Zadanie 4 - *Cover time of a random walk*

Michał Łukomski 255552

W zadaniu celem było zbadanie czasu pokrycia grafu $G = (V, E)$ za pomocą błądzenia losowego, czas ten zdefiniowany jest jako

$$\tau_{v_0} = \min\{t \in \mathbb{N} : (\forall v \in V)(\exists t' \leq t)(X_{t'} = v)\},$$

gdzie t czas (numer kroku) błądzenia losowego, V to zbiór wierzchołków grafu, X_t to wierzchołek grafu w którym znajduje się agent w czasie t.

Zbadane grafy to:

- klika (graf pełny, *clique*),
- ścieżka (*path graph*) - agent startuje na końcu oraz na środku ścieżki,
- zupełne drzewo binarne (*complete binary tree*),
- lizak (*lollipop*).

Każdy z grafów był analizowany dla $n = 100, 150, \ldots, 2000$ węzłów, i dla każdego n wykonano $k = 1000$ (*clique*, *path*, *binary tree*) lub $k = 10$ (*lollipop*) powtórzeń wykorzystując Xoshiro256++ jako generator liczb pseudolosowych.

Grafy zostały zbudowane z drobnymi modyfikacjami za pomocą bilbioteki *Graphs.jl*.

```
begin
    using Graphs
    using BenchmarkTools
    using Random
    Xoshiro(42)
    using Plots
    using Statistics
    using Polylogarithms  : harmonic
end
```

```
random_walk (generic function with 1 method)
```

```
function random_walk(g::SimpleGraph{Int64}; init_node=1)
    # @assert !has_self_loops(g)

    visited = falses(nv(g)) |> collect
    curr_node = init_node
    visited[curr_node] = true

    for t in 1:1_000_000_000_000
        @inbounds curr_node = rand(g.fadjlist[curr_node])
        if @inbounds !visited[curr_node]
            visited[curr_node] = true

            all(visited) && return t
        end
    end
    @warn "The graph was not fully covered after 10^12 steps."
    return nothing
end
```

```
ns = 100:50:2000
```

```
ns = 100:50:2000
```

```
k = 1000
```

```
k = 1000
```

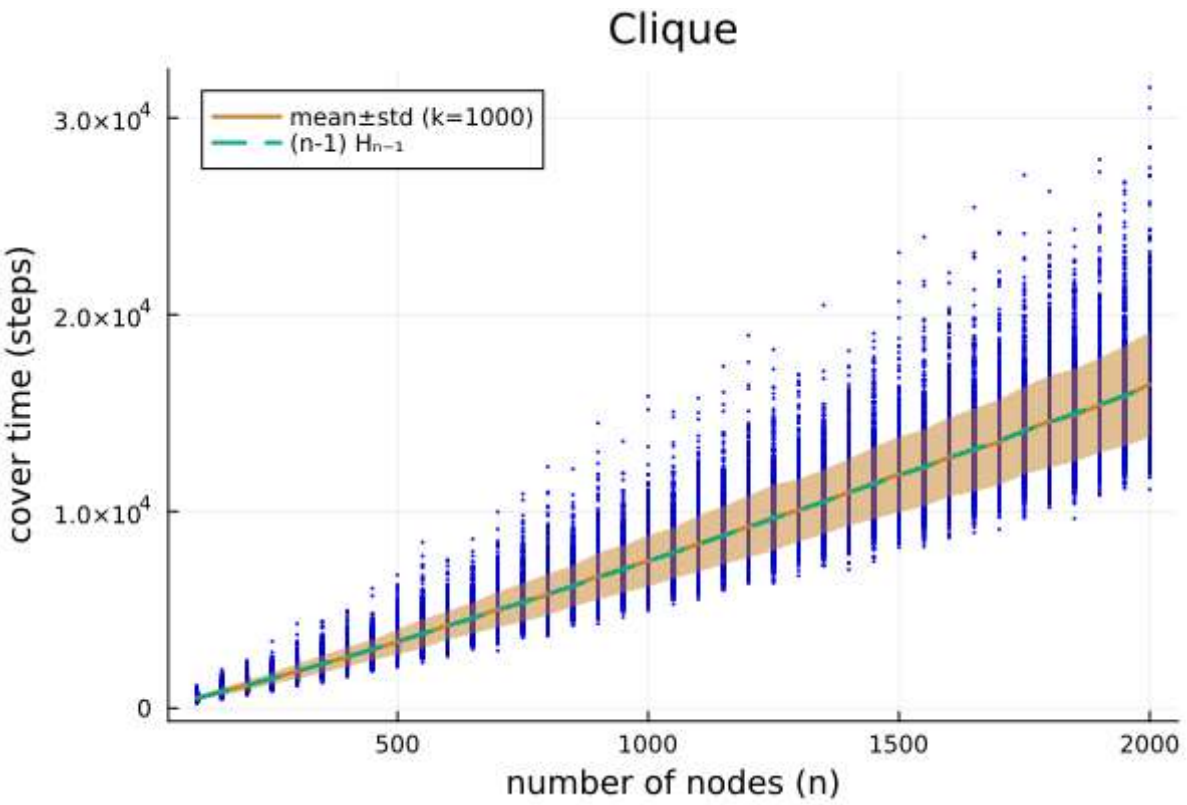
```
plot_cover_time (generic function with 2 methods)
```

```
function plot_cover_time(results, ns=ns)
    k = size(results)[1]
    X = vcat([ns for _ in 1:k]'...)

    plt = scatter(X, results, label=:none, color=:blue, marker=(1,
    stroke(0)))
    plot!(plt, ns, mean(results, dims=1)', ribbon=std(results,
    dims=1)', label="mean±std (k=$k)", linewidth=2)
    xlabel!(plt, "number of nodes (n)")
    ylabel!(plt, "cover time (steps)")
    return plt
end
```

```
experiment_random_walk (generic function with 1 method)
```

```
function experiment_random_walk(graph_constructor, k, ns;
init_node=n->one(n))
    res = zeros(Int, k, length(ns))

    Threads.@threads for j in 1:length(ns)
    # @simd for j in 1:length(ns)
        g = graph_constructor(ns[j])
        i_node = init_node(ns[j])
        @simd for i in 1:k
            @inbounds res[i,j] = random_walk(g, init_node=i_node)
        end
    end
    return res
end
```

Klika




```
begin
    res_clique = experiment_random_walk(
        Graphs.SimpleGraphs.complete_graph, k, ns)

    plot_cover_time(res_clique, ns)
    title!("Clique")

    plot!(ns, n->(n-1)*harmonic(n-1), linewidth=2,
    linestyle=:dash, label="(n-1) Hₙ₋₁")
end
```

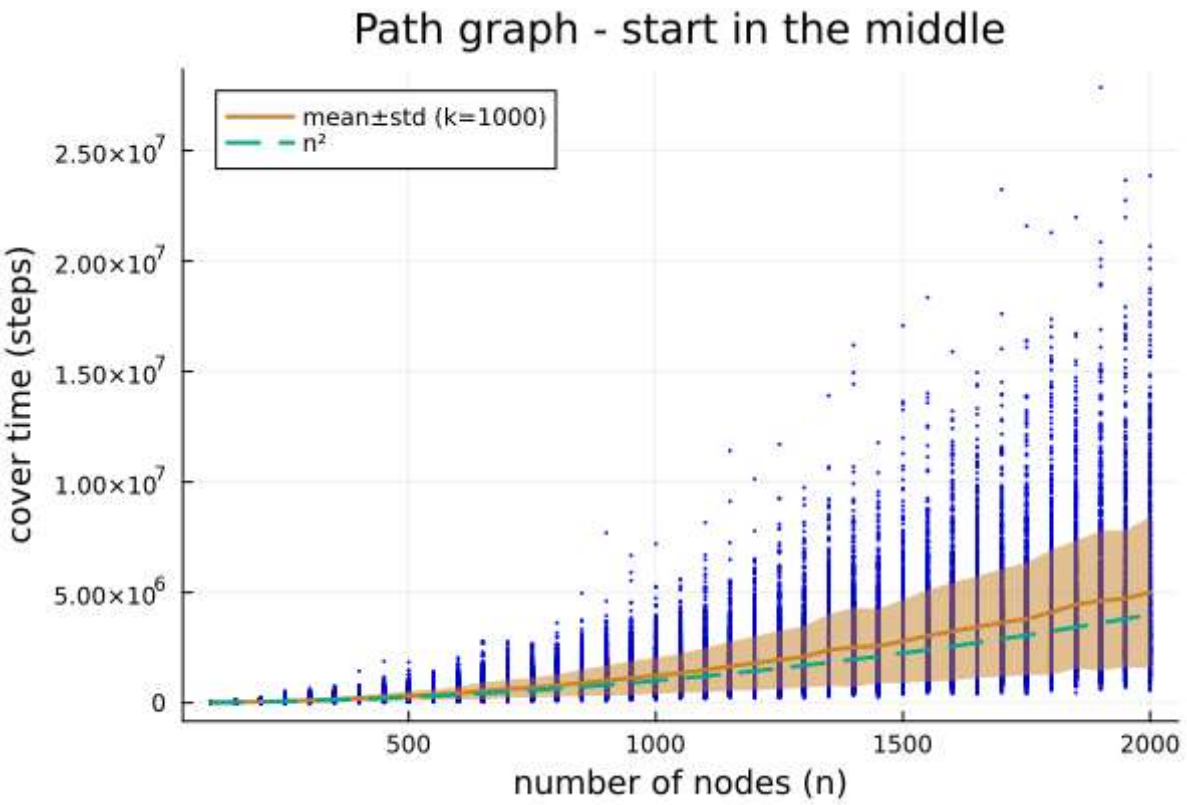
Dla grafu typu "klika" z n węzłami oczekiwany czas pokrycia wynosi

$$(n-1)H_{n-1},$$

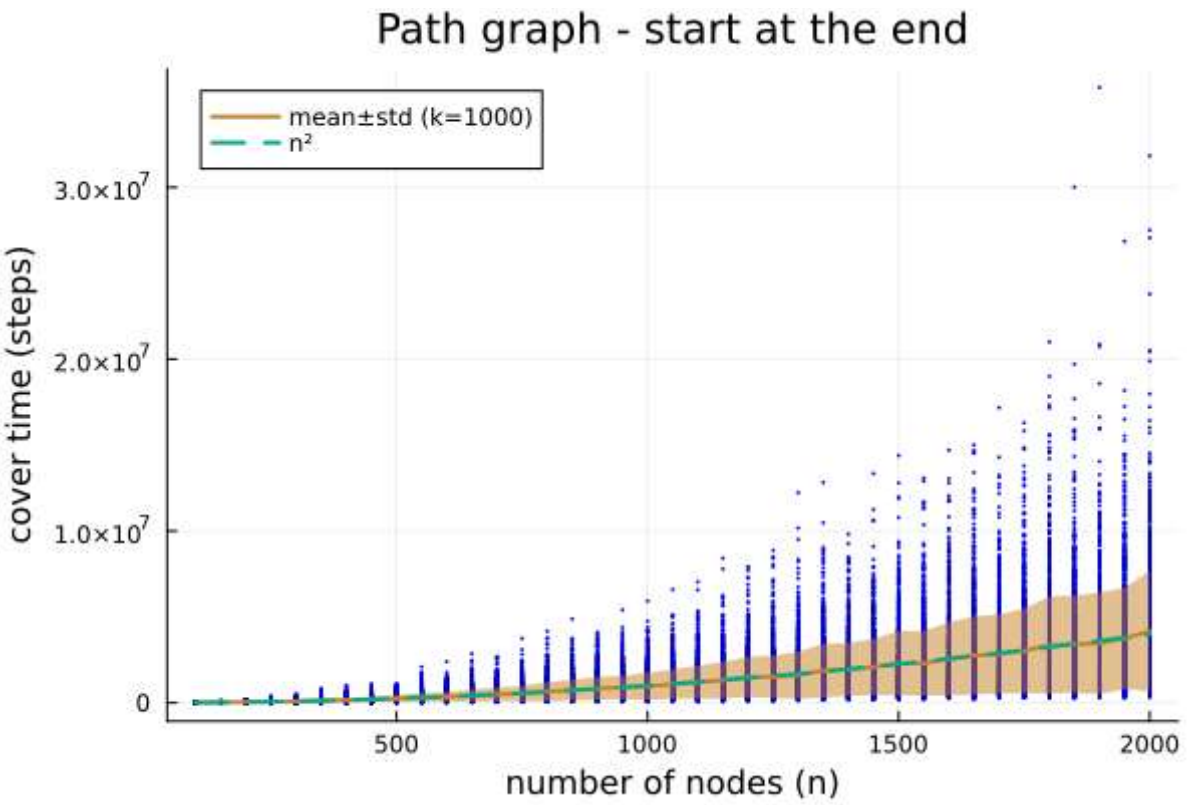
gdzie $H_n = \sum_{i=1}^{n} \frac{1}{i}$ to n-ta liczba harmoniczna.

Widać, że wraz ze wzrostem n odchylenie standardowe zwiększa się, coraz częściej pojawiają się też punkty będące daleko od średniej.

Ścieżka




```
begin
    res_path_middle = experiment_random_walk(Graphs.SimpleGraphs.path_graph, k, ns, init_node=n->n÷2)

    plot_cover_time(res_path_middle, ns)
    title!("Path graph - start in the middle")
    plot!(ns, ns.^2, linewidth=2, linestyle=:dash, label="n²")

end
```




```
begin
    res_path_end = experiment_random_walk(Graphs.SimpleGraphs.path_graph, k, ns, init_node=n->1)

    plot_cover_time(res_path_end, ns)
    title!("Path graph - start at the end")

    plot!(ns, ns.^2, linewidth=2, linestyle=:dash, label="n²")
end
```

Czas pokrycia dla grafów typu ścieżka jest rzędu n^2. Łatwo zauważyć, że ostatnim odwiedzonym węzłem, po którym zbadane są wszystkie wierzchołki grafu, musi być koniec ścieżki. Z tego względu losowy spacer zaczynający się na środku ścieżki będzie miał dłuższy czas pokrycia niż ten zaczynający się na jej końcu (co widać porównując średnie z n^2 - dla startu na końcu jest to praktycznie jedna linia, a dla startu w środku średnia ma delikatnie większy czas niż n^2), gdyż najpierw będzie musiał on dojść do jednego z końców i stamtąd pokrycie całego grafu będzie tak samo odległe, jakby dopiero co zaczął w tym punkcie. Można to sformułować jako

$$\mathbb{E}[\tau_{n/2}] = \frac{1}{2}(h_{n/2,1} + \mathbb{E}[\tau_1]) + \frac{1}{2}(h_{n/2,n} + \mathbb{E}[\tau_n]),$$

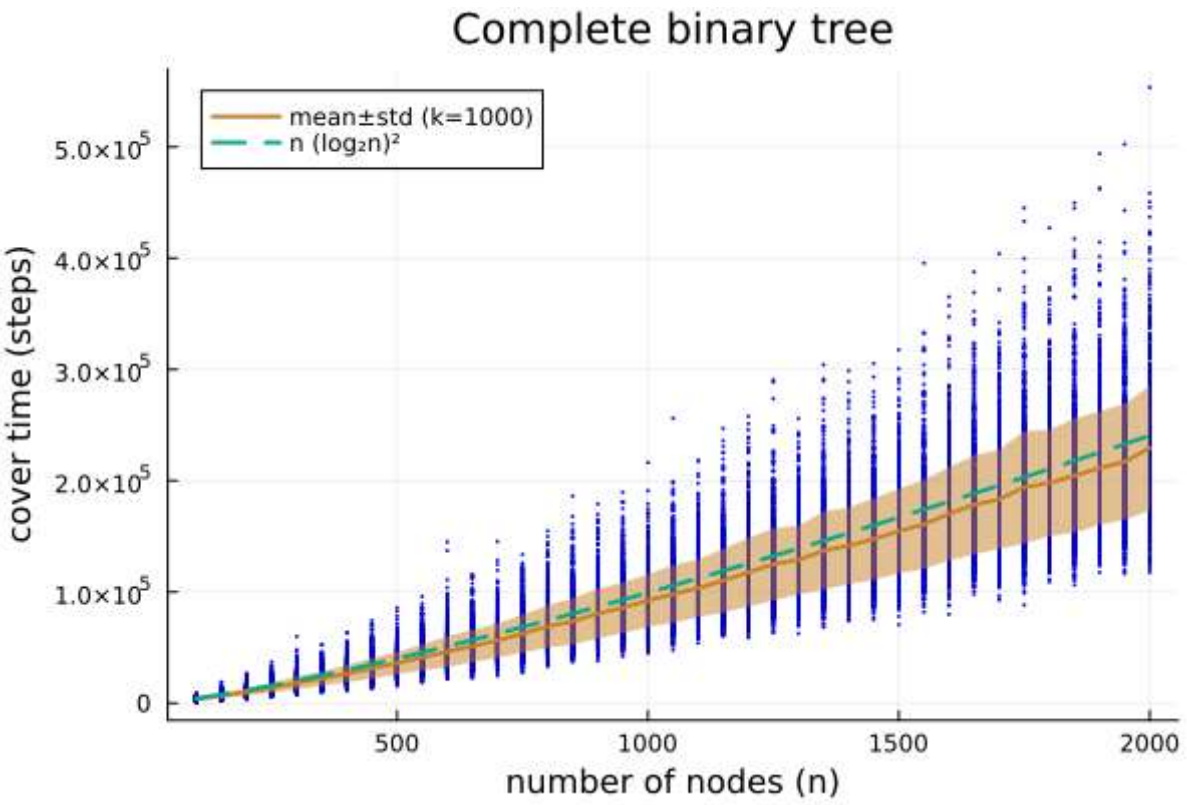
gdzie $h_{i,j}$ to oczekiwana liczba kroków do przejścia ze stanu i do j. Ze względu na symetrię (dla n parzystego będzie to przybliżenie) można zapisać

$$\mathbb{E}[\tau_{n/2}] = h_{n/2,1} + \mathbb{E}[\tau_1].$$

Zupełne drzewo binarne

```
binary_tree_graph (generic function with 1 method)
```

```
function binary_tree_graph(n)
    depth = ceil(Int,log2(n))
    g = Graphs.SimpleGraphs.binary_tree(depth)
    rem_vertices!(g, collect((n+1):(2^depth - 1)))
    return g
end
```




```
begin
    res_btree = experiment_random_walk(binary_tree_graph, k, ns)
    plot_cover_time(res_btree, ns)

    title!("Complete binary tree")

    plot!(ns, ns .* (log2.(ns)).^2, linewidth=2, linestyle=:dash,
    label="n (log₂n)²")

end
```

Oczekiwany czas pokrycia kompletnego drzewa binarnego, startując w korzeniu drzewa jest rzędu

$$n(\log_2 n)^2,$$

gdzie n to ilość węzłów w tym drzewie. Dla zapełnionego ostatniego poziomu drzewa ten czas będzie się zawierał w przedziale

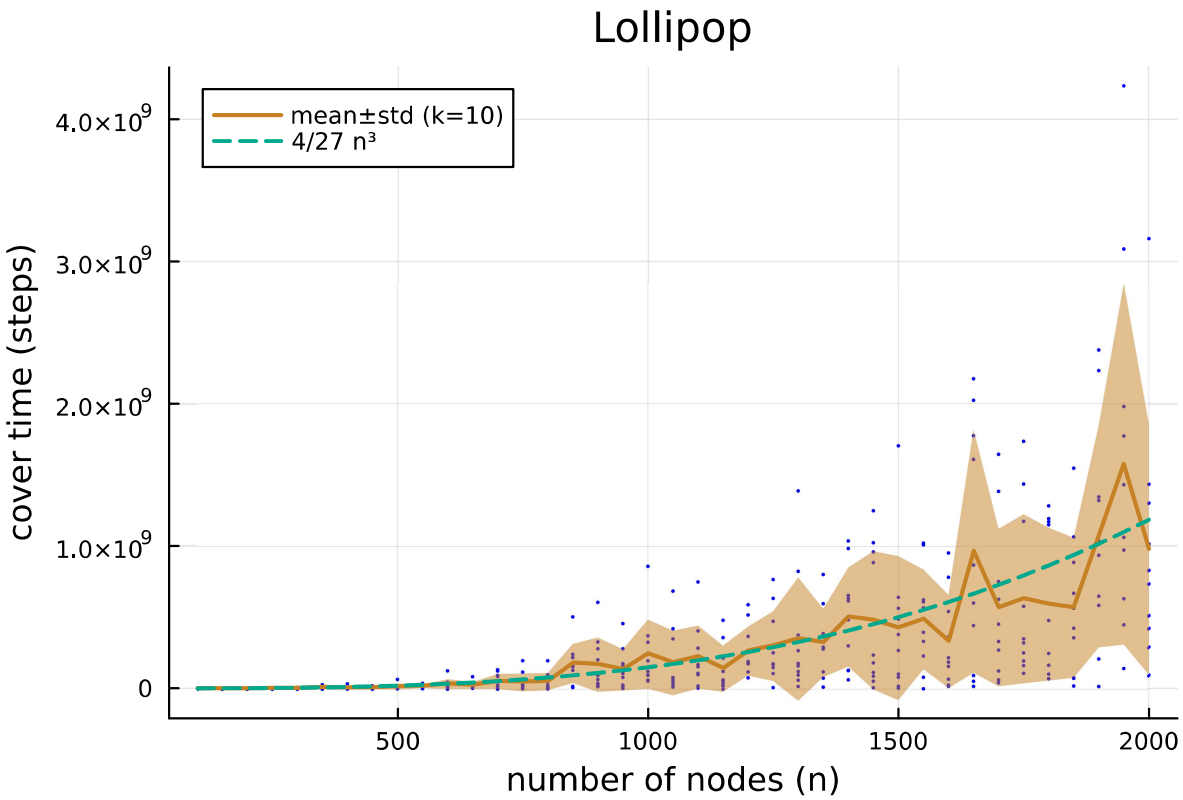
$$\frac{1}{4}(2+o(1))(\log 2)nd^2 \leq \mathbb{E}[\tau] \leq (2+o(1))(\log 2)nd^2$$

gdzie d to głębokość drzewa ($n = 2^{d+1}-1$). (Szacunki z Levin, Peres, *Markov Chains and Mixing Times*)

$(2n/3, n/3)$-Lizak

```
lollipop (generic function with 1 method)
```

```
lollipop(n) = Graphs.SimpleGraphs.lollipop_graph(
    floor(Int,2n/3),
    n - floor(Int,2n/3)
)
```




```
begin
    res_lollipop = experiment_random_walk(lollipop, 10, ns)
    plot_cover_time(res_lollipop, ns)

    title!("Lollipop")
    plot!(ns, 4/27 .* ns.^3, linewidth=2, linestyle=:dash,
    label="4/27 n³")
end
```

Graf o topologii "lizaka" z rozmiarem kliki $\lfloor \frac{2n}{3} \rfloor$ i doczepionej ścieżki $n - \lfloor \frac{2n}{3} \rfloor$ ma oczekiwany czas pokrycia rzędu

$$\frac{4}{27}n^3,$$

co jest blisko górnej granicy udowodnionej przez Feigego, która wynosi

$$\mathbb{E}[\tau] \leq \frac{4}{27}n^3 + o(n^3)$$

dla dowolnego (połączonego) grafu.

Badany tutaj przykład $(2n/3, n/3)$-lollipop osiąga najdłuższy możliwy czas pokrycia [Feige,95a] spowodowany tym, że startując z dowolnego wierzchołka szansa, że wejdziemy na ścieżkę jest niewielka, a jeszcze musi ona zostać pokonana do końca, a jeśli pozostały jakieś nieeksplorowane węzły w klice, to agent musi do niej wrócić i je odwiedzić.

Odchylenie standardowe zwiększa się

Ze względu na długi czas pokrycia dla dużych n-ów obliczenia wykonany tylko dla $k = 10$, co i tak zajęło dużo czasu. Szacuje się, że gdyby chcieć wykonać $k = 1000$ powtórzeń, tak jak przy poprzednich przykładach, to obliczenia trwałyby około 200 godzin, czyli ponad 8 dni nieprzerwanego wykorzystania wszystkich ośmiu rdzeni procesora. Aby przyspieszyć ten proces, można zaimplementować random walk na karcie graficznej (https://arxiv.org/pdf/2009.09103.pdf), wtedy 1000 powtórzeń mogłoby się wykonywać "na raz".

Podsumowanie

Widać, że badane grafy, chociaż miały taką samą liczbę węzłów, to ich czasy pokrycia były diametralnie różne. Najszybciej pokrywany był graf pełny (rzędu nH_n) - z maksymalną liczbą krawędzi jaka może wystąpić w grafie prostym. Graf o minimalnej liczbie krawędzi (tak, że nie dałoby się go podzielić na osobne grafy), czyli ścieżka miał znacznie wyższy czas pokrycia (rzędu n^2) niż graf pełny. Wydawać by się mogło, że im więcej krawędzi tym krótszy czas pokrycia, jednak nic bardziej mylnego, ponieważ można do ścieżki podowawać krawędzie tak, że graf przekształci się w lizak i jego czas pokrycia stanie się rzędu n^3. Jest to największy możliwy czas jaki może mieć wartość oczekiwana pokrycia grafu przez błądzenie losowe. Jak warto zauważyć, jest to czas wielomianowy, więc jeśli jakiś problem uda nam się przedstawić jako graf, to w wielomianowym (od liczby rozwiązań/węzłów) czasie możemy za pomocą błądzenia losowego znaleźć satysfakcjonujące nas rozwiązanie.

Na wykresach można też zaobserwować zwiększające się odchylenie standardowe w zależności od n, czyli im większy mamy graf tym większy rozrzut czasów będzie obserwowany.

W celu wykonania bardziej szczegółowej analizy, można dla większej ilości próbek aproksymować nie tylko średnią oraz odchylenie standardowe czasu pokrycia, ale jego funkcję gęstości prawdopodobieństwa. Wiadomo na pewno, że będzie ona ograniczona od dołu (np. dla grafu pełnego potrzeba przynajmniej n kroków, żeby odwiedzić wszystkie punkty), co implikuje istnienie optymalnej (być może wielu) strategii "zwiedzania" grafu. Nie będzie miała ona natomiast ograniczenia górnego, gdyż teoretycznie (szansa, że tak się stanie jest ogólnie pomijalna i w ∞ wynosi 0) agent może naprzemiennie przemieszczać się pomiędzy dwoma węzłami, więc czas pokrycia może przyjąć dowolnie wielką wartość.