Uniform random generations and rejection method(II) with trivial majorant

Laurent Alonso

Uniform random generations and rejection method(II) with trivial majorant

Laurent Alonso*

August 23, 2023

Abstract. We present a simple algorithm to help generate simple structures: 'Fibonacci words' (i.e., certain words that are enumerated by Fibonacci numbers), Motzkin words, Schröder trees of size n. It starts by choosing an initial integer with uniform probability (i.e., $\frac{1}{n}$ for some n), then uses a basic rejection method to accept or reject it. We show that at least for these structures, this leads to very simple and efficient algorithms with an average complexity of $O(\sqrt{n}\log n)$[1] that only use small integers (integers smaller than n^2). More generally, for structures that satisfy a basic condition, we prove that this leads to an algorithm with an average complexity of $O(n)$. We then show that we can extend it to be more efficient if a distribution has a long 'trail' by showing how we can generate Partial Injections of size n in $O(n^{1/4}\log n)$ and Motzkin left factors of size n and final height h in $O(\sqrt{n}\log n)$.

Key words. Generation, uniform, rejection

1 Introduction

In [Alo94], Alonso proposes a method to uniformly generate a Motzkin word by first choosing the number of steps $(1,1)$: m with an appropriate probability (using a rejection method) and then generates a Motzkin word with m steps $(1,1)$. The algorithm is simple and efficient (its complexity is $O(n)$) and uses only small integers (less than or equal to n). In [GBN10], Gouyou-Beauchamps et al. attempt to generalize this problem and show that there are similar algorithms for the Partial Injection problem and the colored Motzkin word. Alonso[Alo23] attempts to generalize this same problem using a slightly different method and shows that one can use an almost unique algorithm to generate a Fibonacci

*INRIA Nancy-Grand Est, 615, rue du Jardin Botanique, 54600 Vandoeuvre-lès-Nancy, France. Email: Laurent.Alonso@inria.fr

[1] As we only present algorithms that use small integers in this paper, we will use the unit cost RAM model to give the complexity, you can get the log cost RAM model by multiplying this complexity by a factor $\log n$.

word, a Motzkin word, a Schröder tree and that one can extend his method to generate a left Motzkin word of size n and final height h.

All these methods start by choosing an initial binomial distribution $B(m)$ (equiprobable or not)[2] that has a "similar" shape to the final distribution to obtain an upper bound on the final distribution; they then generate a number m with a probability corresponding to $B(m)$ and use a rejection algorithm to accept or reject it.

But is this a good idea? In this paper, we try to challenge this assumption and more precisely to study what happens if we do the opposite: try to generate a number m with the basic uniform distribution and try to accept or reject this value of m. Surprisingly, we show that for almost all the presented problems, we obtain simpler algorithms and much lower average complexity; we also note that the resulting algorithms are simple to obtain and code, that checking their validity is straightforward, that obtaining a coarse upper bound on the average complexity is possible but that computing their average complexity is tedious.

If we compare the obtained algorithms with those presented in [Alo23], we can note that the algorithms are simpler to implement, more efficient and it is easier to prove that the obtained algorithm is valid. However,

- one can replace the first phase of the algorithms of [Alo23] which consists in choosing an integer n with a binomial distribution using the methods presented in this paper; this will lead to algorithms with a complexity similar to the ones presented in this paper,
- even though it is easier to prove that the average complexity is in $O(n)$ for the presented problems, the computation of a more accurate upper bound uses basically the same arguments as those of [Alo23] and remains therefore "complex",
- the methods presented in [Alo23] correspond to the execution of a loop which has a "large" complexity (typically in $O(n)$ or in $O(\sqrt{n}\log n)$ if we use the method recommended here) very few times (typically $O(1)$ times). In this paper, it is the opposite: the internal time complexity is very low (typically in $O(\log n)$) but we have to execute the main loop many times (typically $O(\sqrt{n})$ times); this can be prohibitive if you want to use this method as a brick of a more complex algorithm.

In Section 2, we first present a generic algorithm and obtain some upper bounds on the complexity; the principal result is that if $F(m)$ is a unimodal function (with $0 \leq m \leq n$) and $\frac{F(m+1)}{F(m)}$ is decreasing, we obtain a valid algorithm with average complexity in $O(1+n)$. In the next section, we apply this method to choose a number m with probability $\frac{B_M(m)}{\sum_m B_M(m)}$ where $B_M(m) = \begin{pmatrix} M \\ m \end{pmatrix}$ is the equiprobable binomial distribution and we show that we can obtain a more accurate average complexity in $O(\sqrt{M}\,\log M)$. In Section 4, we use this

[2] or a slight variation of a binomial distribution

algorithm to improve the first step of the generation of Fibonacci word, Motzkin word and Schröder tree problems and we obtain an algorithm whose average complexity is $O(\sqrt{n}\log n)$ and $O(n^{3/4})$ for the Partial Injection problem. In Section 5, we study whether we can improve the average complexity when the final distribution has a very long tail and we show in Section 6 that this allows us to obtain an efficient solution for working with a non-equiprobable binomial distribution with $p = 1$ and to improve the first step of the generation of the partial injection with complexity $O(n^{1/4}\log n)$ and of Motzkin left factors of size n and final height h with complexity $O(1+\sqrt{n-h}\log n)$ in Section 7. Finally, we conclude with a discussion.

2 Notations and algorithm

As in [Alo23], we will use:

- $[P]$ the Kenneth Iverson's convention[GKP88](page 24): $[P] = 1$ if P is true and $[P] = 0$ if P is false,
- $random(k)$ the function that returns an integer between 0 and $k-1$ with probability $\frac{1}{k}$.

Let $F(m)$ be a unimodal function defined on $[0, maxM]$ and suppose for simplicity that $f(m) > 0$ when $0 \leq m \leq maxM$[3]. Note:

- $M = \min \operatorname{argmax}_m F(m)$ the smallest value of m which maximizes $F(m)$,
- $\mathcal{R}F(m) = \frac{F(m+1)}{F(m)}$ when $0 \leq m < maxM$,
- $T = \sum_{m=0}^{maxM} F(m)$.

As $\mathcal{R}F(m) > 1$ iff $m < M$, we can use the following formula which relates a coefficient $F(m)$ to the maximum value: $F(M)$

- $F(m) = \prod_{j=m}^{M-1} \frac{1}{\mathcal{R}F(j)}\, F(M)$ when $m < M$,
- $F(m) = \prod_{j=M}^{m-1} \mathcal{R}F(j)\, F(M)$ when $m \geq M$,

to obtain a rejection algorithm allowing to accept a value m with probability $\frac{F(m)}{F(M)}$ with complexity $O(|m-M|)+O(1)$.

This gives a simple rejection algorithm (Algorithm 1) to draw a number m with probability $\frac{F(m)}{\sum_{i=0}^{maxM} F(i)}$ by repeating the operations,

- first chooses an integer m in $[0, maxM]$ with probability $\frac{1}{maxM+1}$,
- accepts it with probability $\frac{F(m)}{F(M)}$,

[3] As soon as there exists at least one value of m such that $F(m) > 0$, we can replace this condition by $F(m) \geq 0$, the following results will still be valid but we'll have to rewrite the proofs to avoid dividing 0 by 0.

until we accept a value m. Indeed, we can verify that inside a loop, each value m in $[0, maxM]$ is chosen with a probability $\frac{F(m)}{(maxM+1)F(M)}$, which is sufficient to prove that the algorithm is valid.

Algorithm 1 Choose m with probability $F(m) \Big/ \sum_{i=0}^{maxM} F(i)$:

```
1:  function CHOOSE(M, maxM, RF)
2:      while true do
3:          m ← random(maxM + 1)
4:          ok ← True
5:          if m < M then
6:              i ← m
7:              while ok AND i < M do
8:                  if random(numer(RF(i))) ≥ denom(RF(i)) then
9:                      ok ← False
10:                     break
11:                 end if
12:                 i ← i + 1
13:             end while
14:         else
15:             i ← m − 1
16:             while ok AND i ≥ M do
17:                 if random(denom(RF(i))) ≥ numer(RF(i)) then
18:                     ok ← False
19:                     break
20:                 end if
21:                 i ← i − 1
22:             end while
23:         end if
24:         if ok then returns m end if
25:     end while
26: end function
```

We can derive generic upper bounds on the average complexity:

Theorem 1 *If we assume that the tests of lines 8 and 17 can be performed with complexity $O(1)$ and if $\mathcal{R}F(m) > 1$ iff $m < M$:*

- *the average complexity of the algorithm is $O(1 + maxM^2)$,*
- *if $\mathcal{R}F(m)$ is a decreasing function, the average complexity of the algorithm is $O(1 + maxM)$.*

which is proved in the Appendix A. This suggests that at least when $\mathcal{R}F(m)$ is a decreasing function, this simple algorithm (which uses only small numbers, i.e. numbers less than or equal to $\max(maxM + 1, numer(\mathcal{R}F(m)), denom(\mathcal{R}F(m)))$) has a promising average complexity. In the following sections, we will study some problems and try to establish more precise upper bounds on their complexity.

3 Equiprobable binomial distribution

Let us choose a number N and define define $F(m) = \binom{N}{m}$ when $0 \leq m \leq N$, here we want to draw a number m with probability $\frac{F(m)}{\sum_i F(i)} = \binom{N}{m} \Big/ 2^N$.

The simplest method to generate such number m is to draw N random integers 0 or 1 and count the number of 0. This gives an algorithm in $O(N)$ that uses only 0 and 1 random integers and counters for the numbers of $0, \ldots$ (which store some small integers, i.e. integers less than or equal to N).

More efficient methods (see [Hö93]) exist, but they require some random variates: this assumes that it is possible to generate random reals uniformly distributed in $[0, 1]$ and to do computations with these reals, an assumption that is very strong as computers only used floating points with limited precision.

We can try to use the function CHOOSE with $maxM = N$, $M = \lfloor \frac{N}{2} \rfloor$, $\mathcal{R}F(m) = \frac{F(m+1)}{F(m)} = \frac{N-m}{m+1}$ because F is a unimodal function that takes its maximum value in M and $\mathcal{R}F(m)$ is decreasing. This algorithm uses only integers less than or equal to $N+1$ and its average complexity is $O(N)$. However, we can get a more precise upper bound on the complexity which is proved in Appendix B:

Proposition 1 *This algorithm has an average complexity of* $O(\sqrt{N} \log N)$.

We can also note that each value of m *is drawn with an average complexity of* $O(\sqrt{N} \log N) + O(|m - M|)$.

Some remarks:

- The worst case complexity is infinite. If necessary, we can obtain a worst-case complexity in $O(N)$ by counting the number of times the function *random* is called, then if we get a failure and this number is greater than M, we can return to the classical method.
- Although the average complexity is much lower than that of the classical algorithm, it must be mitigated by the fact that choosing certain values of m (e.g. $m = 0$) always takes time in $\Theta(N)$. Therefore, if you want to draw uniformly a value m with a binomial distribution with $N = 10^{20}$, using this algorithm is not a good idea because choosing the value $m = 0$ will require a very long waiting time.
- As we will see in the next section, it is important because it suggests that if we want to use the method proposed by Alonso [Alo23] or Gouyou-Beauchamps et al.[GBN10], for simple structures, we can simply start by choosing a uniform distribution and obtain a very efficient and easy to implement algorithm.

Finally, for $N = 10, 10^3, \ldots$, we do some simulations: for each value of N, we draw 1000 random values of m and count the minimum, the maximum, the average number of times the main loop is called; we also count the number of

tests we do in lines 8 and 17. We get:

N	Loop			Test		
	ave	min	max	ave	min	max
10	2.633	1	14	2.916	0	19
10^3	26.431	1	152	66.95	0	340
10^5	252.936	1	1789	942.231	6	5774
10^7	2478.474	3	18054	12002.12	202	85645
10^9	25613.349	28	177612	152499.464	2838	1000468

4 Applications

If we look at the algorithms presented in Gouyou-Beauchamps et al. [GBN10] and in Alonso[Alo23], several[4] starts by drawing a value m with a binomial distribution for some value of $maxM$ with $p = 1$ and $q = 1$ (sometimes with a minor modification) and then accepts a value m with $O(|M - m|)$ operations. We can therefore simply adapt these algorithms by using our algorithm to generate the first value of m, which will decrease the average time complexity of finding the final value m; for most of these algorithms, the average complexity of choosing m will decrease to $O(\sqrt{n}\log n)$ except for Partial Injections whose complexity will decrease to $O(n^{3/4}\log n)$.

We can then note that these algorithms start by drawing an integer m with probability $\frac{F_n(m)}{\sum_m F_n(m)}$ where $F_n(m)$ is a unimodal function such that $\mathcal{R}F_n(m) > 1$ if $m < M$. Therefore, we can obtain a valid algorithm to draw a number m with a suitable probability by choosing:

- for Fibonacci word: $M = \left\lceil \frac{5n-3-\sqrt{5n^2+10n+9}}{10} \right\rceil$, $maxM = \lfloor n/2 \rfloor$, $\mathcal{R}F(m) = \frac{(n-2m)(n-1-2m)}{(m+1)(n-m)}$,
- for Motzkin word: $M = \lfloor n/3 \rfloor$, $maxM = \lfloor (n-1)/2 \rfloor$, $\mathcal{R}F(m) = \frac{(n-2m)(n-2m-1)}{(m+1)(m+2)}$,
- for Schröder tree: $M = \left\lceil -1 + \frac{\sqrt{2n^2+2n}}{2} \right\rceil$, $maxM = n$, $\mathcal{R}F(m) = \frac{(n+m+1)(n-m)}{(m+1)(m+2)}$,
- for Partial Injection: $M = \left\lceil \frac{n+1-\sqrt{4n+5}}{2} \right\rceil$, $maxM = n$, $\mathcal{R}F(m) = \frac{(n-m)^2}{(m+1)}$.

We have:

Proposition 2 *With these choices of values, a number m is drawn with average complexity C_n with :*

- $C_n = O(n^{3/4})$ *for the Partial Injection problem,*

[4] Generation of a Motzkin word, a partial injection, a Fibonacci word, a Schröder tree and a Motzkin left factor of final height h when $h \leq \frac{3n}{7} - 5$

- $C_n = O(\sqrt{n}\log n)$ *for the other problems.*

These algorithms use only small integers (i.e. integers smaller than n^2*).*

We can also note that each value of m *is drawn with an average complexity of* $C_n + O(|m - M|)$*.*

Proof. The average number of executions of the main loop is $\frac{(maxM+1)F_n(M)}{\sum_m F_n(M)}$ (see beginning of Appendix A); using [GBN10] and [Alo23] results, we obtain $O(n^{3/4})$ for the Partial Injection problem and $O(\sqrt{n})$ for the other problems.

We now need to calculate the average time we spend in a loop. As in proof A, suppose we have chosen a value m and note $v(m)$ the average numbers of tests performed in lines 8 and 17. We have:

- $v(M) = 0$, $v(M-1) = v(M+1) = 1$,
- when $m < M$, $v(m) = 1 + \frac{v(m+1)}{\mathcal{R}F(m)}$
- when $m > M$, $v(m) = 1 + \mathcal{R}F(m-1)v(m-1)$.

For the first three problems, we can relate these values to the values of $s(i)$ found for a binomial distribution $B(m)$ with $maxM = 2M$ in the Appendix A. Indeed, we can reuse the results of [Alo23]: we have when $m < M-1$, $\mathcal{R}B(m) \leq \mathcal{R}F(m)$ and when $m > M+1$, $\mathcal{R}B(m) \geq \mathcal{R}F(m)$, which implies that for all m, we have $v(m) \leq s(m)$ and that the average time spent in a loop is $O(\log(2M)) = O(\log n)$.

Concerning the Partial Injection problem, we can note that when $m \leq n_0 = \lfloor n + 1 - \sqrt{2n+3} \rfloor$, $v(m) \leq 1 + \frac{v(m+1)}{2}$. So, we have when $m \leq n_0$, $v(m) \leq 2 + \frac{1}{2^{n_0+1-m}} v(n_0+1)$,

$$\sum_{m=0}^{n_0} v(m) \leq 2(n_0+1) + v(n_0+1) \leq 2n+4.$$

When $m > n_0$, we can use a rough majoration $v(m) \leq \sqrt{2n+3}$, so we get $\sum_{m=0}^{n} v(m) = O(n)$, which gives us an average complexity of $O(1)$ for the time spent in a loop. □

Remark

- Sometimes lines 8 and 17 of the algorithm can be simplified. For example, if we want to generate an m for the Motzkin tree, line 8 can be simplified to $if(random(n-2i-1) >= i+1$ OR $random(n-2i) >= i+2)then$ (this is also true for line 17), we will then find an algorithm that only works with a number less than or equal to $n+1$.

Finally, we run 1000 trials with $n = 10^3$ or $n = 10^7$ and compute for the loop and test counters: the average, minimum and maximum values.

	Loop			Test		
	ave	min	max	ave	min	max
Fib(10^3)	20.935	1	153	40.983	0	253
Motz(10^3)	25.395	1	201	39.311	0	289
Sch(10^3)	31.069	1	317	63.375	0	554
Inj(10^3)	97.2	1	657	103.295	0	689
Fib(10^7)	2216.199	3	22230	7955.89	92	76507
Motz(10^7)	2685.683	2	20823	6935.725	100	48683
Sch(10^7)	2994.821	7	25682	11086.241	216	82911
Inj(10^7)	98148.946	36	619533	98295.043	60	620425

5 Algorithm for distribution with trailing tail

We have seen in the previous section some direct uses of Algorithm 2 that are efficient. However, if we study other problems: non-equiprobable binomial distribution with $p = 1$ and some generic q, ..., the use of the Algorithm 2 will always give an efficient solution but we can get better results by noting that a long tail phenomenon appears: there exists a value $trailM > M$ such that when $k \geq trailM$, $\mathcal{R}F(k) \leq \frac{1}{2}$. This implies that $\sum_{m=trailM+1}^{maxM} F(m) \leq F(trailM)$, which suggests that instead of drawing a random integer between 0 and $maxM$, it would be more efficient to draw a random integer between 0 and $trailM + 1$ and update our algorithm a bit.

To simplify the presentation, we will only propose a solution in this specific case, but it is easy to adapt this solution to other problems when downstream and/or upstream tails appear or when we have $\mathcal{R}F(k) \leq \frac{2}{3}$ when $k \geq trailM$ for some value of $trailM > M$, ...

So let us define a simple "probability" function U as follows:

- $U(m) = 1$ when $m \leq trailM$,
- $U(trailM + i) = \frac{1}{2^i}$ when $1 \leq i \leq maxM - trailM$,

we have $\sum_{m=0}^{maxM} U(m) \leq trailM + 2$ and we can draw a number m with probability $\frac{U(m)}{\sum_{i=0}^{maxM} U(i)}$ by drawing a random integer between 0 and $trailM + 1$ with a uniform probability then if we find $m \leq trailM$, we return m otherwise we put $m = trailM + 1$ and we increase m as long as $random(2) > 0$. Finally, if we find a value $m > maxM$, we reject this value and try again.

To reflect this change, we use a new function to accept this change $\mathcal{R}G$ with $G(m) = \frac{U(M)F(m)}{U(m)}$, we have $G(m) \geq F(m)$ (as $U(M) = 1$ and $U(m) \leq 1$) and :

- when $m < trailM$, $\mathcal{R}G(m) = \mathcal{R}F(m)$,
- when $m \geq trailM$, $\mathcal{R}G(m) = 2\mathcal{R}F(m)$.

Therefore, we get $\mathcal{R}G(m) > 1$ iff $m < M$ and we can use the following algorithm:

Algorithm 2 Modified algorithm for distribution with trailing trail: $\mathcal{R}F(k) \leq \frac{1}{2}$ when $k \geq trailM$

```
1: function CHOOSE(M, maxM, trailM, RF)
2:     while true do
3:         m ← random(trailM + 2)
4:         if m = trailM + 1 then
5:             while m ≤ maxM AND random(2) > 0 do
6:                 m ← m + 1
7:             end while
8:         end if
9:         ok ← True
10:        if m ≤ M then
11:            i ← m
12:            while ok AND i < M do
13:                if random(numer(RF(i))) ≥ denom(RF(i)) then
14:                    ok ← False
15:                    break
16:                end if
17:                i ← i + 1
18:            end while
19:        else
20:            if m > maxM then continue end if
21:            i ← m − 1
22:            while ok AND i ≥ M do
23:                a ← 1 + [i ≥ trailM]
24:                if random(denom(RF(i))) ≥ a numer(RF(i)) then
25:                    ok ← False
26:                    break
27:                end if
28:                i ← i − 1
29:            end while
30:        end if
31:        if ok then returns m end if
32:    end while
33: end function
```

We can etablish some generic upper bounds on the average complexity:

Theorem 2 *If we assume that the tests of lines 13 and 24 can be performed with complexity $O(1)$, if $\mathcal{R}F(m) > 1$ iff $m < M$ and if $\mathcal{R}F(m) \leq \frac{1}{2}$ when $m \geq trailM$:*

- *the average complexity of the algorithm is $O(1 + trailM^2)$,*
- *if $\mathcal{R}F(m)$ is a decreasing function, the average complexity of the algorithm is $O(1 + trailM)$,*

which are proved in the Appendix C.

6 A rejection Algorithm for a non-equiprobable binomial distribution with $p = 1$

Let us choose values q, α with $q \geq 1$ and $0 \leq \alpha \leq q - 1$ and let us note $F_M(m) = \left(\begin{array}{c} (q+1)\,M + \alpha \\ m \end{array} \right) q^{(q+1)\,M+\alpha-m}$, we have $\sum_{m=0}^{(q+1)\,M+\alpha} F_M(m) = (q+1)^{(q+1)M+\alpha}$, $\mathcal{R}F_M(m) = \frac{F_M(m+1)}{F_M(m)} = \frac{(q+1)M+\alpha-m}{q(m+1)}$. $F_M(m)$ is unimodal, it reaches a maximum value when $m = M$, $\mathcal{R}F(m)$ is decreasing.

Note first that when $m \geq 2M + 1$,

$$\begin{array}{rl} & 2((q+1)M + \alpha - m) - q(m+1) \\ \leq & 2((q+1)M + \alpha - (2M+1)) - q(2M+1+1) = -2(q - \alpha + 1) - 2M < 0. \end{array}$$

and thus $\mathcal{R}F_M(m) \leq \frac{1}{2}$.

Therefore, we can call Algorithm 2 with M, $trailM = 2M + 1$, $maxM = (q+1)M + \alpha$, $\mathcal{R}F = \mathcal{R}F_M$ to obtain an algorithm that draws a random number with the following probability: $\frac{F_M(m)}{(q+1)^{(q+1)\,M+\alpha}}$ and which uses only small integers (less than $(q+1)\,M + \alpha$). Theorem 2 gives us an upper bound on the average complexity in $O(M+1)$ (a bound that does not depend on the values q and α); more precisely fact, we have :

Proposition 3 *Its average complexity is $O(1+\sqrt{M}\,\log(1+M))$ and its average complexity to draw a number m is $O(1 + \sqrt{M}\,\log(1+M) + |m - M|)$. These bounds do not depend of the values q and α.*

which is proved in Appendix D

Finally, we run 1000 trials with $M = 10^5$ and different values of q and α:

	Loop			Test		
	ave	min	max	ave	min	max
$q = 1, \alpha = 0$	349.95	1	2075	1346.422	27	8192
$q = 10^3, \alpha = 0$	240.544	1	1410	1676.422	24	9702
$q = 10^6, \alpha = 0$	253.246	1	1533	1737.546	21	9611
$q = 10^9, \alpha = 0$	230.069	1	1647	1631.123	42	10415
$q = 10^3, \alpha = q - 1$	249.527	1	1746	1720.894	15	10540
$q = 10^6, \alpha = q - 1$	243.489	1	1587	1686.923	27	9735
$q = 10^9, \alpha = q - 1$	254.332	1	2176	1759.796	5	12404

7 Application to some problems with a trailing tail

As in Section 4, we want to study if we can reuse the same method to draw an integer m with probability $\frac{F(m)}{\sum_i F(i)}$ for some other structures.

Generating a Motzkin left factor of size n and final height h
If we examine the algorithms presented in [Alo23], we find that we can improve the average complexity of the first step of generating a Motzkin left factor of final height h $\left(\text{when } h < n - \frac{3}{2} - \frac{\sqrt{8n+9}}{2}\right)$ by using the previous algorithm.

However, we can do better, if $F_n^h(m)$ is the number of Motzkin left factors of size n and final height h with m steps $(1, 1)$, we have $\mathcal{R}F(m) = \frac{(n-h-2m)(n-h-2m-1)}{(m+1)(m+2+h)}$; we can use Algorithm 2 with $M = \lceil \tilde{m} \rceil$ $\left(\text{with } \tilde{m} = \frac{2n}{3} - \frac{h}{2} + \frac{1}{6} - \frac{\sqrt{(2n+5)^2 - 3h(h+2)}}{6}\right)$, $trailM = 2M + 1$ and $maxM = \left\lfloor \frac{n-h}{2} \right\rfloor$.

Proposition 4 *Calling* CHOOSE *with these arguments gives a valid algorithm to draw a number m with probability $\frac{F_n^h(m)}{\sum_i F_n^h(i)}$ with average complexity in $O(1 + \sqrt{M}\, log(M + 1))$ (with $M \leq \frac{n-h}{3}$).*

Proof. Let us first assume that $h < n - \frac{3}{2} - \frac{\sqrt{8n+9}}{2}$. If we consider the binomial distribution B_M defined by M, $q = \left\lceil \frac{n-h}{2\tilde{m}} \right\rceil - 1$ and $\alpha = max(q - 1, q - \lceil q(M - \tilde{m}) \rceil)$, we can use lemmas 7 and 8 in [Alo23] to prove that the average number of loops is $O(\sqrt{M})$ and we have $\mathcal{R}B_M(i) < \mathcal{R}F(i)$ when $i < M - 1$ and $\mathcal{R}B_M(i) > \mathcal{R}F(i)$ when $i > M$, this is enough to prove that $\mathcal{R}F(m) \leq \frac{1}{2}$ when $n \geq trailM$, that the algorithm is valid and that the average time spent in a loop is $O(\log M)$.

Now suppose that $n - \frac{3}{2} - \frac{\sqrt{8n+9}}{2} \leq h < n - \sqrt{n+2}$, which corresponds to the case $M = 1$. We notice first that $\mathcal{R}F(m)$ is a decreasing function in h and so $\mathcal{R}F(3) \leq \frac{2n+27-5\sqrt{8n+9}}{4n+14-2\sqrt{8n+9}} \leq \frac{1}{2}$ when $n \geq 2$. This proves that the algorithm is

valid (because when $n < 2$, $F(3)$ is not defined). It also allows to prove that the average complexity passed in a loop is $O(1)$ by imitating the proof of the Proposition 3 and that the average number of loops is $O(1)$ because $m = M$ is drawn with probability $\frac{1}{M+3} = \frac{1}{4}$ and accepted with probability 1.

Finally, suppose that $n - \sqrt{n+2} \leq h$, this corresponds to the case $M = 0$. We have $\mathcal{R}F(1) \leq \frac{n+8-5\sqrt{n+2}}{2n+6-2\sqrt{n+2}} \leq \frac{1}{2}$ which is enough to prove that the algorithm is valid and that its complexity is $O(1)$. □

Finally, we run 1000 trials with $n = 10^5$ and different values of h:

	Loop			Test		
	ave	min	max	ave	min	max
$h = 0$	355.675	1	2567	552.752	13	3812
$h = 10^4$	305.894	1	2102	528.14	2	3396
$h = 2\,10^4$	264.341	1	1857	501.912	2	3168
$h = 5\,10^4$	154.633	1	1129	396.291	6	2523
$h = 8\,10^4$	55.144	1	365	181.797	1	1143
$h = 9\,10^4$	26.436	1	338	92.282	0	1008
$h = 10^5$	2.927	1	19	0	0	0

Partial Injection(revisited)

In [GBN10], Gouyou-Beauchamps et al. propose to start choosing the number k of elements that appear in a partial injection. It seems more promising to generate the number $m = n - k$ of elements that do not appear in a partial injection.

Indeed, let $F_n(m) = \frac{n!^2}{(n-m)!m!^2}$ the number of partial injection of size n which contains $n - m$ elements, we have $\mathcal{R}F(i) = \frac{(n-i)}{(i+1)^2}$, $M = \lceil \tilde{m} \rceil$ (with $\tilde{m} = -\frac{3}{2} + \frac{\sqrt{5+4n}}{2}$), $tailN = 2M + 1$ and $maxM = n$.

We have:

Proposition 5 *Calling* CHOOSE *with these arguments gives a valid algorithm to draw a number m with probability $\frac{F_n(m)}{\sum_i F_n(i)}$ with average complexity in $O(n^{1/4} \log n)$.*

which is proved in Appendix E

Finally, we run 1000 trials with different values of n :

n	Loop			Test		
	ave	min	max	ave	min	max
10	2.447	1	12	2.462	0	15
10^3	6.833	1	43	12.117	0	68
10^5	20.161	1	218	49.338	0	499
10^7	64.305	1	525	196.472	4	1487
10^9	206.651	1	1576	754.928	10	5320

8 Discussion

In this paper, we studied some algorithms that choose a number m with probability $\frac{F(m)}{\sum_{i=0}^{maxM} F(i)}$ where F has the following properties:

Property 1

- *F is a unimodal function,*
- *$\mathcal{R}F$ is a decreasing function.*

and we proposed a basic algorithm 1 which is valid, very simple, uses only basic parameters (the value of M which maximizes $F(m)$, $maxM$ and the function $\mathcal{R}F$) and guarantees an average complexity in $O(maxM + 1)$. We have shown that for many $F(m)$ distributions, this average complexity can be much lower (but this requires writing proofs that can be boring).

We also noticed that for some distributions, $F(m)$ has a very long tail (i.e., there is a value $trailM$ such that when $m \geq trailM$, $\mathcal{R}F(m) \leq \frac{1}{2}$). In this case, we proposed a trick to improve the basic upper bound on the average complexity to $O(trailM)$[5].

Comparison with [GBN10] and [Alo23] Gouyou-Beauchamps et al. and Alonso prove that at least in the generation of certain structures, one can obtain some efficient algorithm by first drawing a number m with probabilities and then using this number to generate an element of that structure. They propose to first generate a number m with some binomial distribution "closed" to the desired distribution, and then accept or reject it using an efficient algorithm. We try here to question this assumption, i.e. to study what happens if we start with a distribution very far from the desired one: the basic uniform distribution?

We prove that for the proposed problems, instead of starting with an equiprobable binomial distribution, using a uniform distribution yields simpler algorithms that have much lower complexity on average. We also show that we can extend this method to generate a left Motzkin factor of size n and final height h and partial injection of size n using a small trick to generate the value of m differently, which in the long run keeps the average complexity low.

This suggests that we can choose many different distributions (close or not to the desired distribution) and potentially obtain efficient algorithms using the generic framework proposed by [GBN10].

Open problems In this paper, we have only proposed an algorithm to generate a non-equiprobable binomial distribution with $p = 1$. Can we find an efficient algorithm in the general case?

We can also notice that we have studied algorithms that need $\max(0, |m - M| - 1)$ final tests to accept a value of m, so we can only expect to obtain a

[5] Of course, it is possible to adapt it to a function with a long head tail and a potential long tail. It is also possible to modify a little the definition of the tail.

final complexity in $\Omega\left(E(|m-M|)\right)$ (where E is the expected value). But, in all cases, we obtained a final complexity of $O\left(E(|m-M|)\log(E(|m-M|)\right)$, can we find simple algorithms with better average complexity?

Finally, we have studied here the problem of finding an integer value of m such that $\frac{F(m)}{\sum_{i=0}^{maxM} F(i)}$, can we use similar methods to find integer values m, n such that $\frac{F(m,n)}{\sum_{i,j} F(i,j)}$? If so, is it better to start with a multinomial distribution, two 'binomial' distributions as in Gouyou-Beauchamps et al.[GBN10] and Alonso[Alo23], two uniform distributions (as in this article) or a mixture of these two methods?

References

[Alo94] Laurent Alonso. Uniform generation of a motzkin word. *Theoretical Computer Science*, 134(2):529–536, 1994.

[Alo23] Laurent Alonso. Uniform random generations and rejection method(i) with binomial majorant, 2023.

[GBN10] Dominique Gouyou-Beauchamps and Cyril Nicaud. Random generation using binomial approximations. *Discrete Mathematics & Theoretical Computer Science*, pages 359–372, 2010.

[GKP88] Ronald L. Graham, Donald Ervin Knuth, and Oren Patashnik. Concrete mathematics. 1988.

[Hö93] Wolfgang Hörmann. The generation of binomial random variates. *Journal of Statistical Computation and Simulation*, 46(1-2):101–110, 1993.

A Appendix: proof of Theorem 1, generic upper bound.

Note when a value m is drawn, $s(m)$: the average number of tests in lines 8 and 17 of the Algorithm 1 performed within the execution of the main loop.

We have:

- $s(M) = 0$,
- when $m < M$, $s(m) = 1 + \frac{s(m+1)}{\mathcal{R}F(m)}$,
- when $m > M$, $s(m) = 1 + \mathcal{R}F(m-1)s(m-1)$.

The average number of tests performed in a loop is equal to $\frac{\sum_i s(i)}{maxM+1}$. A value is accepted in the main loop with the probability $\sum_i \frac{1}{maxM+1}\frac{F(i)}{F(M)} = \frac{1}{maxM+1}\frac{T}{F(M)}$ (with $T = \sum_i F(i)$), so the average number of loops is $\frac{(maxM+1)F(M)}{T}$ and the average complexity of the algorithm is $((maxM+1) + \sum_i s(i))\frac{F(M)}{T}$.

We thus obtain a coarse majoration:

Lemma 1 *The average complexity of the algorithm is* $O\left((1+maxM)^2\frac{F(M)}{T}\right) = O(1+maxM^2)$

noting that $s(i) = O(|M-i|) = O(maxM)$ and $\frac{F(M)}{T} = O(1)$.

Remark: If $T = \Omega((1+maxM)\,F(M))$ (which can happen if the function F varies slowly), we get an average complexity of $O(1+maxM)$.

When $\mathcal{R}F$ is a decreasing function and $M = maxM$

We assume in this subsection that $\mathcal{R}F$ is a decreasing function and F an increasing function.

Let $T^k = \sum_{m=0}^{k} F(m)$, we obtain:

Lemma 2 *When* $k \geq 1$, $F(k)T^{k-1} \leq F(k-1)T^k$.

Proof. Indeed, we have when $i \leq k-1$, $F(i)F(k) \leq F(i+1)F(k-1)$ as $\mathcal{R}F(k-1) \leq \mathcal{R}F(i)$. Therefore:

$$F(k)T^{k-1} = \sum_{i=0}^{k-1} F(i)F(k) \leq \sum_{i=0}^{k-1} F(i+1)F(k-1) = F(k-1)\sum_{i=1}^{k} F(i) \leq F(k-1)T^k.$$

□

We can use a recurrence to obtain a majoration, let's define:

- when $m \geq k$, $s^k(m) = 0$,
- when $m < k$, $s^k(m) = 1 + \frac{s^k(m+1)}{\mathcal{R}F(m)} = \sum_{i=m}^{k-1}\frac{F(m)}{F(i)} = s^{k-1}(m) + \frac{F(m)}{F(k-1)}$,

- and $S^k = \sum_{i=0}^{k} s^k(i)$,

when $k = 0$, we have $T^0 = F(0)$, $s^0(m) = 0$ for all m, while we find T and $s(m)$ by taking $k = M$.

We can now write $C^k = \frac{S^k F(k)}{T^k}$ which gives us $C^0 = 0$ while C^{maxM} is the average number of tests performed in all the loops.

Proposition 6 *We have $C^{maxM} \leq maxM$.*

Proof. We will prove by recurrence that $C^k \leq k$, this formula is true when $k = 0$. Let us suppose that it is true for $k = K - 1$.

We have:

$$S^k = \sum_{m=0}^{k} s^k(m) = \sum_{m=0}^{k-1} \left(s^{k-1}(m) + \frac{F(m)}{F(k-1)} \right) = S^{k-1} + \frac{T^{k-1}}{F(k-1)}.$$

Therefore:

$$\begin{aligned} C^K &= \frac{S^{K-1}F(K) + T^{K-1}\frac{F(K)}{F(K-1)}}{T^K} = \frac{F(K)T^{K-1}}{F(K-1)T^K} \left(\frac{S^{K-1}F(K-1)}{T^{K-1}} + 1 \right) \\ &\leq 1(K-1+1) = K \end{aligned}$$

by using Lemma 2. □

When $\mathcal{R}F$ is a decreasing function

We can now relax the condition that $M = maxM$.

Indeed, we have by using the Proposition 6:

$$\sum_{m=0}^{M} s(m)\frac{F(M)}{T} \leq \sum_{m=0}^{M} s(m)\frac{F(M)}{T^M} \leq M,$$

similarly, changing the variable m to $maxM - m$, we obtain:

$$\sum_{m=M}^{maxM} s(m)\frac{F(M)}{T} \leq (maxM - M).$$

Therefore, the average number of tests is $O(M + maxM - M) = O(maxM)$ while the average number of loops is $O(\frac{(maxM+1)F(M)}{T}) = O(maxM + 1)$. This ends the proof.

B Appendix: proof of Proposition 1, binomial distribution

We only need to prove that the average complexity is $O(\sqrt{N}\log N)$ when $N \geq 1$ because the last statement is a consequence of the fact that we need $O(|m-N|)$ tests to accept a value m.

First notice that in the main loop, an initial value of m (drawn with probability $\frac{1}{N+1}$) is accepted with probability $\frac{F(m)}{F(M)}$. Thus, in this loop, we accept a value m with a total probability $\frac{1}{N+1}\sum_{m=0}^{N}\frac{F(m)}{F(M)} = \frac{2^N}{(N+1)F(M)} = \Theta\left(\frac{1}{\sqrt{N}}\right)$ when $N \geq 1$. Therefore, we can expect to run the main loop $O(\sqrt{N})$ times.

Now suppose we have drawn an integer m (drawn with probability $\frac{1}{N+1}$) in line 3 and define $s(m)$ the average number of comparisons made in line 8 or in line 17 as in the proof of the Theorem 1, we have :

- $s(M) = 0$,
- when $m < M$, $s(m) = 1 + \frac{m+1}{N-m}s(m+1)$,
- when $m > M$, $s(m) = 1 + \frac{N-m+1}{m}s(m-1)$.

So the average complexity spent in one step of the loop is $O\left(\sum_{m=0}^{N}\frac{s(m)}{N+1}\right) + O(1)$.

Note first that $f(m) = \frac{m+1}{N-m}$ is an increasing function in m. Now note $m_k = \left\lfloor \frac{2^k-1}{2^{k+1}}N - 1 \right\rfloor$ when $k \geq 0$. We have when $m \leq m_k$:

$$f(m) \leq f(m_k) \leq f\left(\frac{2^k-1}{2^{k+1}}N - 1\right) = \frac{\frac{2^k-1}{2^{k+1}}N}{\frac{2^k+1}{2^{k+1}}N+1} \leq \frac{2^k-1}{2^k+1} \leq 1 - \frac{1}{2^k}.$$

This gives us when $m \leq m_k < M$, $s(m) \leq 1 + (1-\frac{1}{2^k})s(m+1)$ and so $s(m_k - i) \leq 2^k + (1-\frac{1}{2^k})^{i+1}s(m_k+1)$ when $0 \leq i \leq m_k$.

So when $k \geq 1$, we get :

$$\begin{aligned}
\sum_{i=m_{k-1}+1}^{m_k} s(i) &\leq 2^k(m_k - m_{k-1}) + 2^k s(m_k+1) \\
&\leq 2^k(m_k - m_{k-1}) + 2^k(N/2 - m_k) \leq 2^k(N/2 - m_{k-1}) \\
&\leq 2^k\left(N/2 - \left(\frac{2^k-1}{2^{k+1}}N - 1 - 1\right)\right) \leq N + 2^{k+1}
\end{aligned}$$

using a coarse majoration of $s(m_k+1) \leq N/2 - (m_k+1)$.

So when $N \geq 2$, we get:

$$\frac{1}{N+1}\sum_{i=0}^{M} s(i) = \frac{1}{N+1}\left(\sum_{k=1}^{\lfloor log_2(M)\rfloor}\sum_{i=m_{k-1}+1}^{m_k} s(i) + O(1)\right) = O(\log N).$$

When $i \geq M$, we can use the fact that the relations are symmetric by performing a change of variable i in $N - i$, which gives:

$$\frac{1}{N+1}\sum_{i=0}^{N} s(i) = \frac{2}{N+1}\sum_{i=0}^{\lfloor N/2 \rfloor} s(i) = O(\log N)$$

which gives the average complexity of one step of the loop.

C Appendix: proof of Theorem 2, generic upper bound for distributions with a trail.

Lemma 3 *The average number of executions of the loops is* $\frac{(trailM+2)F(M)}{T} = O(trailM + 1)$.

Proof. A value is accepted in the main loop with probability $\sum_{i=0}^{trailM} \frac{1}{trailM+2}\frac{G(i)}{G(M)} + \sum_{i=trailM+1}^{maxM} \frac{1}{trailM+2}\frac{1}{2^{i-trailM}}\frac{G(i)}{G(M)} = \sum_i \frac{1}{trailM+2}\frac{F(i)}{F(M)} = \frac{1}{trailM+2}\frac{T}{F(M)}$, so the average number of loops is $\frac{(trailM+2)F(M)}{T} \leq trailM + 2$. □

Now let's note $s(m)$ the average number of tests performed in a loop when a value m is chosen, we have :

- $s(M) = 0$,
- when $m < M$, $s(m) = 1 + \frac{s(m+1)}{\mathcal{R}F(m)}$,
- when $M < m \leq trailM$, $s(m) = 1 + \mathcal{R}F(m-1)s(m-1)$,
- when $trailM < m$, $s(m) = 1 + 2\mathcal{R}F(m-1)s(m-1)$,

the average number of tests performed within a loop is equal to:

$$\frac{\sum_{i=0}^{trailM} s(i) + \sum_{i=1}^{maxM-trailM} \frac{s(trailM+i)}{2^i}}{trailM + 2}$$

Note first that:

$$\sum_{i=1}^{maxM-trailM} \frac{s(trailM+i)}{2^i} \leq \sum_{i=1}^{maxM-trailM} \frac{trailM+i}{2^i} \leq trailM + 2,$$

while

$$\sum_{i=0}^{trailM} s(i) \leq \sum_{i=0}^{trailM} trailM \leq (trailM+1)trailM.$$

Thus the average of the number of tests inside a loop is $O(trailM + 1)$ which allows to prove the theorem in the generic case.

We can now assume that $\mathcal{R}F(m)$ is a decreasing function. Let us first consider a function F' such that $F'(m) = F(m)$ when $m \leq trailM$ and $F'(m) = 0$ when $m > trailM$, using the proof of the Theorem 1, we know that the total number of tests is $\frac{S'^{trailM} F(M)}{T'} \leq trailM$ (with $T' = \sum_i F(i)$). We thus obtain for the average number of tests performed in all loops:

$$\begin{aligned}\frac{S^{maxM} F(M)}{T} &\leq \frac{(S'^{trailM} + trailM + 2) F(M)}{T'} \\ &= \frac{S'^{trailM} F(M)}{T'} + \frac{(trailM + 2) F(M)}{T'} = O(trailM + 1).\end{aligned}$$

which ends the proof since the average number of loops is also $O(trailM + 1)$.

D Appendix: proof of Proposition 3, non equiprobale binomial distribution with $p = 1$

The average number of execution of the loops is $\frac{(2M+3)F(M)}{\sum_m F(m)} = O(1 + \sqrt{M})$. Indeed, when $M \geq 1$, we can use the Lemma 3 of [Alo23] and when $M = 0$ (and $\alpha > 0$), we have $\sum_m F_M(m) \geq F_M(0)$ and thus $\frac{(2M+3)F_M(M)}{\sum_m F_M(m)} = O(3) = O(1)$.

Finally, we need to prove that the average time spent inside a main loop execution is $O(1 + \log(1 + M))$. Let us denote $s(m)$ the average number of comparisons performed at line 13 or line 24.

The average complexity spent in one loop execution will be $O\left(\sum_{m=0}^{2M} \frac{s(m)}{2M+3} + \sum_{m=2M+1}^{(q+1)M+\alpha} \frac{1}{2^{m-2M-1}} \frac{s(m)}{2M+3}\right) + O(1)$ with:

- $s(M) = 0$,
- when $m < M$, $s(m) = 1 + \frac{q(m+1)}{(q+1)M+\alpha-m} s(m+1)$,
- when $M < m \leq 2M + 1$, $s(m) = 1 + \frac{(q+1)M+\alpha-m+1}{q\,m} s(m-1)$,
- when $2M + 1 < m$, $s(m) = 1 + 2\frac{(q+1)M+\alpha-m+1}{q\,m} s(m-1)$,

Note first that when $m > 2M + 1$, we can reuse the tail majoration in proof of Theorem 2 to get:

$$\sum_{m=2M+2}^{(q+1)M+\alpha} \frac{1}{2^{m-2M-1}} s(m) \leq O(2M + 3) = O(M + 1).$$

When $M \leq 1$, we have $\sum_{m=0}^{2M+1} s(m) = O(1)$, so the average complexity spent in one step of the loop is $O(\frac{M+1}{M+3}) = O(1)$.

We can now assume that $M > 1$; we start by trying to obtain an upper bound for $\sum_{i=0}^{M-1} s(i)$. Note first that $f(m) = \frac{q(m+1)}{(q+1)M+\alpha-m}$ is an increasing

function in m. Let $m_h = \lfloor (1-\frac{1}{2^h})M - 1 \rfloor$ when $h \geq 0$. we have when $m \leq m_h$:

$$\begin{aligned} f(m) \leq f(m_h) &\leq f\left(\left(1-\frac{1}{2^h}\right)M - 1\right) = \frac{q\,M(2^h-1)}{(M\,q+\alpha+1)2^h + M} \\ &\leq \frac{q\,M(2^h-1)}{M\,q\,2^h} = 1 - \frac{1}{2^h}. \end{aligned}$$

We thus obtain $s(m_h - i) \leq 2^h + \left(1-\frac{1}{2^h}\right)^{i+1} s(m_h+1)$ when $0 \leq i \leq m_h$. We can now mimick the proof of Proposition 1 to obtain: $\sum_{i=0}^{M-1} s(i) = O(M\,log(M))$.

Let us now study $\sum_{i=M}^{2M+1} s(i)$. First note that $f(m) = \frac{(q+1)M+\alpha-m+1}{q\,m}$ is a decreasing function in m. Let us now note $m_0 = 2\,M+1$ and when $h \geq 1$, $m_h = \lfloor (1+\frac{1}{2^h})M + 1 \rfloor$. We have when $h \geq 1$ and $m_h \leq m \leq 2M+1$:

$$\begin{aligned} f(m) \leq f(m_h) &\leq f\left((1+\frac{1}{2^h})M + 1\right) = \frac{(M\,q+\alpha)2^h - M}{q\,(M(2^h+1)+2^h)} \\ &\leq \frac{(M\,q+q-1)2^h - M}{q\,(M(2^h+1)+2^h)} = 1 - \frac{M\,q+M+2^h}{q\,(M(2^h+1)+2^h)} \\ &\leq 1 - \frac{M\,q}{q\,(M(2^h+1)+2^h)} \leq 1 - \frac{1}{2^{h+1}}. \end{aligned}$$

Therefore, we obtain $s(m_h+i) \leq 2^{h+1} + (1-\frac{1}{2^{h+1}})^{i+1} s(m_h - 1)$ when $0 \leq i \leq 2M+1-m_h$ and $\sum_{i=M}^{2M+1} s(i) = O(M\,log(M))$ which ends the proof.

E Appendix: proof of Proposition 5, partial injection problem revisited

$\mathcal{R}F$ is a decreasing function, so when $m \geq 2M+1$, $\mathcal{R}F(m) \leq \mathcal{R}F(2M+1) \leq \mathcal{R}F(2\tilde{m}+1) \leq \frac{1}{4} \leq \frac{1}{2}$. This algorithm is therefore valid and if we note $s(m)$ the average complexity passed in a loop when m is chosen, $\sum_{m=2M+1}^{n} \frac{1}{2^{m-2M-1}} s(m) = O(M)$. We can prove that the average number of loops is in $O(n^{1/4})$ using the same arguments as in [GBN10].

So it is enough to prove that the average time spent in the loop is in $O(\log n)$. Let us consider the binomial distribution B_M defined by M, $q = \lceil \sqrt{n} \rceil + 1$, $\alpha = max(q-1, q - \lceil q(M-\tilde{m}) \rceil)$. We can now compare the complexity of this algorithm with that of the binomial distribution method. Indeed, we have $\mathcal{R}B_M(i) < \mathcal{R}F(i)$ iff $f(i) = q(n-i) - ((q+1)M + \alpha - i)(i+1) > 0$. We have $f(-1) = q(n-1) > 0$ when $n > 1$, $f(n) < 0$ and $f'(x) = 2x+1-(q+1)M-q-\alpha$, so f has a unique root $\tilde{x}$ in $[0,n]$, and we have $f(x) > 0$ if $x < \tilde{x}$ and $f(x) < 0$ if $x > \tilde{x}$.

So $f(\tilde{m}) = (q(\tilde{m}+1) - (q+1)M - \alpha + \tilde{m})(\tilde{m}+1) = (q - (q+1)(M-\tilde{m}) - \alpha)(\tilde{m}+1)$ with $q - q(M-\tilde{m}) - 1 \leq \alpha \leq q - q(M-\tilde{m})$ by definition of α. This gives us $-(\tilde{m}+1) \leq f(\tilde{m}) \leq \tilde{m}+1$.

Let $f_1(x) = f(x+1) - f(x) + (\tilde{m}+1)$, we have when $x = \tilde{m} - 1$ or $x = \tilde{m}$:

$$\begin{aligned} f_1(x) &= 2x - (q+1)M - q - a + \frac{3+\sqrt{4n+5}}{2} \\ &\leq 2\tilde{m} - (\sqrt{n}+1+1)\tilde{m} - (\sqrt{n}+1) + \frac{3+\sqrt{4n+5}}{2} \\ &= \frac{(1-\sqrt{n})\sqrt{4n+5} + \sqrt{n} + 1}{2} < 0 \end{aligned}$$

when $n \geq 3$.

Thus, when $n \geq 3$, $f(M-2) \geq f(\tilde{m}-1) = f(\tilde{m}) - f_1(\tilde{m}-1) + (\tilde{m}+1) > 0$ and $f(M+1) \leq f(\tilde{m}+1) = f(\tilde{m}) + f_1(\tilde{m}) - (\tilde{m}+1) < 0$. This suffices to prove that $s(m) \leq s_{B_M}(m)$ for all m. Therefore, we obtain an average complexity of one-step inside the loop in $O(log(M)) = O(\log n)$.