^*(n)$ and what are associated strategies?

- *Random strategies.* Suppose the Nagger or the Mover used random strategies. The problem is to estimate the expected $f^*(n)$. This is akin to random walks with random steps, but differs in having either one of the player being deterministic or adversarial.
- *Limiting information.* Suppose the Nagger and/or the Mover do not know n. This is similar to graph exploration problems studied in theoretical computer science with limited number of tokens or memory. [1].
- *Making each player nag as well as move.* The roles the players play somewhat overlap since Nagger moves and Mover nags the other by restricting the movement. In other two-person games too, this type of ambiguity is seen (for example, the Pusher-Chooser games of [5]). We propose a modified game in which Nagger and Mover alternate their roles each round. Positions visited by Nagger are colored red, and those by Mover are colored blue. Mover is restricted to non-red positions and Nagger to non-blue. What are suitable strategies and how many positions are visited by each player?

7 Acknowledgements

We thank Graham Cormode, Uriel Feige, Richard Guy, Susan Hope and Peter Winkler for helpful discussions. Yin Zhang independently calculated $f^*(n)$.

References

[1] M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The Power of a Pebble: Exploring and Mapping Directed Graphs. Proc. of the 30th Annual ACM Symposium on Theory of Computing (STOC), pages 269-278, 1998.

[2] D. Knuth. Sorting and Searching, Addison-Wesley, 1998.

[3] I. Niven and H. Zuckerman. An introduction to the theory of numbers. John Wiley and Sons Inc, 1960.

[4] N. Sloane. On line Encyclopedia of Sequences. http://www.research.att.com/ njas/sequences/

[5] J. Spencer and P. Winkler. Three thresholds for a liar. *Combinatorics, Probability and Computing*, 1:1 (1992), 81–93.

[6] E. Weisstein. *Josephus Problem.* From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/JosephusProblem.html