Kleinberg's grid unchained

Céline Comte, Fabien Mathieu

Kleinberg's Grid Unchained

Céline Comte[a,b,1], Fabien Mathieu[a,1,*]

[a] *Nokia Bell Labs France, Nozay*
[b] *Télécom ParisTech, Université Paris-Saclay, Paris*

Abstract

One of the key features of small-world networks is the ability to route messages in a few hops, using a decentralized algorithm in which each node has a limited knowledge of the topology. In 2000, Kleinberg proposed a model based on an augmented grid that asymptotically exhibits such a property.

In this paper, we revisit the original model with the help of numerical simulations. Our approach is fueled by a new algorithm that can sample augmenting links in an almost constant time. The speed gain allows us to perform detailed numerical evaluations. We first observe that, in practice, the augmented scheme proposed by Kleinberg is more robust than what is predicted by the asymptotic behavior, even in very large finite grids. We also propose tighter bounds on the asymtotic performance of Kleinberg's greedy routing algorithm. We finally show that, if the model is fed with realistic parameters, the results are in line with real-life experiments.

Keywords: Small-World Routing, Kleinberg's Grid, Simulation, Rejection Sampling

1. Introduction

In a prescient 1929 novel called *Láncszemek* (in English, *Chains*), Karinthy imagines that any two people can be connected by a short chain of personal acquaintances, using no more than *five* intermediaries [1]. Years later, Milgram validates the concept by conducting real-life experiments. He asks volunteers to transmit a letter to a target destination across the United States through the intermediary of acquaintances [2, 3]. While not all messages arrive, successful attempts reach their destination after six hops on average, thus popularizing the notion of *six degrees of separation*.

Yet, for a long time, no theoretical model could explain why and how this kind of *small-world* routing worked. One of the first and most famous attempts to provide such a model is Kleinberg's grid [4].

1.1. Kleinberg's grid

Jon Kleinberg abstracts social networks using a grid augmented with *shortcuts*. He shows that, if the shortcuts are sampled from to a specific distribution, the trivial greedy

*Corresponding author
Email address: fabien.mathieu@nokia.com (Fabien Mathieu)
[1] The authors are members of LINCS, see http://www.lincs.fr.

routing strategy manages to reach any destination after a few hops. This seminal work has inspired multiple studies from both the theoretical and empirical social systems communities.

In [4, 5], Kleinberg considers a directed random graph $G(n, r, p, q)$, where n, p, and q are positive integers and r is a non-negative real number. A graph instance is built from a square lattice of $n \times n$ nodes endowed with the Manhattan distance d: if u and v are two nodes with respective coordinates (i, j) and (k, ℓ), the distance between u and v is $d(u, v) = |i - k| + |j - \ell|$. The Manhattan distance represents some natural proximity (geographic, social, ...) between the nodes. Each node has *local* neighbors as well as *long range* neighbors. The local neighbors of a node u are the nodes v such that $d(u, v) \leq p$. The long range neighbors of u, also called its *shortcuts*, are q nodes sampled independently and identically from a power law distribution: the probability that a given long range edge starting from u arrives in v is proportional to $(d(u, v))^{-r}$.

In a given $G(n, r, p, q)$ instance, the problem of *decentralized routing* consists in delivering a message from a node u to a node v in a hop-by-hop basis. At each step, the message bearer chooses the next hop among its neighbors. The decision can only use the lattice coordinates of the neighbors and destination. The main example of a decentralized algorithm is *greedy routing*, where the current node forwards the message to its neighbor which is the closest to the destination for the Manhattan distance d (in case of ties, an arbitrary breaking rule is used).

The main metric used to analyze the performance of a decentralized algorithm is the *expected delivery time*, defined as the expected number of hops required to transmit a message between two nodes chosen uniformly and independently at random in the graph. In [4, 5], Kleinberg proves the following theoretical results:

- for $r = 2$, the expected delivery time of the greedy algorithm is $O(\log^2 n)$;
- for $0 \leq r < 2$, the expected delivery time of any decentralized algorithm is $\Omega(n^{(2-r)/3})$;
- for $r > 2$, the expected delivery time of any decentralized algorithm is $\Omega(n^{(r-2)/(r-1)})$.

Kleinberg's results are often interpreted as follows: if $r < 2$, then the shortcuts are too long, so that they help to cross long distances during the first steps but are useless for the last steps; if $r > 2$, then the shortcuts are too short, so that they help to reduce the last steps but not to cross long distances; $r = 2$ is the only exponent that makes the shortcuts effective for both long and short distances. The fact that only one value of r asymptotically works is sometimes seen as a sign that Kleinberg's model is not robust enough to explain the efficacy of the small-world routing proposed by Karinthy and experimented by Milgram. However, as briefly discussed by Kleinberg in [6], there is in fact some margin if we consider grids of bounded size.

1.2. Our results

Our contribution is twofold. In Section 2, we propose a fast simulator to evaluate the performance of the greedy routing in Kleinberg's grid. Then, in Section 3, we use this simulator to conduct three studies. The two sections are relatively independent, so that a reader mainly interested in the applications may skip Section 2. We now detail our contributions.

In Section 2.1, we evaluate the complexity of shortcut sampling, the main bottleneck for performing fast simulations. Section 2.2 introduces the dynamic rejection sampling method, which we use to reduce the complexity of shortcut sampling. With this simplification, the only remaining complexity bottleneck is sampling the shortcut radii from a static distribution. We compare several approaches in Section 2.3. Using the fastest one, we can simulate an elementary step of the greedy algorithm in $\tilde{O}(1)$[2]. All these techniques have been implemented in a Julia simulator that is publicly available on GitHub [7]. The performance of this simulator is evaluated in Section 2.4. We observe that the theoretical improvement of dynamic rejection sampling translates into a gain of several orders of magnitude compared to previous attempts, both in terms of the grid size (up to 10^{84} nodes in the present paper, and virtually unlimited) and sampling speed (several millions of shortcuts per second can be drawn on a regular computer).

Fueled by the capacity to obtain quick and accurate results, even for very large grids, we give a fresh look at Kleinberg's grid through three independent studies. In Section 3.1, we investigate the tolerance of greedy routing with respect to the shortcut distribution. We show that, in practice, the model is more robust than what is predicted by the asymptotic bounds: even in very large grids, the range of exponents r that grant short routing paths is quite large. Then, the results of Section 3.2 show that the bounds proposed in [4] are not tight, except for $r = 0$ and $r = 2$. We use the simulator to conjecture tighter bounds. Finally, in Section 3.3, we compare the performance of greedy routing in Kleinberg's grid with the results of Milgram's experiment. We observe that, despite the simplicity of Kleinberg's model, the performance of the greedy routing algorithm is consistent with the *six degrees of separation* phenomenon when the grid parameters are tuned with realistic settings.

This paper focuses on the performance of the greedy routing algorithm. Unless stated otherwise, we assume that $p = q = 1$, so that each node has up to four local neighbors and one shortcut. For ease of notation, we let $e_r(n)$ denote the expected delivery time of the greedy routing algorithm in $G(n, r, 1, 1)$.

1.3. Related work

Theoretical Results. While we focus on the original model, let us give a brief, non-exhaustive overview of the extensions that have been proposed since. Most of the proposals modify either the graph model or the decentralized routing algorithm. Examples of the graph model extensions are generalizations of the original model (other grid dimensions or random number of shortcuts per node [8, 6, 9]), graphs inspired by peer-to-peer overlay networks [10], or arbitrary graphs augmented with shortcuts [11, 12]. Other proposals of routing algorithms typically try to enhance the performance of the greedy routing algorithm by granting nodes additional knowledge of the topology [13, 10, 14].

A large part of the above-mentioned works aims at improving the $O(\log^2 n)$ bound of the greedy routing algorithm. For example, in the small-world percolation model (a variant of Kleinberg's grid with $O(\log n)$ shortcuts per node), the expected delivery time of the greedy routing algorithm is $O(\log n)$ [10].

[2]The $\tilde{O}$ notation is traditionally used to ignore the polylogarithmic factor. In the present paper, we use $\tilde{O}$ and $\tilde{\Omega}$ to ignore the logarithmic cost of due to the integer representation. In particular, when multiple variables are considered, the implicit logarithm only applies to the size parameter n.

Experimental studies. Many empirical studies have analyzed routing in real-life social networks and the possible relation with Kleinberg's model (see, for example, [15, 6] and the references given there). To the best of our knowledge, numerical evaluations of the theoretical models are more limited. Such evaluations are usually performed by averaging R runs of the considered routing algorithm.

In [5], Kleinberg computes $e_r(n)$ for $n = 20,000$ and $r \in [0, 2.5]$, using $R = 1,000$ runs per estimate. However, he uses a torus instead of a regular grid (this will be discussed later in the paper). In [10], networks of size up to 2^{24}, corresponding to $n = 2^{12}$ in the grid, are investigated using $R = 150$ runs per estimate.

Closer to our work, Athanassopoulos *et al.* propose a study, centered on numerical evaluations, that considers Kleinberg's model as well as some variants with fixed source and destination nodes [16]. They compute $e_r(n)$ for values of n up to 3,000 and $r \in \{0, 1, 2, 3\}$, using $R = 900$ runs per estimate.

To compare with, in the present paper, we consider values of n up to 10^{42} and $r \in [0, 4]$, with at least $R = 10,000$ runs per estimate.

2. Fast estimation of the expected delivery time

This section describes and evaluates our simulator. In order to understand its originality, we first point out the main difficulty for building a fast simulator: shortcut sampling.

2.1. Complexity of the classical approaches

As stated in [16], the main computational bottleneck for simulating Kleinberg's model comes from the shortcuts. A naive approach would proceed as follows:

- The grid contains qn^2 shortcuts.
- For each shortcut, there are $n^2 - 1$ candidates which can be chosen with a non-zero probability. Hence, drawing a shortcut by inverse transform sampling has a cost $\tilde{\Omega}(n^2)$;
- The probability distribution of the shortcuts depends on the originating node, even if we use coordinates that are relative to this node. For example, a corner node only has two nodes at distance 2 while an inner node has four of them. This means that, up to symmetry, each node has a unique shortcut distribution [3]. This prevents us from mutualizing shortcut sampling among nodes.

In the end, building shortcuts as described above, for each of the R independent runs, yields a time complexity $\tilde{\Omega}(Rn^4)$ for evaluating $e_r(n)$, which is unacceptable in large grids.

The first issue is easy to address: as observed in [4, 10, 16], we can use the *principle of deferred decision* [17] and compute the shortcuts on the fly, as the path is built, because they are drawn independently and a shortcut is never encountered twice on a given path. This reduces the complexity to $\tilde{\Omega}(Rn^2 e_r(n))$.

[3] To take advantage of the symmetry, we can consider the isometric group of a square grid, which can be built with the quarter-turn and flip operations. However, its size is 8, so that, even using symmetry, there are at least $n^2/8$ distinct (non-isomorphic) distributions.

2.1.1. Torus approximation

In order to reduce the complexity even more, we can approximate the grid by a torus. This approach was adopted in [5, 10]. The toroidal topology brings two major simplifications compared to a flat grid:

- The probability distribution of the *shortcut offset*, giving the position of a shortcut relative to its originating node, does not depend on this node. Therefore, it is possible to use of the same shortcut drawing process for all nodes;
- Taking advantage of the strong radial symmetry of the shortcut distribution, we can pick up a shortcut by successively drawing a *radius* and an *angle*:
  - The radius gives the length of the shortcut offset, that is, the distance of the shortcut to its originating node;
  - The angle uniquely identifies a shortcut offset among all the shortcut offsets with the same radius.

  The numbers of possible radii and angles are both in $O(n)$, to compare with the number of possible shortcuts, which is in $\Omega(n^2)$. This reduction of the number of values to choose from can make the computation faster.

To illustrate the advantage of using a torus instead of a grid, consider drawing k shortcut offsets originating from k distinct nodes. In a grid, the cost in time is $\tilde{\Omega}(n^2k)$ if inverse transform sampling is used for each shortcut distribution. In the torus, we can precompute the probability distribution of the radii (which range from 1 to n, the maximal distance in the torus), then draw k radii at once and, for each of them, choose an angle uniformly at random. Assuming that sampling a float from the uniform distribution on $[0, 1)$ can be made in $\tilde{O}(1)$, the main bottleneck is drawing the radii. The complexity depends on the algorithm used to perform this operation. For example, with Vose's alias method [18], it can be performed in $\tilde{O}(n + k)$[4].

Since we aim at evaluating Kleinberg's grid as it was originally defined, we cannot use the torus approximation. However, using a variation of the rejection sampling method, it is possible to draw shortcuts in the original model (the flat grid) with the same complexity as in the torus.

2.2. Dynamic rejection sampling for drawing shortcuts

In order to keep a low computational complexity without making any simplification in the model, we propose to draw a shortcut originating from a given node u as follows:

1. We embed the actual grid G containing the lattice nodes of $G(n, r, p, q)$ in a virtual lattice B_u made of the points inside a ball with center u and radius $N = 2(n - 1)$ for the Manhattan distance d. The value of N guarantees that G is included in B_u, irrespective of the location of u.
2. We draw a shortcut offset inside B_u so that the probability to choose a shortcut v is proportional to $(d(u, v))^{-r}$. This can be done by drawing successively a radius and an angle, similarly to the torus approach:

[4] See Section 2.3 for more details on the drawing of radii.

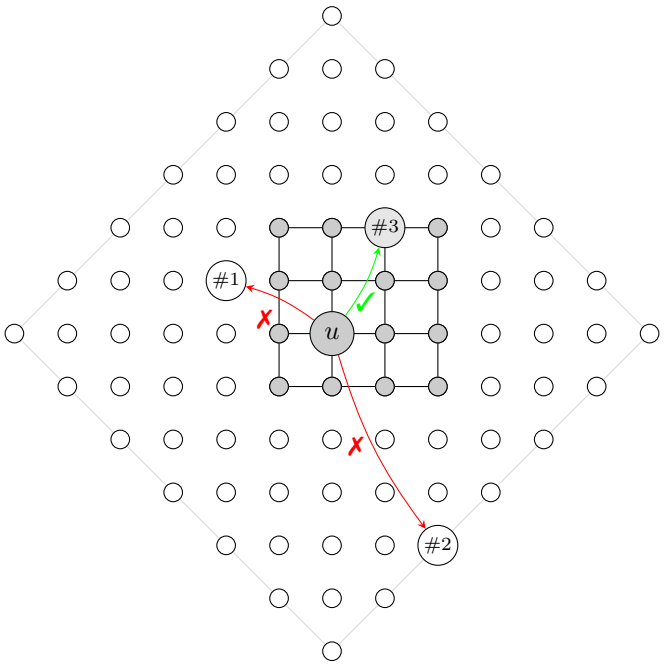


Figure 1: Main idea of the dynamic rejection sampling approach ($n = 4$).

Radius For each $i = 1, \ldots, N$, there are exactly $4i$ nodes with radius i in B_u, and each of them is drawn with a probability proportional to i^{-r}. The probability to draw any node with radius i is thus proportional to $(4i)i^{-r}$, that is to i^{1-r}. Therefore, we draw an integer between 1 and N so that the probability to draw i is $i^{1-r}/\sum_{j=1}^{N} j^{1-r}$. Section 2.3 presents and compares three drawing methods.

Angle All nodes with a given radius i are equally likely. We choose one of them by drawing an integer uniformly at random between 1 and $4i$.

3. This shortcut offset defines a unique node v, chosen with a probability proportional to $i^{-r} = (d(u, v))^{-r}$, among the $4i$ nodes at distance i from u in the virtual lattice B_u. If v belongs to the actual grid G, then it is returned as a shortcut, otherwise we try again (back to step #2).

This technique, illustrated in Figure 1, is inspired by the *rejection sampling* method [19]. By construction, it gives the correct distribution: the node v that is eventually returned belongs to the actual grid and has been drawn with a probability proportional to $(d(u, v))^{-r}$.

We call this *dynamic* rejection sampling because the shortcut distribution depends on the originating node u. Indeed, considering u as a relative center, the actual grid G moves with u and acts like an acceptance mask. On the other hand, the distribution over the virtual lattice B_u remains constant. Dynamic rejection sampling makes it possible to perform, in the flat grid, the two optimizations that motivated the torus approximation: the uniqueness of the shortcut offset distribution allows for mutualization, while

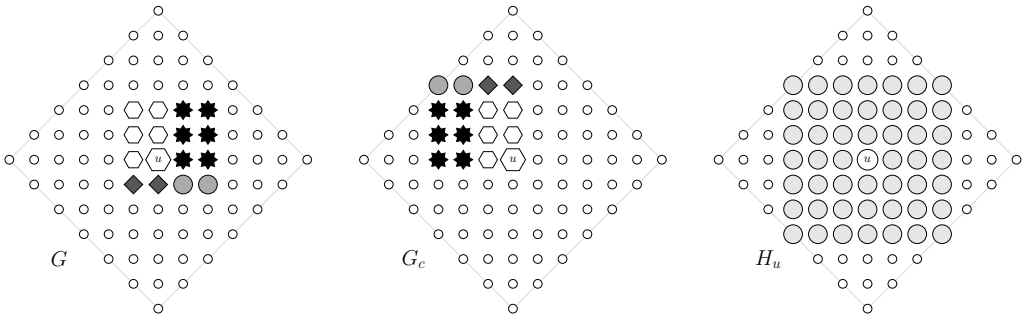


Figure 2: Graphical representation of the set G, G_c and H_u used in the proof of Lemma 1 ($n = 4$). The sub-lattices used to build a bijection between G and G_c are represented with distinct shapes and gray levels.

decoupling the radius and angle reduces the complexity because we draw two random variables among $O(n)$ instead of a single vector among $\Omega(n^2)$.

The only possible drawback of this approach is the number of attempts required to draw a correct shortcut. Luckily, this number is contained.

Lemma 1. *The probability that a node drawn in B_u belongs to G is at least $\frac{1}{8}$.*

Proof. We will prove that

$$\frac{\sum_{v \in G \setminus \{u\}} (d(u,v))^{-r}}{\sum_{v \in B_u \setminus \{u\}} (d(u,v))^{-r}} > \frac{1}{8}.$$

We use the fact that the probability decreases with the distance, combined with geometric arguments. Let G_c be one of the four $n \times n$ lattices that have u as one of their corner. Let H_u the $(2n-1) \times (2n-1)$ lattice centered in u.

Considering the probability of drawing a node in $G \setminus \{u\}$, the worst case is when u is in a corner of G: there is a bijection f from G to G_c such that, for all $v \in G \setminus \{u\}$, we have $d(u, f(v)) \geq d(u, v)$. Such a bijection can be obtained by splitting G into $G \cap G_c$ and three other sub-lattices that are flipped over $G \cap G_c$ (see Figure 2). This gives

$$\sum_{v \in G \setminus \{u\}} (d(u,v))^{-r} \geq \sum_{v \in G \setminus \{u\}} (d(u,f(v)))^{-r} = \sum_{v \in G_c \setminus \{u\}} (d(u,v))^{-r}.$$

Then, we observe that the four lattices G_c that can obtained (depending on the corner occupied by u) fully cover H_u. In fact, axis nodes are covered redundantly. We obtain

$$\sum_{v \in H_u \setminus \{u\}} (d(u,v))^{-r} < 4 \sum_{v \in G_c \setminus \{u\}} (d(u,v))^{-r}.$$

Lastly, if we fold $B_u \setminus H_u$ back into H_u like the corners of a sheet of paper, we obtain a strict injection g from $B_u \setminus H_u$ to $H_u \setminus \{u\}$ (the diagonal nodes of H_u are not covered) such that, for all $v \in B_u \setminus H_u$, we have $d(u,v) \geq d(u, g(v))$. This gives

$$\sum_{v \in B_u \setminus H_u} (d(u,v))^{-r} \leq \sum_{v \in B_u \setminus H_u} (d(u,g(v)))^{-r} < \sum_{v \in H_u \setminus \{u\}} (d(u,v))^{-r}.$$

This concludes the proof, as we get

$$\frac{\sum_{v\in G\setminus\{u\}}(d(u,v))^{-r}}{\sum_{v\in B_u\setminus\{u\}}(d(u,v))^{-r}} \geq \frac{\sum_{v\in G_c\setminus\{u\}}(d(u,v))^{-r}}{\sum_{v\in H_u\setminus\{u\}}(d(u,v))^{-r}+\sum_{v\in B_u\setminus H_u}(d(u,v))^{-r}} > \frac{1}{8}.$$

□

When $r = 0$ (uniform shortcut distribution), the bound $\frac{1}{8}$ is asymptotically tight: the success probability is exactly the ratio between the number of nodes in $G \setminus \{u\}$ and $B_u \setminus \{u\}$. This ratio is $(n^2-1)/(4(n-1)(2n-1))$, which tends to $\frac{1}{8}$ when n goes to infinity. On the other hand, as r increases, the probability mass concentrates more and more around u (hence in G), so that we should expect better performance. This intuition will be corroborated by the numerical results in Section 2.4.2.

Remarks.

- The dynamic rejection sampling approach can be used in other variants of Kleinberg's model, like in higher dimension grids or when the number of shortcuts per node is a random variable (like in [9]). The only requirement is the existence of some *root* distribution (like the distribution over B_u) that can be carved to match any of the possible distributions with a simple acceptance test.
- Only the nodes from H_u may belong to G, so that nodes from $B_u \setminus H_u$ are always uselessly sampled. For example, in Figure 1, this represents 36 nodes out of 84. By drawing shortcuts in $H_u \setminus \{u\}$ instead of $B_u \setminus \{u\}$, we would increase the success rate lower bound to $1/4$. However, this would make the algorithm implementation more complex (the number of nodes at distance i would not always be $4i$), which, in practice, is not worth the factor 2 improvement of the lower bound. Yet, this approach may become worthy in a grid with a higher dimension β. Adapting the proof from Lemma 1, we observe that using a ball of radius $\beta(n-1)$ leads to a bound $\beta!(2\beta)^{-\beta}$, while restricting the support of the root distribution to a grid of side $2n-1$ leads to $2^{-\beta}$. Both bounds are asymptotically tight for $r = 0$. Therefore, the grid approach is $\beta^\beta/\beta!$ times more efficient that the ball approach in terms of the rejection rate[5].

2.3. Drawing shortcut radii

In the dynamic rejection sampling technique described above, all elementary steps but drawing the shortcut offset radii can easily be implemented in $\tilde{O}(1)$. The main computational challenge is to draw the radii. Let us recall that the shortcut offsets belong to a ball of radius $N = 2(n-1)$, and that, for each $i = 1, \ldots, N$, radius i is chosen with probability $i^{1-r}/\sum_{j=1}^{N} j^{1-r}$. Therefore, drawing the radii is equivalent to drawing random variables from a Zipf distribution with parameters $r-1$ and N. Sampling from a Zipf distribution is not natively implemented in the programming language we chose (Julia). In this section, we propose and compare three methods, and analyze their impact on the computational complexity of the simulator (we will benchmark their effective performance in Section 2.4).

[5]The gain $\beta^\beta/\beta!$ grows exponentially with β.

2.3.1. Bulk sampling

Taking advantage of the fact that all radii are drawn independently from the same distribution, we can precompute the probability mass function over the $N = 2(n-1)$ possible values, and then use a generic sampling method to draw k values at once. Once all k values are consumed, we draw another batch of k values, and so on until R independent runs have been performed. This is the approach that has been adopted in [20].

The theoretical complexity of the bulk approach is tightly connected to the sampling method that is used. Our implementation uses the *sample* function of Julia *StatsBase* module[6], which relies on the alias method [21]. It requires $\tilde{O}(n)$ in memory and takes an initialization time $\tilde{O}(n)$, after which each value can be drawn in $\tilde{O}(1)$. Overall, the time to draw k values is $\tilde{O}(n+k)$.

Note that the built-in *sample* function that we use does not keep track of the previous computations, so that the initialization phase must be performed at each call of the function.

To give the complexity of the bulk approach, we observe that the average number of shortcuts we need to draw over the R runs is $\frac{Re_r(n)}{s_r(n)}$, where $s_r(n)$ is the success rate of the dynamic rejection sampling technique with parameters r and n. As $s_r(n) \in [1/8, 1]$ (cf. Lemma 1), this means that we should aim for a complexity $\tilde{O}(n + Re_r(n))$: $\tilde{O}(n)$ to perform at least one initialization step and $\tilde{O}(Re_r(n))$ to draw (on average) $\frac{Re_r(n)}{s_r(n)}$ values. With a fixed bulk size k, the actual complexity is $\left\lceil \frac{Re_r(n)}{s_r(n)k} \right\rceil \tilde{O}(n+k)$. We chose $k = n$, which ensures the following:

- The memory usage required to store the k values does not exceed the $\tilde{O}(n)$ memory that is already required by the alias method;
- It gives the desired time complexity $\tilde{O}(n + Re_r(n))$.

Remark. Other choices of k may perform better than $k = n$ in terms of the actual speed. For example, $k = n$ is not efficient when $n \gg Re_r(n)$ because the bulk is over-sized and a lot of shortcut offsets are drawn but not used. Ideally, k should be a tight upper bound of the number of radii to draw, but we didn't explore this optimization for the following reasons:

- Implementing this optimization requires bootstrapping the estimation of $e_r(n)$ and $s_r(n)$;
- The resulting bulk may be too large to fit in memory;
- The time complexity would be of the same order as with $k = n$.

2.3.2. Vose's alias method

As a first attempt to enhance the performance of our simulator, we implemented our own sampler based on the alias method. Compared to the bulk approach, we obtain the following improvements:

[6] `http://statsbasejl.readthedocs.io/en/latest/sampling.html`

- We used the algorithm proposed by Vose in [18] to reduce the initialization time. Vose's initialization starts with an array containing the values of the probability mass function of the radii, then re-organizes the values in this array, and finally defines a second array of aliases[7]. It requires $O(n)$ elementary operations against $O(n \log n)$ for the version proposed in [21]. In both cases, the time complexity is $\tilde{O}(n)$ and the memory occupation $\tilde{O}(n)$. However, the logarithmic factor in Vose's algorithm only comes from the cost of the integer representation.

- After the initialization step, we can sample a radius with a time complexity $\tilde{O}(1)$. It boils down to choosing an index of the arrays uniformly at random and then drawing a Bernoulli random variable to decide whether the index or its alias is returned.

In the end, the time and memory complexities are of the same order as with the bulk method: $\tilde{O}(n + Re_r(n))$ and $\tilde{O}(n)$, respectively. Yet, in practice, a better performance should be expected from Vose's alias method because some useless computations of the bulk approach are avoided, such as performing the initialization several times for the same run or drawing more radii than necessary.

2.3.3. Rejection sampling

The methods we have just described are both limited by the $\tilde{O}(n)$ time and memory cost of the initialization step. This is unavoidable because they are both designed to work with an arbitrary distribution and thus require to build and scan the probability mass function. Recalling that we use a Zipf distribution, we now show how to obliterate the initialization cost.

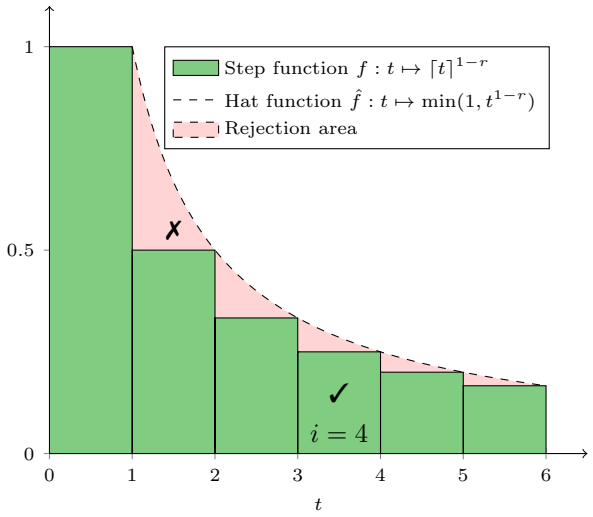


Figure 3: Illustration of the rejection sampling method for drawing radii ($n = 4, r = 2$).

[7] Cf. [18] for more details. A nice pedagogic explanation of alias methods, including the one proposed by Vose, is also available at `http://www.keithschwarz.com/darts-dice-coins/`.

We consider a rejection sampling method inspired by [22]. The principle is summarized in Figure 3. We represent the probability mass function of the target Zipf distribution by a step function. This step function can be closely dominated by some hat function which is simpler to compute. The rejection sampling method consists in drawing points uniformly at random under the curve of the hat function until one of them falls under the target step function; when it is the case, the corresponding integer is returned.

In detail, we distinguish between two cases, depending on the monotonicity of the probability mass function:

- If $r \leq 1$, the function $i \mapsto i^{1-r}$ is non-decreasing. We represent the Zipf distribution by the step function f defined on $[1, N+1[$ by $f(t) = \lfloor t \rfloor^{1-r}$, which is dominated by the hat function $\hat{f}$ defined by $\hat{f}(t) = t^{1-r}$. We use inverse transform sampling to sample a value x from $\hat{f}$ on $]1, N+1[$. Then, we draw a second value from the Bernoulli distribution with parameter $i^{1-r} / \int_i^{i+1} \hat{f}(t)dt$ to decide whether we return $i = \lfloor x \rfloor$ or we sample another value.

- If $r > 1$, the function $i \mapsto i^{1-r}$ is decreasing. We represent the Zipf distribution by the step function f defined on $]0, N[$ by $f(t) = \lceil t \rceil^{1-r}$, which is dominated by the hat function $\hat{f}$ defined by $\hat{f}(t) = \min(1, t^{1-r})$ (the function is capped by 1 to avoid divergence issues around 0). We use inverse transform sampling to sample a value x from $\hat{f}$ on $]0, N[$. Then, we draw a second value from the Bernoulli distribution with parameter $i^{1-r} / \int_{i-1}^{i} \hat{f}(t)dt$ to decide whether we return $i = \lceil x \rceil$ or we sample another value. Note that, if $i = 1$, f and $\hat{f}$ are equal and the Bernoulli test can be omitted.

2.4. Performance evaluation

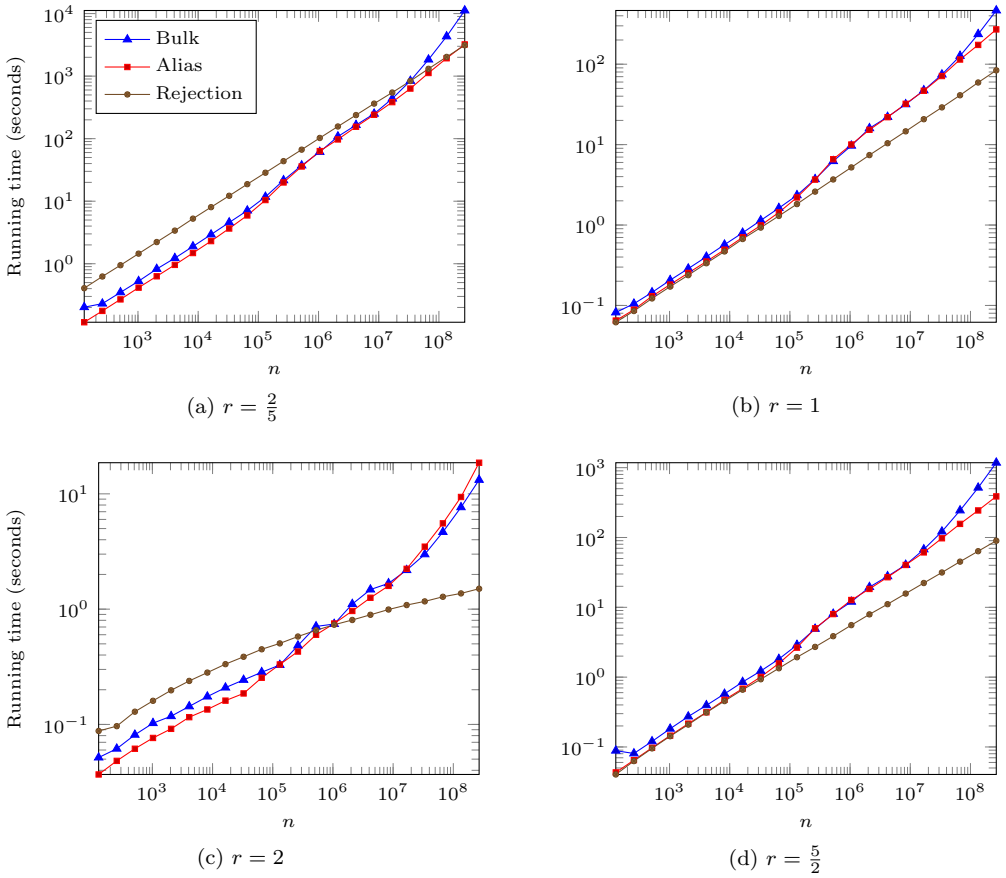
We are now interested in evaluating the performance of the algorithms that we presented in Sections 2.2 and 2.3. We used them to implement a simulator that estimates $e_r(n)$, or, more generally, the performance of greedy routing in a $G(n, r, p, q)$ grid. The simulator is written in Julia 0.5. For readers interested in looking under the hood, the code is available on GitHub [7]. We chose to display it under a *Jupyter Notebook* format, which allowed us to embed some additional details about the algorithm design in Markdown.

Three versions of the simulator are available, one for each radius sampling algorithm. We call them *Bulk*, *Alias* and *Rejection* for short (cf. Section 2.3). *Bulk* and *Alias* use a standard representation of integers over 64 bits, as their memory footprint $\tilde{O}(n)$ limits the feasible values of n. Since *Rejection* is not affected by that limit, its design automatically selects the best integer representation among three possibilities: 64 bits, 128 bits, or arbitrary long.

Simulations were executed on a standard commodity computer equipped with a 3.30GHz processor and 32 gigabytes of memory. Unless said otherwise, $e_r(n)$ is obtained by averaging $R = 10,000$ runs. We mainly focus on $r \in [0, 3]$, except in Section 3.2, where investigating $r \in [0, 4]$ was required. The grid side n ranges from $n = 2^7$ (corresponding to $(2^7)^2 \approx 16,000$ nodes) to $n = 2^{28}$ (about 72 quadrillions nodes). A notable exception is Figure 7, where n goes up to $n = 10^{42}$ to demonstrate the capacity of our simulator

to consider very large grids ($(10^{42})^2 = 10^{84}$ is the order of magnitude of the number of particles in the observable universe).

2.4.1. Overall performance of the simulator



(a) $r = \frac{2}{5}$ (b) $r = 1$

(c) $r = 2$ (d) $r = \frac{5}{2}$

Figure 4: Computing $e_r(n)$ using $R = 10,000$ runs.

The average running time of the simulations is plotted in Figure 4 as a function of n, for $r \in \{\frac{2}{5}, 1, 2, \frac{5}{2}\}$. We observe running times ranging from seconds to a few minutes. To compare with, in [16] (which is, to the best of our knowledge, the only work disclosing computation times[8]), a single run takes about 4 seconds for $n = 400$. With a similar time budget, our best implementation can run $R = 10,000$ simulations in a grid of side up to: $n = 2^{15} \approx 33,000$ for $r = \frac{2}{5}$; $n = 2^{19} \approx 524,000$ for $r = 1$ or $r = \frac{5}{2}$; $n = 2^{44} \approx 18 \times 10^{12}$ for the optimal exponent $r = 2$. Note that values of n higher than 2^{28} are not displayed in Figure 4 to keep homogeneous scales, but a higher scale is available in Figure 7.

[8] Apart from the early version [20] of the present article.

The main message conveyed by Figure 4 is that *Alias* and *Rejection* improve the Bulk method introduced in [20], which was already several orders of magnitude faster than any previous work we are aware of. If we look in more details, we observe the following performance trends:

- The shape of the performance of *Alias* is similar to that of *Bulk*, although it is better in most cases;
- *Rejection* seems to be especially efficient for large values of n. Note that, for $r = \frac{2}{5}$, it does not seem to bring any obvious improvement compared to *Bulk* and *Alias*.

2.4.2. Detailed analysis

For a better understanding of the simulation performance, we separately consider two performance indicators: the overhead induced by the dynamic rejection sampling and the time cost per sampled radius.

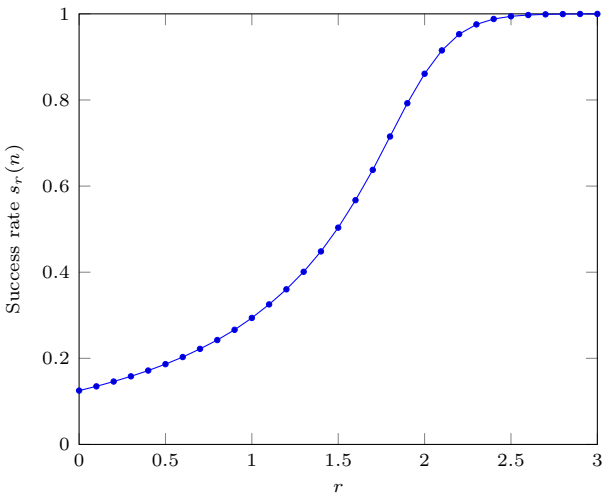
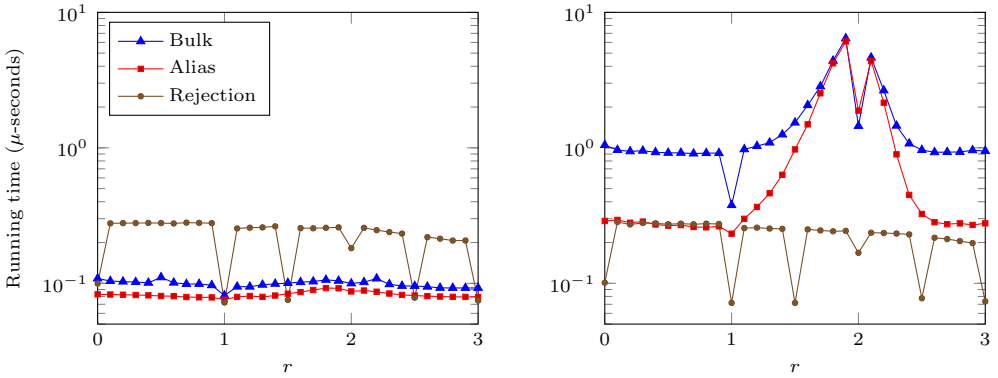


Figure 5: Fraction of the shortcuts drawn in B_u that belongs to G, observed during a computation of $e_r(n)$ ($n = 2^{14}, R = 10,000$).

Dynamic rejection sampling. The performance of the simulator is affected by the success rate of the dynamic rejection sampling approach which, as stated in Lemma 1, has a proven lower bound $\frac{1}{8}$. Figure 5 gives a numerical evaluation of the success rate $s_r(n)$ as a function of r, for $n = 2^{14}$ (the actual value of n has no significant impact as long as it is large enough). To do this, we counted the proportion of the shortcuts that belong to the grid G among those that were drawn. We verify Lemma 1: the success rate is always at least $1/8 = 0.125$, which is a tight bound for $r = 0$. We also confirm the intuition that, as r increases, the probability mass concentrates, yielding higher success rates. For $r = 1$, the rate is about 0.29, and it climbs to 0.86 for $r = 2$. Failed shortcuts become negligible for $r \geq 2.5$, where the success rate is greater than 0.99. Overall, Figure 5 shows that the cost of drawing some shortcuts outside G is small compared to the benefits to drawing all shortcuts from the same distribution.

Remark. Figure 5 shows that $r = 2.5$ has a much better success rate than $r = 1$, while we see in Figure 4 that running times tend to be higher for $r = 2.5$. The reason is that $e_1(n)$ is lower than $e_{2.5}(n)$, which overcompensates the difference of the success rate.



(a) $n = 2^{14}$ and $R = 10,000$ (b) $n = 2^{28}$ and $R = 10,000$

Figure 6: Cost per shortcut as a function of r (dynamically rejected shortcuts are included).

Drawing the shortcut radii. Figure 6 presents the average cost per shortcut as a function of r, for $n \in \{2^{14}, 2^{28}\}$ and $R = 10,000$. This cost, which we use to benchmark the performance, is defined as the ratio of the total computation time to the number of shortcut offsets that we were drawn during the simulation, including the ones that were eventually rejected. The success rate of the dynamic rejection sampling mainly depends on r but it is unaffected by the choice of the method used to draw radii. Therefore, the proposed metric gives an evaluation of the radii sampling methods that is unbiased by that success rate of dynamic rejection sampling. We make the following observations:

- With the parameters we considered, *Rejection* requires less than $0.3\mu s$ to draw a shortcut and seems unaffected by the choice of the grid side n. It confirms the complexity analysis made in Section 2.3: radii are drawn in $\tilde{O}(1)$ with no initialization cost.

- The two other methods are deeply affected by the choice of n. This is likely due to the cost of the initialization step which, divided by the number of shortcuts to draw, has an impact of order $\frac{ns_r(n)}{Re_r(n)}$. We remark that the worst performance is achieved when r is close to 2 and n large, which maximizes $\frac{n}{e_r(n)}$.

- Despite their theoretical disadvantage, *Bulk* and *Alias* outperform *Rejection* when n is small. The reason is that, apart from the issue of the initialization, *Bulk* and *Alias* only use simple mathematical operations. For instance, if we compare *Alias* and *Rejection*, we observe that both consume two random numbers to draw a shortcut, after which *Alias* just needs up to two memory accesses while *Rejection* requires multiple calls to the *power* function, which is more costly.

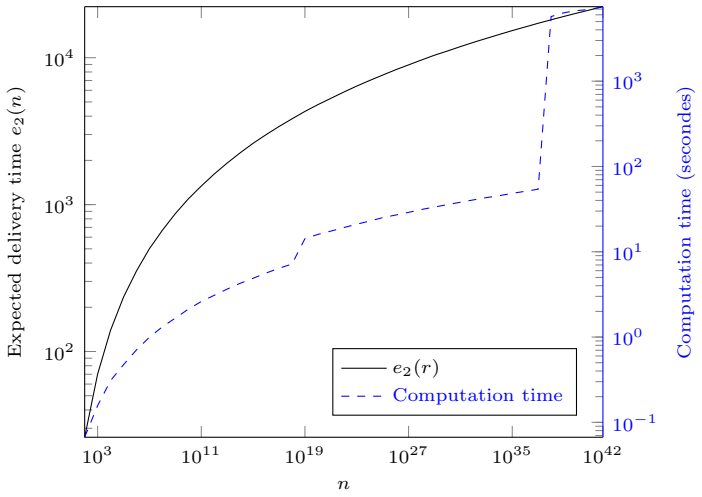


Figure 7: Expected delivery and computation time, with $r = 2$.

- *Alias* performs globally better than *Bulk*. This is not surprising, as the former is essentially an enhanced version of the latter (cf. Section 2.3).

- Some values of r give a simulation time which is noticeably shorter, namely $r \in \{0, 1, \frac{3}{2}, 2, \frac{5}{2}, 3\}$ for *Rejection* and $r \in \{1, 2\}$ for the two other methods. These values correspond to cases where the code or Julia internal optimizations can simplify some computations. For example, when $r = 1.5$, *Rejection* simplifies computations like $x \mapsto x^{1-r}$ into $x \mapsto 1/\sqrt{x}$, which is faster. Note that the improvement affects more *Rejection* because it is the method that consumes the most calls to the *power* function. In particular, this explains why, in Figure 4, *Rejection* performs relatively better for $r \in \{1, 2, \frac{5}{2}\}$ than for $r = \frac{2}{5}$.

To summarize the comparison between *Bulk*, *Alias* and *Rejection*:

- We included *Bulk* in the benchmark because it is the method proposed in [20] and uses an off-the-shelf sampling function. Yet, it should be considered as a legacy algorithm as *Alias* performs better.

- *Alias* is of interest if the available memory can support it and if the initialization time is small compared to the simulation time, i.e., if $Re_r(n) \gg n$.

- *Rejection* should be used otherwise. Actually, we recommend using *Rejection* as a default choice in practice: it works for arbitrary large grids, and the cases where *Alias* outperforms *Rejection* typically correspond to small values of n. For these values, the computation time tends to be small anyway (compared to other choices of parameters), independently of the method we use.

2.4.3. Simulating the Universe

To conclude this performance evaluation, Figure 7 illustrates the capabilities of *Rejection* by displaying the expected delivery time and the computation time as functions

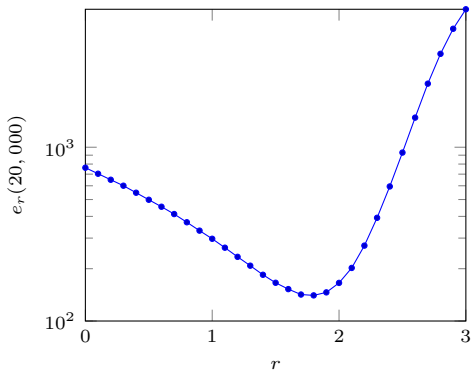


Figure 8: Expected delivery time for $n = 20,000$.

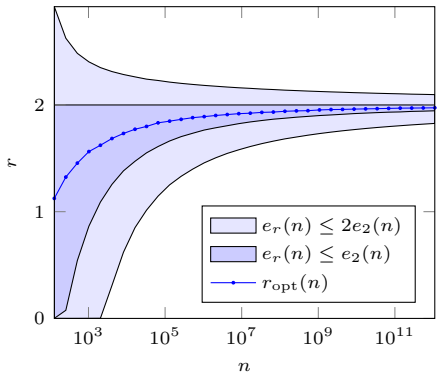


Figure 9: Reasonable values of r.

of n, with $r = 2$. While the computation time stays reasonable, the highest displayed value, $n = 10^{42}$, corresponds to grids containing as many nodes as there are particles in the observable universe.

The performance jumps near $n = 10^{19}$ and $n = 10^{38}$ are the observable consequence of the complexity factor $\log n$ implied by the *soft-O* and *soft-*Ω notations: they are the thresholds where the simulator needs to modify its integer representation for representing coordinates in the grid.

3. Applications

Given the tremendous amount of strong theoretical results on small-world routing, we can question the interest of proposing a simulator (even a fast one!). In this section, we prove the interest of the numerical evaluations through three (almost) independent studies.

3.1. Efficient enough exponents

In [5], Kleinberg gave an estimation of $e_r(20,000)$ using a torus approximation (cf. Section 2.1.1). A few years later, he discussed the results in more details, observing that [6]:

- $e_r(n)$ stays quite stable for $r \in [1.5, 2]$;
- the best value of r is actually slightly lower than 2.

We believe that these observations are very important as they show that greedy routing in Kleinberg's grid is more robust than what is predicted by the theory: it is efficient as long as r is *close enough* to 2. Using our simulator, we can perform the same experiment on the flat grid. As shown by Figure 8, the results are quite similar to the ones observed in [5].

Yet, there is a small but *essential* difference between the two experiments: Kleinberg approximated his grid by a torus while we stay true to the original model. Why do we use the word *essential*? Both shapes have the same asymptotic behavior (the proofs in

[5] are straightforward to adapt), so why should we care? It seems to us that practical robustness is an essential feature if we want to accept Kleinberg's grid as a reasonable model for routing in social networks. To the best of our knowledge, no theoretical work gives quantitative results on this robustness in finite grids, so we need to rely on numerical evaluations. However, when Kleinberg uses a torus approximation, we can not rule out that the observed robustness is a by-side effect of the toroidal topology. The observation of a similar phenomenon on a flat grid discards this hypothesis. In fact, it suggests (without proving) that the robustness with respect to the exponent for grids of finite size may be a general phenomenon.

We now propose to investigate this robustness in more detail. We have evaluated the following values which outline, for a given n, the values of r that can be considered reasonable for performing greedy routing:

- The value of r that minimizes $e_r(n)$, denoted by r_{opt};
- The smallest value of r such that $e_r(n) \leq e_2(n)$, denoted by $r_{\min}(e_2(n))$;
- The smallest and largest values of r such that $e_r(n) \leq 2e_2(n)$, denoted by $r_{\min}(2e_2(n))$ and $r_{\max}(2e_2(n))$, respectively.

The results are displayed in Figure 9. All values but r_{opt} are computed by bisection. For r_{opt}, we use a Golden section search [23]. Finding a minimum requires more accuracy, so the search for r_{opt} uses $R = 1,000,000$ runs per estimate. Luckily, as the computation operates by design through near-optimal values, we can increase the accuracy while keeping reasonable running times.

Besides confirming that $r = 2$ is asymptotically the optimal value, Figure 9 shows that the range of reasonable values for finite grids is quite comfortable. For example, considering the range of values where $e_r(n)$ is less than twice $e_2(n)$, we observe that:

- For $n \leq 2^{11}$ (less than four million nodes), any r between 0 and 2.35 works;
- For $n \leq 2^{14}$ (less than 270 million nodes), the range is between 0.85 and 2.26;
- Even for $n = 2^{24}$ (about 280 trillions nodes), all values of r between 1.58 and 2.16 can still be considered as *efficient enough*.

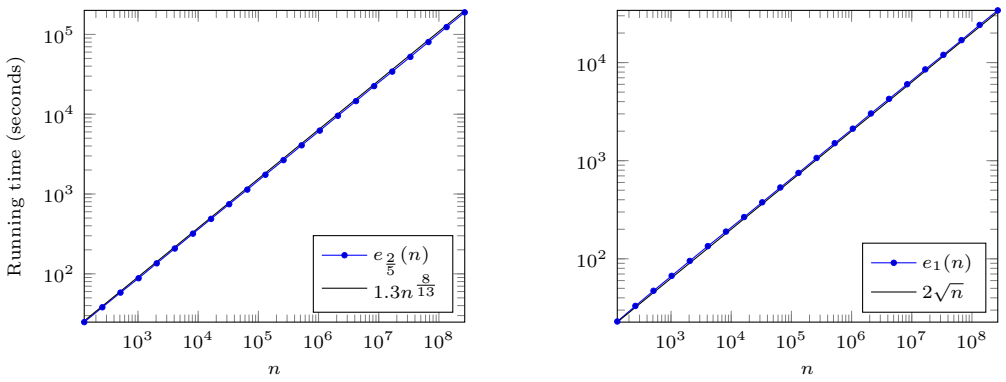
3.2. Asymptotic Behavior

Our simulator can be used to verify the theoretical bounds proposed in [4]. For example, Figure 10 shows $e_r(n)$ for r equal to $\frac{2}{5}$, 1, 2, and $\frac{5}{2}$.

As predicted, $e_r(n)$ seems to behave like $\log^2 n$ for $r = 2$ and like n^α for the two other cases. Yet, the exponents we found differ from the ones proposed in [4]. For $r = \frac{2}{5}$, we find $\alpha = \frac{8}{13}$, while the lower bound is $\frac{8}{15}$; for both $r = 1$ and $r = 2.5$, we observe $\alpha = \frac{1}{2}$ instead of $\frac{1}{3}$. Intrigued by the difference, we want to compute α as a function of r.

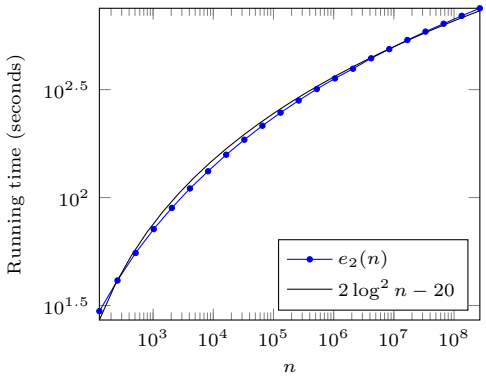
If $e_r(n)$ behaves exactly like n^α, then α can be obtained by choosing two distinct values n_1 and n_2 and computing

$$\frac{\log(e_r(n_2)) - \log(e_r(n_1))}{\log n_2 - \log n_1}.$$

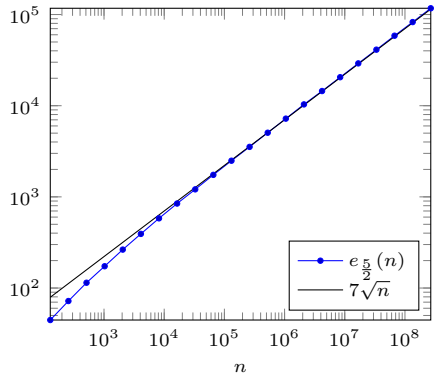


(a) $r = \frac{2}{5}$

(b) $r = 1$



(c) $r = 2$



(d) $r = \frac{5}{2}$

Figure 10: Expected delivery time for different values of r.

However, $e_r(n)$ may not behave exactly like n^α. For example, in Figure 10, we see that a $\log^2 n$ curve appears to have a positive slope in a logarithmic scale, even for large values of n, while we should except $\alpha = 0$ for $r = 2$. To mitigate the possible distortion, we used, for each value of r we considered, the highest possible values of n_1 and n_2 that could be computed in a reasonable time.

The results are displayed in Figure 11. The range of r was extended to $[0, 4]$ due to the observation of a new transition at $r = 3$.

Based on Figures 10 and 11, it seems that the bounds proposed by Kleinberg in [4] are tight only for $r = 0$ and $r = 2$. Formally proving more accurate bounds is beyond the scope and spirit of this paper, but we can conjecture new bounds hinted by our simulations:

- For $0 \leq r < 2$, we have $e_r(n) = \Theta(n^{\frac{2-r}{3-r}})$;
- For $2 < r < 3$, we have $e_r(n) = \Theta(n^{r-2})$;
- For $r > 3$, we have $e_r(n) = \Theta(n)$.

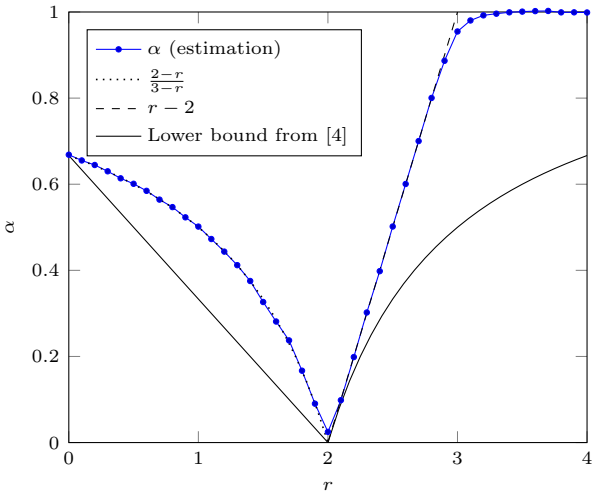


Figure 11: Estimates of the exponent α.

These conjectured bounds are consistent with the ones proposed in [8], where it is proved that, in the 1-dimensional ring, we have:

- For $0 \leq r < 1$, $e_r(n) = \Omega(n^{\frac{1-r}{2-r}})$;
- For $r = 1$, $e_r(n) = \Theta(\log^2 n)$;
- For $1 < r < 2$, $e_r(n) = O(n^{r-1})$;
- For $r = 2$, $e_r(n) = O(n\frac{\log\log n}{\log n})$;
- For $r > 2$, $e_r(n) = O(n)$.

It is likely that the proofs in [8] can be adapted to the 2-dimensional grid, but some additional work may be necessary to demonstrate the tightness of the bounds. Moreover, our estimates may have missed some slower-than-polynomial variations. For example, the logarithms in the bounds for $r = 2$ in [8] may have a counterpart in the grid for $r = 3$. This would explain why the estimates are not sharp around that critical value (like for the case $r = 2$ and the term in $\log^2 n$, except that, for $r = 2$, very high values of n can be computed in a reasonable time, leading to a better estimate of the exponent α).

Remark. For $r \geq 3.5$, we have observed that $e_r(n) \approx \frac{2}{3}n$, which is the expected length of the delivery path in absence of shortcuts: for these high values of r, almost all shortcuts link to an immediate neighbor of the current node, which makes them useless, and the rare exceptions are so short that they do not make any noticeable difference.

3.3. Six degrees of separation

What makes Milgram's experiments [2, 3] so famous is the surprising observation that only a few hops are required to transmit messages in social networks, a phenomenon called *six degrees of separation* in the popular culture.

Yet, the values of $e_r(n)$ that we observed until now are quite far from the magic number six. For example, in Figure 8, the lowest value is 140. In addition to the asymptotic lack of robustness with respect to the exponent (discussed in Section 3.1), this may partially motivate the amount of work accomplished to increase the realism of the model and the performance of the routing algorithm.

In our opinion, a good model should be as simple as possible while being able to accurately predict the phenomenon that needs to be explained. We propose here to answer a simple question: is the genuine model of greedy routing in $G(n, r, p, q)$ a good model? Sure, it is quite simple. To discuss its accuracy, we need to tune the four parameters n, r, p, and q to fit the conditions of Milgram's experiments as honestly as we can.

Size Milgram's experiments were conducted in the United States, with a population of about 200,000,000 individuals at the late sixties. All inhabitants were not susceptible to participate in the experiments: under-aged, undergraduate or disadvantaged people may be considered as *de facto* excluded from the experiments. Taking that into consideration, the correct n is probably somewhere in the range $[5000, 14000]$. We propose to set $n = 8,500$, which corresponds to about 72,000,000 potential subjects.

Exponent In [6], Kleinberg investigates how to relate the r-harmonic distribution with real-life observations. He surveys multiple social experiments and discusses the correspondence with the exponent of his model, which gives estimates of r between 1.75 and 2.2.

Neighborhood The default value $p = q = 1$ means that there are no more than five "acquaintances" per node. This is quite small compared to what is observed in real-life social networks. For example, the famous Dunbar's number, which estimates the number of *active* relationships, is 150 [24]. More recent studies seem to indicate that the average number of acquaintances is larger, ranging from 250 to 1500 (see [25, 26, 27] and references within). We propose to set p and q so that the neighborhood size $2p(p+1)+q$ is about 600, the value reported in [26]. Regarding the partition between the local links (p) and the shortcuts (q), we consider three typical scenarios:

- Shortcut scenario ($p = 1$, $q = 600$): the neighborhood is almost exclusively made of shortcuts, and local links are only here to ensure the termination of greedy routing;
- Balanced scenario ($p = 10$, $q = 380$);
- Local scenario ($p = 15$, $q = 120$, with a value of q that is not too far from Dunbar's number).

Having set all parameters, we can evaluate the performance of greedy routing. The results are displayed in Figure 12. We observe that the expected delivery time roughly stands between five and six for a wide range of exponents:

- Shortcut scenario: $r \in [1.4, 2.3]$;
- Balanced scenario: $r \in [1.3, 2.3]$;

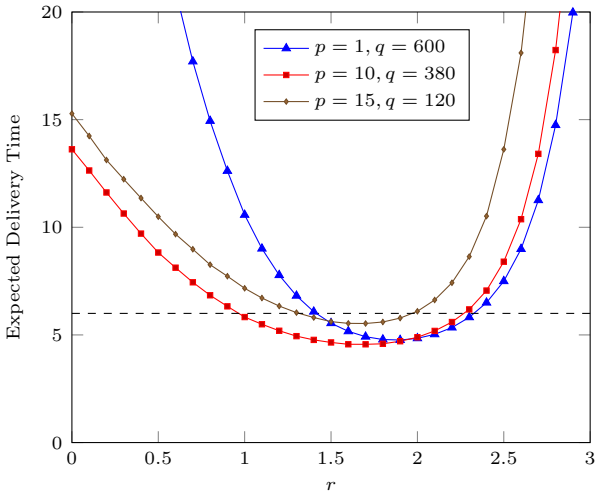


Figure 12: Performance of greedy routing for parameters inspired by Milgram's experiments ($n = 8,500$).

- Local scenario: $r \in [1.3, 2]$.

Except for the local scenario, which leads to slightly higher routing times for $r > 2$, the six degrees of separation are achieved for all values of r that are consistent with the observations surveyed in [6]. This allows us to answer our question: the augmented grid proposed by Kleinberg is indeed a good model to explain the *six degrees of separation* phenomenon.

References

[1] F. Karinthy, Láncszemek (1929).
[2] S. Milgram, The small world problem, Psychology Today 67 (1) (1967) 61–67 (1967).
[3] J. Travers, S. Milgram, An experimental study of the small world problem, Sociometry 32 (1969) 425–443 (1969).
[4] J. Kleinberg, The small-world phenomenon: An algorithmic perspective, in: Proceedings of the 32nd ACM Symposium on Theory of Computing, 2000, pp. 163–170 (2000).
[5] J. Kleinberg, Navigation in a small world, Nature 406 (2000) 845 (Aug 2000).
[6] D. Easley, J. Kleinberg, The small-world phenomenon, in: Networks, Crowds, and Markets: Reasoning About a Highly Connected World, Cambridge University Press, 2010, Ch. 20, pp. 611–644 (2010).
[7] C. Comte, F. Mathieu, A Julia simulator for greedy routing in Kleinberg's grid., https://github.com/balouf/Kleinberg (March 2017).
[8] L. Barrière, P. Fraigniaud, E. Kranakis, D. Krizanc, Efficient routing in networks with long range contacts, in: Proceedings of the 15th International Conference on Distributed Computing, DISC '01, London, UK, 2001, pp. 270–284 (2001).
[9] P. Fraigniaud, G. Giakkoupis, Greedy routing in small-world networks with power-law degrees, Distributed Computing 27 (4) (2014) 231–253 (2014).
[10] G. S. Manku, M. Naor, U. Wieder, Know thy neighbor's neighbor: The power of lookahead in randomized P2P networks, in: Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing (STOC), ACM, 2004, pp. 54–63 (2004).
[11] P. Fraigniaud, C. Gavoille, A. Kosowski, E. Lebhar, Z. Lotker, Universal Augmentation Schemes for Network Navigability: Overcoming the $\sqrt{(}n)$-Barrier, Theoretical Computer Science 410 (21-23) (2009) 1970–1981 (2009).

[12] P. Fraigniaud, G. Giakkoupis, On the searchability of small-world networks with arbitrary underlying structure, in: Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC), 2010, pp. 389–398 (Jun. 6–8 2010).
[13] P. Fraigniaud, C. Gavoille, C. Paul, Eclecticism shrinks even small worlds, Distributed Computing 18 (4) (2006) 279–291 (2006).
[14] C. Martel, V. Nguyen, Analyzing kleinberg's (and other) small-world models, in: Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing, PODC '04, ACM, New York, NY, USA, 2004, pp. 179–188 (2004).
[15] D. Liben-Nowell, J. Novak, R. Kumar, P. Raghavan, A. Tomkins, Geographic routing in social networks, Proceedings of the National Academy of Sciences of the United States of America 102 (33) (2005) 11623–11628 (2005).
[16] S. Athanassopoulos, C. Kaklamanis, I. Laftsidis, E. Papaioannou, An experimental study of greedy routing algorithms, in: High Performance Computing and Simulation (HPCS), 2010 International Conference on, 2010, pp. 150–156 (June 2010).
[17] M. Mitzenmacher, E. Upfal, Probability and Computing: Randomized Algorithms and Probabilistic Analysis, Cambridge University Press, New York, NY, USA, 2005 (2005).
[18] M. D. Vose, A linear algorithm for generating random numbers with a given distribution, IEEE Trans. Softw. Eng. 17 (9) (1991) 972–975 (Sep. 1991). doi:10.1109/32.92917.
[19] J. von Neumann, Various techniques used in connection with random digits, Nat. Bureau Standards 12 (1951) 36–38 (1951).
[20] F. Mathieu, Kleinberg's Grid Reloaded, in: 20th International Conference on Principles of Distributed Systems (OPODIS 2016), Madrid, Spain, 2016 (Dec. 2016). doi:10.4230/LIPIcs.OPODIS.2016.21.
URL https://hal.inria.fr/hal-01417096
[21] A. J. Walker, An efficient method for generating discrete random variables with general distributions, ACM Trans. Math. Softw. 3 (3) (1977) 253–256 (Sep. 1977). doi:10.1145/355744.355749.
[22] W. Hörmann, G. Derflinger, Rejection-inversion to generate variates from monotone discrete distributions, ACM Trans. Model. Comput. Simul. 6 (3) (1996) 169–184 (Jul. 1996). doi:10.1145/235025.235029.
[23] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Minimization or maximization of functions, in: Numerical Recipes: The Art of Scientific Computing, 3rd Edition, Cambridge University Press, New York, NY, USA, 2007, Ch. 10 (2007).
[24] R. I. M. Dunbar, Neocortex size as a constraint on group size in primates, Journal of Human Evolution 22 (6) (1992) 469–493 (Jun. 1992).
[25] I. de Sola Pool, M. Kochen, Contacts and influence, Social Networks 1 (1978) 5–51 (1978).
[26] T. H. McCormick, M. J. Salganik, T. Zheng, How many people do you know?: Efficiently estimating personal network size, Journal of the American Statistical Association 105 (489) (2010) 59–70 (2010).
[27] B. Wellman, Is Dunbar's number up?, British Journal of Psychology (2011).