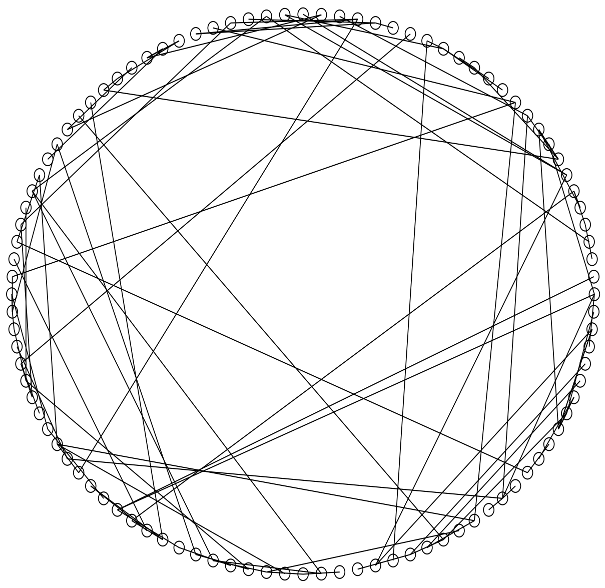


Switching for a Small World

Vilhelm Verendel

Master's Thesis in Complex Adaptive Systems

Division of Computing Science
Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden May 2007

Abstract

Agents in *small world* networks are separated from others by short chains of edges, and they also have the ability to find these paths. Kleinberg (2000) relates a decentralized ability, finding short paths using greedy routing between nodes in a lattice, to a unique distribution on shortcut edges. Sandberg (2005) presents a method that can take a graph from this class without position information and assign positions to make it navigable again. The method is a Markov Chain Monte-Carlo method using the Metropolis-Hastings algorithm where nodes switch positions in a base lattice.

In this thesis we study ways to speed up the method by Sandberg to make graphs from the Kleinberg model navigable. First, we study how to compare different switching methods. Second, we study different ways to propose which positions to switch. Third, we attempt to increase the number of nodes involved in a switch. The new selection kernels are also fit for distributed implementation like in Sandbergs work.

The main results are three new selection kernels, each involving two switching nodes, that improve the time for the algorithm to reach good performance. Involving more nodes in switches also seems to give certain improvement, but switching two nodes with a bias seems more efficient than including an additional node uniformly at random.

Acknowledgements

I would like to thank my advisors Oskar Sandberg and Devdatt Dubhashi for guidance, help and review. Also, thanks to Libertad Tansini and Nina Tahmasebi for their help and discussions.

Contents

1 Introduction and previous work

The *small world* concept dates back over several decades, but the mathematical models done continue to find new applications today. In essence a small world network is one where any two persons are connected to each other through short chains of edges, and graphs of such networks have a small diameter. This thesis is concerned with one method of applying such models to networks without explicit position information, in order to make them more efficient for routing.

A short summary of some background and models attempting to explain small world phenomena are covered here, before we take off from the work by Sandberg [9]. To present the background to this thesis we will only do this in a concise manner, more detailed summaries of the small world ideas can be found in [1],[11],[9]. In section 1.6 we discuss the goals of this thesis.

1.1 Milgrams work and "six degrees"

The work by Milgram [8] in the area of social psychology was a breakthrough in quantifying the small-world property of networks: the findings that have also been mythically called "six degrees of separation". The experiments carried out in North America explored how closely any two persons were connected to each other through the social network of friendships. The way to measure this was to count the number of steps a message would need to take from a random source, through a chain of friends, to a destination. One typical experiment would go like this. A person was selected as the destination of a message (e.g. a letter). Then a large number of people were selected geographically far away[1] and were given the task of passing on the message to the destination. The message could only be passed on between friends, and not directly to the destination, so that the messages had to pass through chains of friends. The results of the experiments showed that often the number of steps needed is surprisingly small. Some interpretations claim the average number was six, but the exact number has been under debate (see e.g. Kleinfeld [7]).

An idea slightly different from that people are on average closely connected as shown in Milgrams empirical work, is that people seemingly were able to construct short paths from what seemed to be *local* information. When people (or *agents* in a network) are able to do this efficiently, we call the network *navigable*. Local information means that messages were only passed on between friends and that people were usually not able to foresee the exact path that a message would take further on. With full knowledge of the graph structure (ie. knowing the full network of social friendships), one may expect that it's possible to construct a short path to a target. But when restricting the decision of an agent to only use local data, the ability to paths through the network is more surprising. The strategy mostly used to pass on the message was to send it on to the friend that seemed closest to the destination, a distance measure that may vary in social networks (and which does not necessarily have to be geographical distance). This implies that some variant of *greedy routing* (always passing on a message to the neighbor closest to the destination) may be used as a method to find short paths in social networks. This is the basis for the decentralized view that is taken for navigation and search in small worlds. Nodes making decisions with only local data is the scenario that we consider in this report.

[1]The measure of distance in social networks can be expected to be much more complicated than geographical distance

1.2 Random graphs

One model that shows that short paths *can* exist in large networks is from the theory of random graphs. The model is often denoted as $G(n, p)$. Such graphs start out with a set of n nodes, and for each pair of nodes we connect them with an edge with probability p (p is constant for a given graph). Thus $p = 0$ is an edgeless graph, and $p = 1$ is a fully connected graph. The interesting things happen when p varies in between. Phenomena such as the formation of large connected components, specific structures and short paths have been studied and shown to occur typically at threshold values (in the limit case). However, social networks are not believed to grow in this fashion. Later and recent models have focused on explaining the growth of small worlds with respect to the coordinates of a *base lattice*, lattice models where edges are added with some relation to the distance they cover between nodes in the lattice.

1.3 Watts-Strogatz model

The model of Watts and Strogatz (1998) [10] may be in some sense closer to how real small worlds seem to be arranged. It is based on that people usually are more likely to have friends nearby, but that most still have some friends that live far away. Even if the previous models had the property of typical short path between any two nodes, some properties were missing in the random graph approach that were present in the Watts-Strogatz model. One example is that social networks often have high *clustering*; here we mean that any two nodes connected to a given node are much more likely to know each other than *any* two nodes in the graph. The new model demonstrated also this property along to the short paths, by growing the edges based on a distance.

Generating a Watts-Strogatz graph means to start out with a base lattice (discrete, and with periodic boundary conditions). Each node starts with an assigned position in the lattice, then a fixed number of edges are added to each node in two steps. In the first step edges are added between each node and its nearest neighbors with respect to the lattice (within a small distance in the lattice), thus the clustering can be shown to be high (since neighborhoods "overlap"). The second step consists of rewiring some of the edges added in the first step.

The parameter, α, denotes the fraction of edges that are subject to random rewiring from the ordered lattice resulting from the first step. In this model the rewired edges get new targets within the lattice by uniform random assignment, thus creating *shortcuts* that cover large distances in the underlying lattice. Using the shortcuts it becomes possible to reach parts of the graph that are far away in the lattice in a small number of steps. Later models take a different approach to the uniform selection of the rewiring target.

By varying α, it became possible to study the transition from a very locally ordered/clustered graph ($\alpha = 0$) into a graph with all edges are randomly rewired ($\alpha = 1$). It was found out that when one introduces just a small fraction of random rewiring, the average shortest path length and network diameter drops rapidly, still keeping the clustering high. See [1] for more details and simulations. These two properties are commonly associated with small-world networks. But this model did not explain *how* the short paths could be used.

1.4 Kleinberg model

Kleinberg (2000) [6] generalised the model by Watts and Strogatz and at the same time approached a different question regarding the small-world phenomenon: aside from the *existence* of short average paths - in which situations can nodes *find* and construct short paths given only local information in each step? Kleinbergs model starts from a base graph with local connections to the nearest-neighbors. Then each node gets an *additional* shortcut by a specific random process (discussed below). For navigation purposes, the local information for each node was restricted to the node's own position (in the lattice), the positions of neighbors, and the position of the destination for a message. What kind of structure would be needed for nodes to route efficiently? Kleinberg first showed that the Watts-Strogatz model did not allow for efficient greedy routing, but related an ability to do this to a specific distribution on shortcut lengths taken with respect to the lattice (a requirement on the random process for adding the shortcuts).

The family of models discussed in Kleinbergs work were the following: given a distance measure $d(x, y)$ beween each pair of nodes x and y in a k-dimensional lattice, the probability for a shortcut edge to exist between two nodes is proportional to the distance covered by it depending on a parameter α. The probability of a long-range shortcut from x to y, would then be proportional to $d(x, y)^{-\alpha}$.

Kleinberg showed that there is a unique distribution within this family that allows for efficient greedy routing (in each step, a node forwards a message to the neighbor closest to the target with respect to distance in the lattice), and it is for $\alpha = k$ where k is the dimension of the lattice. The probability for a shortcut edge from x taking y as target needs to be

$$p(x \leftrightarrow y) = \frac{d(x, y)^{-k}}{H_k(n)} \tag{1}$$

where $H_k(n)$ is a normalization constant.

For $\alpha = 0$, this would correspond to selecting uniform random selection of shortcut targets, roughly like the Watts-Strogatz model (but this model adds shortcuts instead of rewiring existing connections). When α increases, it becomes more probable for a shortcut edge to cover shorter distance (take targets closer to the source node, with respect to distance in the lattice). k is the critical value of α where we have a distribution balancing the proportion of shortcut distances that is needed for greedy routing. A graph where shortcuts have been generated by applying this specific distribution is shown to allow for greedy routing with on average $O(log^2(n))$.

1.5 Sandberg: Distributed Routing in a Small World

Sandberg (2005) [9] presents a method to take a graph where the shortcut edges have been generated as in the Kleinberg model, but that comes without position information (no lattice), and make it navigable. The method assigns positions to the nodes in a new base lattice in order to make it navigable. This is done by trying to match the Kleinberg distribution given in Equation 1 for the assigned positions, so that the distance covered by the edges reflect this. The way to make such a topology navigable again is to make an estimation for the configuration of positions that is good enough to allow for efficient greedy routing.

To make an estimation, the Kleinberg model is seen as the process to generate a set of shortcut edges, and the graph that we get is an instance. The algorithm is from the field of Markov Chain Monte-Carlo and is on a form so that it allows for a decentralized and distributed implementation in a graph. Thus it may be used to create navigable overlay networks in a distributed way in this kind of topology.

By considering all possible configurations of positions for nodes in this graph, a Markov chain is constructed on the set of all possible configurations. This corresponds to all the possible ways to assign N different numerical positions to N nodes, so the state space of the chain is of size $N!$. One *configuration* is defined as a function ϕ from the set of nodes V to the integer position assigned to each node in a k-dimensional base lattice. Running the resulting Markov chain for a number of steps improves the efficiency of greedy routing for graphs with shortcut edges from the ideal Kleinberg distribution from Equation 1, partially also for a more general class of (random) graphs. The methods seems to fit well for applications in anonymous networks, and more generally in overlay networks, in situations where the only contacts are between peers that trust each other [2]. Approaching the Kleinberg model is done through a sequence of *position switches* which define the transition matrix of the Markov chain. This is also known as the *selection kernel* in the Metropolis-Hastings algorithm, which we will be concerned with in much of this work. The rest of this section briefly describes the derivations of the distributed algorithm in [9].

1.5.1 Probability model

Assume that we get a graph (V, E) with nodes V and edges E, and that the configuration ϕ has assigned positions to each node of V in a k-dimensional lattice. If the edges have been generated by the Kleinberg model then the probability of the particular set of edges E depends on how much distance the edges cover in the configuration. Adding of edges occur independently in the Kleinberg model, so we get the conditional distribution

$$P(E|\phi) = \prod_{i=1}^{m} \frac{1}{d(\phi(x_i), \phi(y_i))^k H_G} \tag{2}$$

where $d(\phi(x_i), \phi(y_i))$ is the distance covered by edge i in the lattice, and H_G is a normalizing constant depending on all possible assignment of m edges with this configuration.

In our problem the initial ϕ used when applying the Kleinberg distribution is what we have no previous information on, and which we need a good estimate for. The task is to estimate the configuration ϕ in order to generate an embedding $\hat{\phi}$ of the nodes V into a k-dimensional lattice, so that the distribution of distances will fit the Kleinberg model. The approach is taken as follows.

1.5.2 Bayesian approach

Instead of trying to directly compute a configuration that maximizes Equation 2, which has been shown to be NP-complete [3], Sandberg takes a Bayesian approach. In a case where we only get the vertices and edges of a graph, known to have been generated using the Kleinberg distribution in a base lattice, we can view the adding of edges as an experiment and the edges E as the outcome. This leads us to consider a distribution on all the possible configurations, instead of trying to recover the most likely configuration. This becomes the parameter of the model and is given as the *a posteriori* distribution on the configuration

$$P(\phi|E) = \frac{P(E|\phi)P(\phi)}{P(E)} \tag{3}$$

The idea is to draw a sample from this distribution, and then assign it to the nodes V. This distribution will give a bias for configurations that cover shorter lengths with respect to the base lattice. The approach to sample the posterior distribution is done as follows.

1.5.3 Metropolis-Hastings

Metropolis-Hastings is a method in the field of Markov Chain Monte-Carlo [4]. When one wants to draw samples from a distribution one usual approach is to integrate over the distribution function. When this is complicated and the size of the problem is large, the Metropolis-Hastings method can be seen as one way to approximately sample from the distribution. The idea of the method is to construct a Markov chain with the distribution that one wants to sample from as a *stationary distribution* for the chain. If the chain is made irreducible and aperiodic (ergodic) then the chain will converge towards the stationary distribution from any starting state (or distribution). In the limit case this holds exactly, in practice and finite times the number of iterations needed depends on the requirements of the problem (see e.g [4], [5] for good presentations).

The Metropolis-Hastings algorithm works with two components, α and β, that are used control the transition between states of the Markov chain. One result from this that fits well with simulating the chain is that there is no need to explicitly generate the whole transition matrix. The *selection kernel* $\alpha(X_i, \cdot)$ is a distribution on the states that may be selected next from any state X_i, and this distribution is designed with some freedom depending on the application. In each step of the chain, a proposed state X_{i+1} is drawn depending on a current state X_i with probability $\alpha(X_i, X_{i+1})$, and accepted with probability $\beta(X_i, X_{i+1})$ given by

$$\beta(X_i, X_{i+1}) = min\left(1, \frac{\pi(X_{i+1})\alpha(X_{i+1}, X_i)}{\pi(X_i)\alpha(X_i, X_{i+1})}\right) \tag{4}$$

where π is the stationary distribution that we want the chain to converge to. If the step is not accepted then the chain stays in the current state.

The trick of the algorithm is that the use of α and β make the chain *reversible*; that is, $\forall j, k: \ \pi(X_j)P_{j,k} = \pi(X_k)P_{k,j}$ where P is the transition matrix for the chain. This is also the property that gives π as the stationary distribution (for details, see Appendix A). The general form of the Metropolis-Hastings is now presented: in each step i do

1. Propose a new state X_{i+1} for the chain following $\alpha(X_i, X_{i+1})$.
2. Accept the new state with $\beta(X_i, X_{i+1})$

1.5.4 Metropolis-Hastings on the set of configurations

Now what is left is to create a Markov chain with stationary distribution given by Equation 3, that is, the probability of a given configuration of positions being used to generate the observed set of edges. Each state i is thus a configuration ϕ_i. In [9] $\alpha(\phi_s, \phi_r) = \alpha(\phi_r, \phi_s)$ by symmetric proposals. Evaluating $\beta(\phi_s, \phi_r)$ is done as follows

$$\begin{aligned}\beta(\phi_s,\phi_r) &= min\left(1, \frac{P(\phi_r|E)\alpha(\phi_r,\phi_s)}{P(\phi_s|E)\alpha(\phi_s,\phi_r)}\right)\\ &= min\left(1, \frac{P(E|\phi_r)}{P(E|\phi_s)}\right)\\ &= min\left(1, \prod_{i=1}^{m} \frac{d(\phi_s(x_i),\phi_s(y_i))^k}{d(\phi_r(x_i),\phi_r(y_i))^k}\right) \qquad (5)\end{aligned}$$

where we have used the symmetric selection kernels and uniform a priori assumptions for cancellation. The expression holds for β given any symmetric selection kernel. What does a good symmetric selection kernel look like? The approach taken is to let $\alpha(\phi_s,\phi_r)$ denote a position switch between two nodes, drawn uniformly random from the graph. This will define each step of the Markov chain.

What differs between ϕ_s and ϕ_r is thus only the positions of two nodes involved in a switch. A state ϕ_r is the x,y-switch of ϕ_s if $\phi_s(x) = \phi_r(y)$ and $\phi_s(y) = \phi_r(x)$, and $\forall_{z\neq x,y}.\phi_s(z) = \phi_r(z)$. As a consequence of this, the only thing that differs is the lengths covered by the edges connected to the two nodes involved in the switch. Now this is used to simplify Equation 5 to

$$\beta(\phi_s,\phi_r) = min\left(1, \prod_{i\in E(x\vee y)} \frac{d(\phi_s(x_i),\phi_s(y_i))^k}{d(\phi_r(x_i),\phi_r(y_i))^k}\right) \qquad (6)$$

where $E(x \vee y)$ is the set of edges connected to x or y. Evaluating β can thus be done by only using information that is local to the both nodes x and y involved in the switch. This gives rise to a distributed implementation, if two nodes can be selected approximately uniformly at random. This gives a selection kernel equal to

$$\alpha(\phi_s,\phi_r) = \begin{cases} 2/(n(n-1)) & \text{if } \phi_r \text{ is } x,y\text{-switch of } \phi_s \\ 0 & \text{otherwise}\end{cases}$$

The Markov chain is thus defined by Equations 6 and 7 as

$$P\phi_s,\phi_r = \alpha(\phi_s,\phi_r)\beta(\phi_s,\phi_r)$$

for all states ϕ_s, ϕ_r that are an x,y-switch of each other.

1.5.5 Summary

In summary the Metropolis-Hastings algorithm for trying to embed the nodes of a graph with no position information to fit a Kleinberg distribution is done as follows. Initialize the chain (initial state) with any random configuration ϕ_0. In each step i do

1. Propose a new state for the chain ϕ_{i+1} following $\alpha(\phi_i, \phi_{i+1})$
2. Accept ϕ_{i+1} as new state with probability $\beta(\phi_i,\phi_{i+1})$, otherwise stay in ϕ_i

This method is demonstrated to work well on graphs generated as the *ideal Kleinberg model* (where all edges are added with the Kleinberg distribution) without no a priori assigned positions. It is also studied for graphs with only the shortcut edges. That is, no edges to nodes closest in the lattice as in the Watts-Strogatz model. Being able to pass on messages to *any* node that is closest with respect to the lattice distance is typically lacking in overlay networks, and this also takes away the possibility for a greedy route to always strictly approach the target in the base lattice - and *dead ends* (where a node has no neighbor closer to the target than itself) have to be considered. This removes one piece of Kleinbergs proof but can be handled in practice by backtracking or taking on a different routing strategy. One easy approach is to continue a route to the best neighbor possible.

The results are shown to make also these graphs (with only shortcuts) efficiently navigable, and here we will partially simulate them in section 2. It is also interesting to see that the algorithms also makes some improvements on the navigability of random graphs.

We can see that the expression for Equation 5 will prefer configurations where the edges E span less distance in the base lattice, and thus the algorithm can also be seen as a minimization procedure. In terms of the *Simulated Annealing* framework, we can see that the *energy function* to minimize would be the log sum of edge distances (and $T = 1$ for the inverse temperature).

1.6 Goal and results of this thesis

The work in this thesis starts off from the work by Sandberg described in section 1.5, and the goal of this work is to speed up the algorithm by modifications to the selection methods (where the freedom lies in the selection kernel of the Metropolis-Hastings algorithm). We will use the model where a graph only comes with the shortcut edges and no “local” connections in the lattice, to focus on the case of making overlays navigable.

The rest of this work is arranged as follows

1. In section 2 we study the performance of the Metropolis-Hastings algorithm discussed in section 1.5 and conclude that there can be room for improvement. We also describe how we typically measure routing performance.

2. In section 3 we state and simulate two stop criteria to be used for comparing the efficiency of different selection methods further on. The first criteria relates to how fast the method improves the actual solution, the second criteria can be used when we know the ideal performance.

3. In section 4 we propose three different selection methods for improving the speed of which the algorithm has effect, we call this approach *local selection* since they relate to the positions that graph neighbors take in the lattice. This is done by still considering two nodes in each switch, but trying to let nodes choose smarter where to end up on the lattice.

4. In section 5 we try a different approach: not selecting switching peers with a goal, but instead by including more peers (more configurations available in each step) for the switch in each step of the Markov chain. We then try to compare the methods against each other.

The main result is that that our three local selection methods reach the performance at the stop criteria faster than selecting nodes uniformly random does. Involving more nodes also indicates that there is some improvement in doing this if one considers steps of the chain, but that a directed switch gives more improvement than only increasing the number of nodes involved.

2 Measuring performance and the current algorithm

In this section we discuss how we will measure navigability properties of a graph later on, and we discuss the current performance of the Metropolis-Hastings approach (by uniformly selecting two nodes for switching). We also make some numerical experiments to show the effect of the algorithm.

2.1 Measuring performance

For a network of size N, we start with the nodes placed out in a 1-dimensional lattice, and to each node we add $3log_2N$ shortcut edges according to the ideal Kleinberg model (Equation 1). The degree is somewhat arbitrary to the extent that it is high enough to keep the number of dead ends low so that we can see the effect of greedy routing, but also chosen for comparison purposes because it was used in [9].

When a graph has been generated, we first measure the routing performance. Then the positions assigned to the nodes get randomly shuffled as a means to forget the initial configuration (this is the same as to start from a random state of the chain). Now we can measure how well we can *recover* the positions used to generate the Kleinberg model by repeating the experiments after the algorithm has been run for a number of steps. Each greedy route is made between two nodes selected uniformly random from the graph, and is given maximally log_2^2N steps to succeed reaching the target, before terminating it and counting it as unsuccessful. Performance is evaluated over 10^5 different routes. We handle dead ends in greedy routing by continuing with the *best possible* neighbor, instead of stopping, if the current node has no neighbor closer to the target than itself. Three factors are in focus of our performance measurements: *average length* (number of steps required for successful route) in greedy routes, *success rate* and the *energy function* (as defined in section 1.5.5).

2.2 Metropolis-Hastings performance

We created graphs of different sizes as described above and evaluate the Metropolis-Hastings algorithm as in section 1.5 by periodically evaluating the performance. The period is defined in *rounds*, where one round is N steps (proposed position switches) of the chain. One round then corresponds to on average one initiated switch per node in the graph. For this experiment we evaluated with the period of 50 rounds.

2.3 Results and evaluation

We can see the results in figures 1-4. As the network size grows we can expect performance on the initial (shuffled) positions to become only worse. The improvement also seems to have largest effect in the initial $100N$ steps of the chain or so. After this initial phase the *acceptance rate* (switches that get accepted by Equation 6) quickly falls to a low level when seen on longer time-scales and the improvement goes slowly.

But while the routing properties only improve slowly we can see there is also room for further minimizing the energy function, also happening with a slow rate in comparison. This could imply that the algorithm often is stuck in local minima. The low acceptance rate implies that it takes a long time to propose (or find) states that minimize the distances covered by the edges. One way to make the algorithm gain performance faster may be either to make better switches from the beginning, or to find the switches more efficiently (increase acceptance rate). There may be room for improvement.

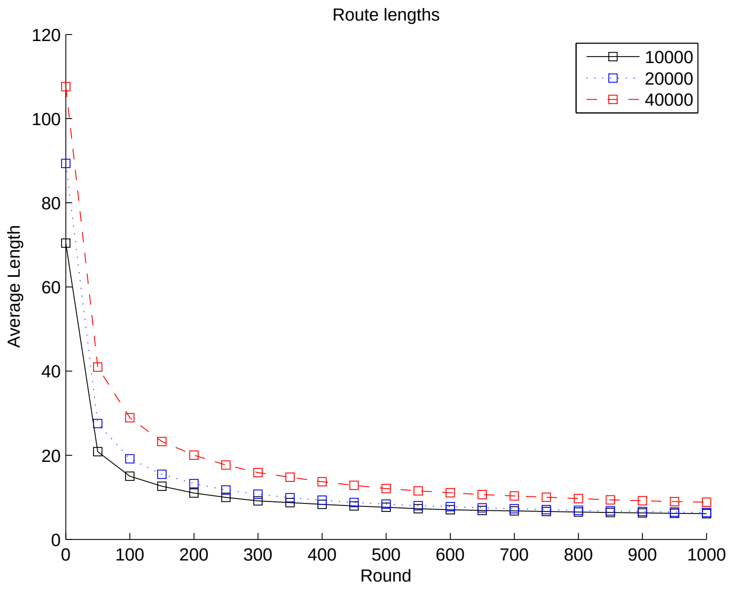


Figure 1: Recovering the small world from randomly shuffled positions. 1000 rounds. Averaged over 10 rounds.

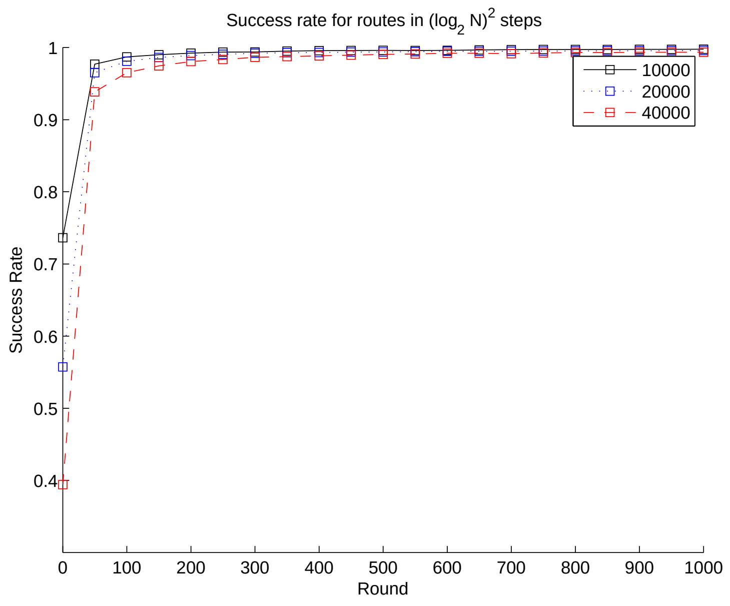


Figure 2: Recovering the small world from randomly shuffled positions. 1000 rounds. Averaged over 10 rounds.

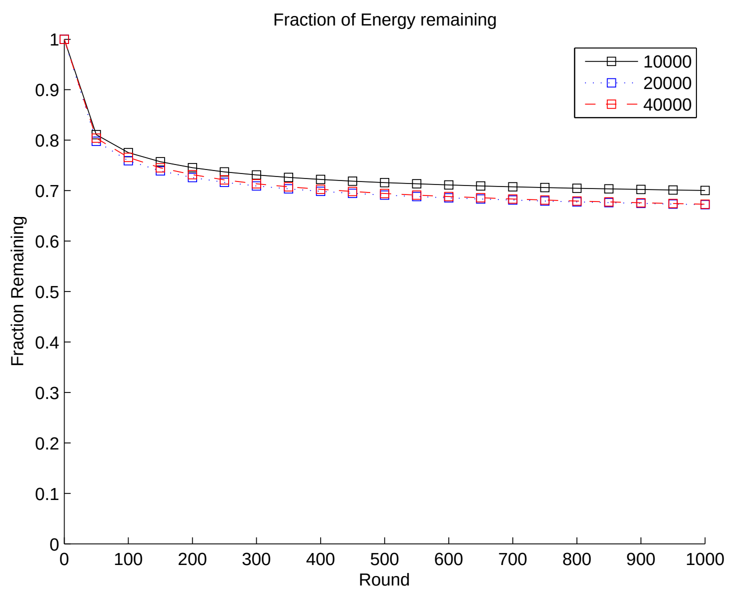


Figure 3: Effect on the energy function (log sum of edge distances). Averaged over 10 rounds.

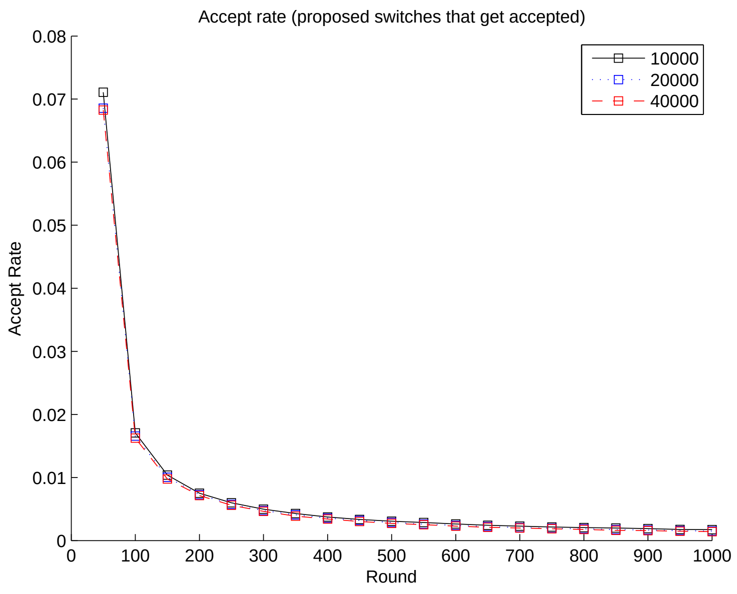


Figure 4: Fraction of proposals that got accepted. Averaged over 10 rounds.

3 Convergence rate of Metropolis-Hastings switching

The results discussed in section 2.3 may indicate that it is possible to improve the algorithm by changing the way to find switches. Before we can go on and try improving performance, we will need to decide for a method to compare the efficiency between different switching strategies. In this section we discuss the role of the selection kernel, short-term vs long-term results of applying the Metropolis-Hastings algorithm (with different selection methods), and propose two different criteria for comparing the performance of selection methods. This is in contrast to just studying time evolution of the performance as in section 2, because we do not know where to stop if we just study what has happened after a certain number of steps. The algorithm can run forever, but when should we evaluate it? What we want to have is a way to know when we have achieved a goal with the algorithm and stop, to see how well other methods work against the same goal. The first stop criteria depends on the rate that the algorithm has of improvement (on a given graph), the second depends on getting close enough to the performance in the ideal model.

3.1 Role of the proposal distribution

Since the only factors that change the navigability properties of the graph are the position switches (proposed chain transitions *that get accepted*), the time of increase in efficiency may depend on the time spent (number of iterations) finding these. The way to control the mixing rate in Metropolis-Hastings is to modify the proposal distribution. As previously stated this is the property of the selection kernel that assigns, to each state, a distribution on the set of states that may be proposed next from that state. More generally, this represents which states are counted as neighbors to each other in the state graph of the Markov chain. As being shown in Appendix A, we are free to design this relationship within bounds that allow for a lot of freedom.

If the selection kernel can be designed so that it proposes states with a larger probability in the stationary distribution, then the improvement rate may improve. One of the most straight-forward kernels is the *symmetric selection* kernel, here where proposed position switches are selected uniformly random from the set of possible changes. Proposing with a bias, e.g. for the stationary distribution, may replace this.

The long-term results of the Metropolis-Hastings approach, proposing switches between nodes uniformly random, have already been simulated well in [9], partly also here in section 2. What we try to do here is to characterize how the algorithm improves navigability. Remember that in the asymptotic case (when the number of steps go to ∞) we will obtain the sought distribution with any reasonable (irreducible and aperiodic) selection kernel, starting from any state. But what we also need to take into account is when the selection kernel may be differing in practice on a shorter time-scale. The idea from now on will be to run the algorithm until we get results that are *good enough* in practice. This view also lets us compare the improvement speed of different methods by simulation.

3.2 Time to slow mixing

To begin with, we try to evaluate the convergence rate of the algorithm by studying how many iterations are needed until we see clear signs that the main improvement has been done. The bounds on this are arbitrary; either one could compare the results after a very long time, the alternative is to run the chain until the improvement has slowed done and a phase of slow improvement begins.

What quantities we could expect the algorithm to improve is at least three-fold. We study the average length and success rate just as described in section 2.1. For each of these 3 quantities, we want to study how fast the algorithm will effect them and how long it takes before it (roughly) stops improving. This leads us to study the convergence rate; that is the number of iterations needed by the algorithm before the main improvement has been done. Now we present two different criteria for this.

3.3 Bounding on improvement rate

With this stop criteria, we stop the algorithm when the following two criteria are met (for a graph with N nodes):

1. The success rate does not increase more than a fraction k during the last M *proposed* rounds (times N steps) of the chain
2. Same as above, but for the average path length

Other bounds could be put on things such as the energy function, or the variance, quantiles etc. This indicates that the Metropolis Hastings algorithm no longer makes large improvements on performance. We typically use $k = 0.02$ as a numerical trade-off between sampling accuracy and computation time, over at least 100 *rounds* of the Markov chain. Note that we scale the interval in number of chain steps as the network size grows, using rounds seems more natural especially from the view of a distributed algorithm. With this interval we evaluate the change of the graph performance.

3.4 Bounding on ideal performance

Another way to design a rule for comparing performance is to stop the algorithm when performance is within a bound of the performance in the initial (ideal) model. We cannot expect to recover the optimal embedding due to the complex search problem, but in practice it is enough to get a model close to the potentially best configuration. That is after all what we are after from the beginning when trying to sample the posterior distribution.

3.5 Method

A set of disconnected graphs with different sizes were created, edges were then added from the Kleinberg distribution as before. Before we shuffle the positions of the nodes, we measure the initial performance on average length and success rates (L_{ideal}, sr_{ideal}). Then the node positions were randomly shuffled. Each graph is given to the Metropolis-Hastings algorithm and once for each stop criteria. The evaluation was done by attempting to route 10^5 times between two uniformly random nodes. If the network met the convergence criteria, we would terminate the algorithm for this graph.

The bound on improvement rate parameter k was set to 0.02 over 100 rounds. The interval at where the improvement is measured and evaluated were set as a tradeoff between accuracy of change and accuracy of comparison (between different network sizes). The bound on the ideal model was run until the following criteria were met

1. $sr > 0.9sr_{ideal}$

2. $L < 2L_{ideal}$

which means that we stop when the performance of average length is within twice the length of the ideal model, and that we have at least 90% of the ideal success rate. For all our simulations with the ideal-bound criteria the bound on average length took longest to achieve.

3.6 Results and evaluation

Results from the simulations are shown in figures 5-8. Figure 5 shows us the gap between the ideal model and the shuffled case (the situation where we have not fitted the graph well with geographical information). This is what we want the algorithm to reduce well in a number of steps as small as possible. For smaller graphs we can see (fig 8) that the bound on process improvement takes more steps before it stops, and that performance is better when stopping (fig 6). The explanation may be that the smaller the graph, a higher fraction of switch pairs may be proposed on average (the number of steps in a round grows linearly with size, number of potential pairwise switches quadratically) – leading to an easier way to find good switches to keep up an improvement rate.

However, we can see that the bound on ideal performance becomes the most restrictive criteria when we scale the size of the graph. We also note that the number of steps to reach this criteria clearly grows faster than linearly (since number of steps in a round scales with the network size).

We can thus conclude that the bound on ideal performance is the most is best at guaranteeing a bounded satisfactory solution (this is not surprising since the bound is explicit on the performance). The bound on improvement may still tell us something since it is less restrictive for larger sizes and we reach it faster, which would tell us about when agents stop seeing large improvements in such a network. It also has an advantage of using it for instances where we dont have access to the ideal performance, such as the experiments on random graphs done by Sandberg [9].

The conclusion is that it may be useful to keep both the criteria for later simulations since the may show slightly different things about the results.

3.7 Raised questions

In summary, there are several questions of interest involving our two stop criteria:

1. How does the network size affect the number of Metropolis-Hastings iterations needed? This seems to vary with different criteria, can we make it vary with different switching methods? The same question holds for the number of actual (accepted) switches that is needed.

2. What would be the best way to measure improvement? The usage of different stop criteria may have impact on how different methods are evaluated. Evaluation on several different criteria may also be used to mix methods.

3. Can we characterize the “good” switches that get accepted with the uniform selection method? It may be used for improving acceptance rate. Is there a selection kernel that is more efficient?

We will try to tackle these questions partly in the next section.

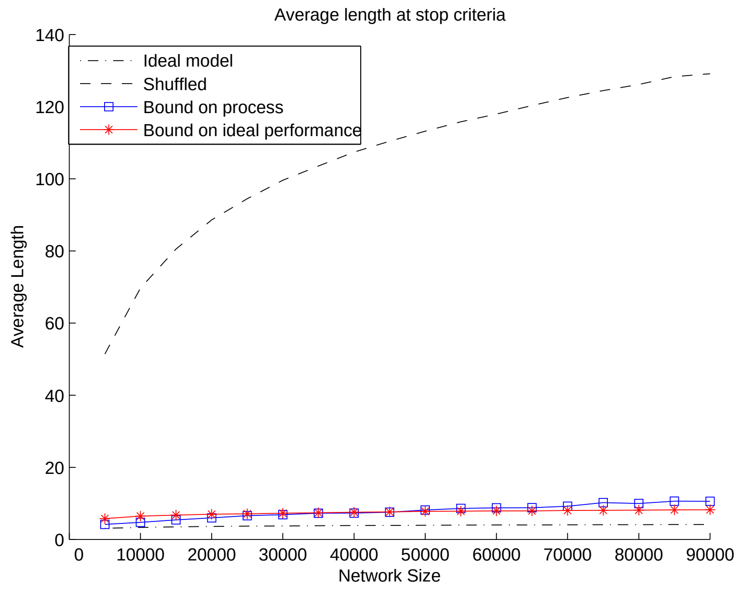


Figure 5: Avgerage lengths when stop criteria reached, compared against shuffled performance. Averaged over 10 runs.

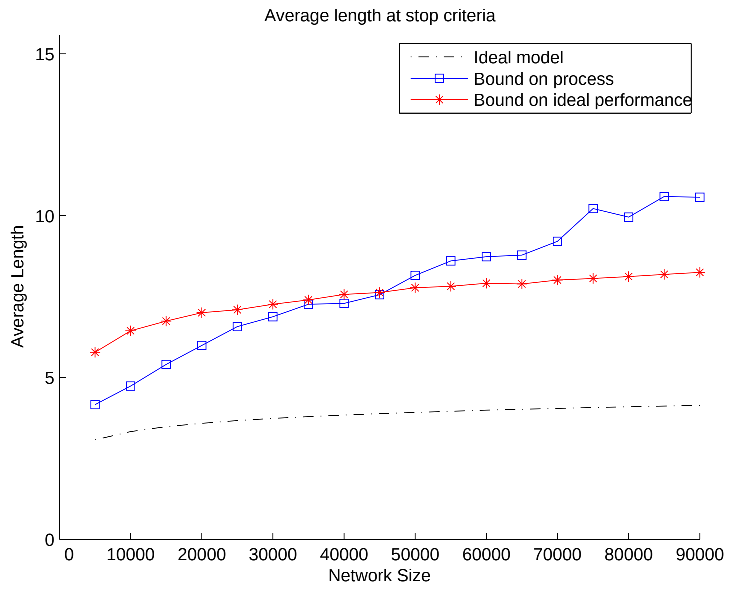


Figure 6: Average lengths when stop criteria reached, compared against each other. Averaged over 10 runs.

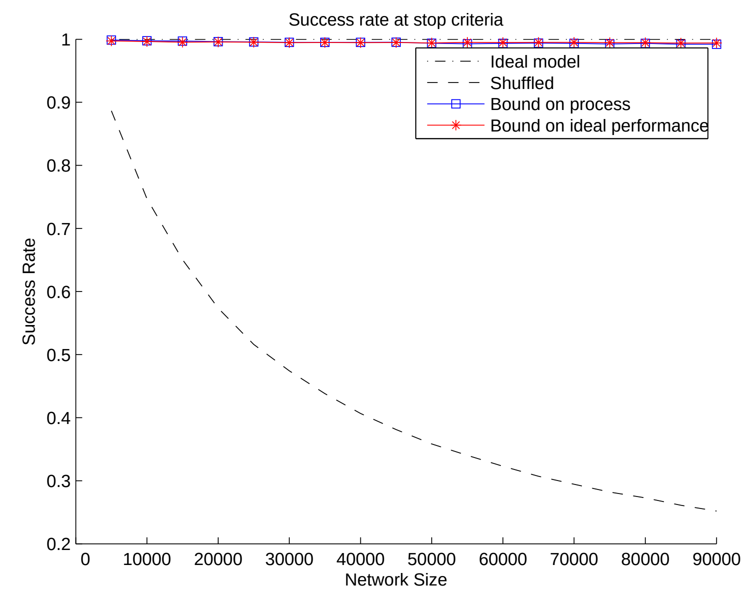


Figure 7: Success rate when stop criteria reached. Averaged over 10 runs.

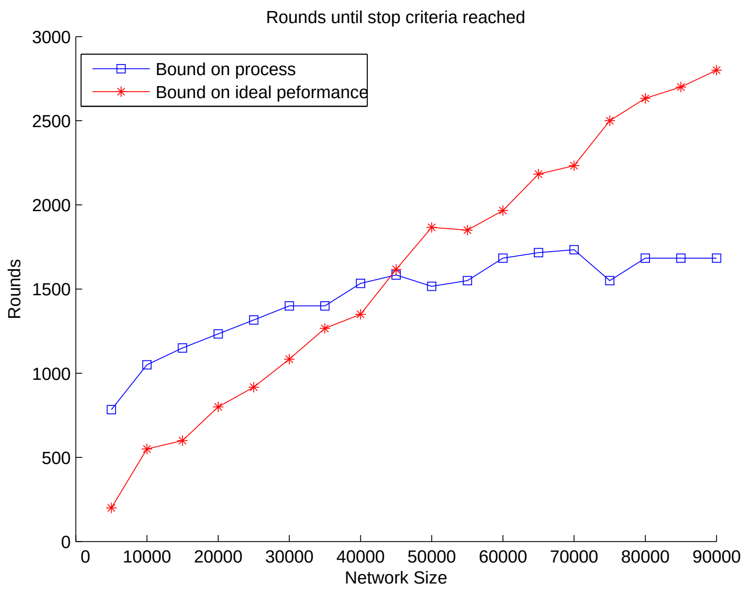


Figure 8: Studying how rounds to stop criteria depend on network size. Averaged over 10 runs.

4 Local selection

As we have seen, the Metropolis-Hastings algorithm will give bias for configurations where edges cover less distance with respect to the base lattice. This can also be formulated as that the position of a node is preferred to other positions when it is close to the positions taken by neighbors in the graph. Our strategy will be to propose position switches that direct the Markov chain this way. In this section we experiment with different ways of doing this, and we will be working with information *local* to the nodes which may fit well with distributed implementations.

4.1 Approach for improvement

Since we want to improve the number of steps needed for the algorithm to reach a good result, a first attempt is to propose those switches that have a high probability of being accepted (that have a higher probability in the stationary distribution π). We see from Equation 6 that a direct way to improve the accept rate is to make the edges in the proposed configuration span less distance in the lattice than the current configuration.

Since the selection kernel can be almost arbitrarily selected, but still have our desired stationary distribution (at least in the theoretical asymptotic case, see [4] and Appendix A), we can see what decisions can be taken for switches that depend on information local to their neighborhoods. This makes it possible to design strategies that work on only local information at each node, thus making a decentralized implementation an option as before.

One idea is that a node would propose switching to a position to minimize the energy function. This will be a direct attempt to minimize the log sum (or maximize Equation 2) of distances to its neighbors, thus reducing the energy function by only looking at the local neighborhood. We will, however, not restrict ourselves only to minimizing log sums but study other ways as well. We will use the following notation when doing a position switch between x and y:

- x: a random (uniformly selected) node in the graph
- x^*: a node holding a position that minimizes the distances to the neighbors of x, according to some [2] minimization method. There may be several such positions, $x^* \in \{x' \in V : \text{s.t.}\phi(x') \text{ minimizes distances to neighbors of } x\}$
- y: a node that gets selected by x for proposed position switch, based on the local information of x
- ϕ is the configuration as before, that for each node $x \in V$ assigns it a position in the lattice
- x, y-switch: from any configuration ϕ this is the set of configurations where any ϕ' we have $\phi(x) = \phi'(y)$, $\phi(y) = \phi'(x)$, and $\forall_{z \neq x,y}.\phi(z) = \phi'(z)$.
- $d_{\phi_i}(x, y)$ is the distance in the base lattice between any two nodes x and y under configuration ϕ_i (eg $d_{\phi_i}(x, y) = |\phi_i(x) - \phi_i(y)|$)
- $\Phi(a, b, \sigma^2)$ denotes the distribution function for a Gaussian centered on b

[2] Typically the log sum, but we also attempt other methods

Note that the proposed new position $\phi(y)$ for x does not have to correspond exactly to $\phi(x^*)$ to propose a better configuration, it may also be close to make the distances covered smaller. This approach will also be evaluated to examine how well this needs to work.

4.2 Directing position switches

Computing the position $\phi(x^*)$ for a node x that would minimize the distances to its neighbors in the lattice can be done in various ways. Then that position may be the basis of a switching strategy that attempts proposes a switch using this position. There may be some restriction on the possibility to directly switch with this position, in order to make the Markov chain irreducible, which will be discussed below.

Computing a minimizing position may of course be done by a brute-force approach, but when we scale the system this is not not an efficient and reasonable method. When we have no explicit way to solve this then another approach is to use approximative schemes.

The actual problem of finding a minimized position can be formulated as a minimization problem like the following. For a node x, we are seeking position $\phi(x^*)$, which minimizes the sum of log distances between the position of x and its neighbors in the base lattice. Since we have a lattice with periodic boundary conditions, for each position there are more than one directions to go when computing the distance. We take three different approaches to minimize the distances between a *selecting node* x and its neighbors:

1. Minimization of the log sum of the edge distances, sampling with a Gaussian (the *concave* method)

2. Minimization of the sum of squared edge distances, sampling with a Gaussian (the *min-squares* method)

3. Minimization of the log sum of the edge distances, mixed with selection uniformly (the *mixed* method)

For each of the methods we begin by selecting a node x uniform at random from the graph. It is then the neighbors, local information, of this node that is used to sample a node y to involve in a switch. This lets us define the switch in various ways that we will evaluate in each Metropolis-Hastings step. Methods 1 and 3 depend on minimizing a sum of concave functions (the log-sum of distances). We have found no efficient way to solve for this in the discrete case (other than the brute-force method). However, since the impact of the distances grows concavely one approximation can be to always select a target *next to* the position of a random neighbor of x. This will be our approximation. Method 2 (minimizing the sum of squares) can however be computed efficiently as a mean on intervals.

After we have computed this position $\phi(x^*)$ (or set of positions) for a method that minimizes the criteria on distances for x, we need to propose an actual switch. The first two methods use a Gaussian, and the the third uses an element of uniformly selecting the switch mixed with a very directed approach. We now go on and describe how this is used.

4.3 Switching randomness and irreducibility

To keep the accept rate high (to not reject too many proposed switches with equation 4) and the chain to be easily irreducible (it is not clear that the chain irreducibility would hold otherwise) these methods do not exclusively try to switch the positions taken by a node and the positions computed that minimize distances to its neighbors. What we try to ensure is that nodes propose position switches that are likely to be accepted with a high probability rather than just uniformly random sampling (as in section 2), but still allows for proposals "uphill" in the energy landscape. This means that we draw positions from a distribution biased towards a better configuration. [3]

So in a decentralized setting what we do with a node x drawn uniformly at random is to attempt positioning it close to x^*, by proposing a position as a reasonable approximation to $\phi(x^*)$. For the first two methods (concave and min-squared), we have chosen the normal distribution centered at x^*, and with the variance depending on how far node x is from the desired center. The idea is that nodes can move gradually closer to a desired position on the graph; being far from one's center makes a node put let restrictive bounds on where it tries to end up. The third (mixed) method instead uses a *fraction* of selecting positions uniformly random, otherwise directly attempting a switch to the position that minimizes the edge distances for it, thus also enabling backward steps of the chain.

4.4 Concave selection

This method tries to approximately minimize the sum of log distances between a node x and its neighbors by applying a Gaussian around the position of a neighbor selected uniformly random. Given that x is seen as the selecting node, y is drawn by its position from the normal distribution $N(\phi(x^*), \lambda \cdot d_\phi(x, x^*))$. It is thus proportional to the distance to the center of the normal distribution, where λ is a parameter for the strength of proportionality. [4] This defines how an x, y-switch is proposed in this method, and let ϕ' be such an x, y-switch of ϕ when a node to switch position with is selected this way. The transition probabilities will then depend both on the degree of the nodes involved and on several positions (the number of neighbors) that are seen as positions to get close to. To evaluate the Metropolis-Hastings chain we get

$$\alpha(\phi, \phi') = \begin{cases} \frac{1}{N}\left(\frac{v(x,y,\phi)}{\text{deg(x)}} + \frac{v(y,x,\phi)}{\text{deg(y)}}\right) & \text{if } \phi' \text{ is } x, y\text{-switch of } \phi \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where $deg(\cdot)$ denotes degree of a node and

$$v(a, b, \phi) = \sum_{n \in N_a} \Phi(\phi(b), \phi(n), \lambda \cdot d_\phi(a, n)) \tag{8}$$

and N_a denotes the set of neighbors of a. The interpretation of v is that we need to take into account each neighbor that is near the position we'd like to switch to, since we can potentially allow for several neighbors making us propose a switch to the same position.

[3] This is also pragmatic in the sense that a node in a distributed implementation may not be able to reach a specific target, but the goal is to ask for a switch that is approximately good.

[4] The motivation for this is roughly: if a node is far away on the lattice then the probability should also be reasonably large when the reverse step of the chain is evaluated with $\alpha(\phi', \phi)$.

Given a proposed x, y-switch, we accept the proposed state ϕ' with probability

$$\beta(\phi, \phi') = min\left(1, \theta \cdot \prod_{i \in E(x \vee y)} \frac{d(\phi(x_i), \phi(y_i))^k}{d(\phi'(x_i), \phi'(y_i))^k}\right) \tag{9}$$

where x_i, y_i are the endpoints on the lattice of edge i connected to the nodes and

$$\theta = \frac{\text{deg(y)}\ v(x, y, \phi') + \text{deg(x)}\ v(y, x, \phi')}{\text{deg(y)}\ v(x, y, \phi) + \text{deg(x)}\ v(y, x, \phi)} \tag{10}$$

4.5 Min-Square selection

The idea of this method is to propose switches that minimize the sum of squared distances to neighbors. This is not directly targetting the energy function, but we may still hope for some improvement. Let x be a node that has been chosen uniformly random, and $\phi(x^*)$ be the position that minimizes the sum of squared distances to neighbors of x. Assume that we have proposed a switch between x and y under the current state ϕ, based on $\phi(x^*)$, where y is drawn by its position from the normal distribution $N(\phi(x^*), \lambda \cdot d_\phi(x, x^*))$. λ is also a parameter as in the previous method. We now wish to evaluate the Metropolis-Hastings chain according to this method of making an x, y-switch. To evaluate the Metropolis-Hastings chain we get

$$\alpha(\phi, \phi') = \begin{cases} \frac{1}{N}(r(x, y, \phi) + r(y, x, \phi)) & \text{if } \phi' \text{ is } x, y\text{-switch of } \phi \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where

$$r(a, b, \phi) = \frac{1}{|M_a|} \sum_{i \in M_a} \Phi(\phi(b), \phi(i), \lambda \cdot d_\phi(a, i)) \tag{12}$$

where $r(a, b, \phi)$ represents the probability under a configuration ϕ that, given node a has first been selected, we then select b (by its position in the lattice) with applying the Gaussian around $\phi(a^*)$. M_a is the set of positions minimizing the sum of squared distances to positions taken by its neighbors. Computing $\alpha(\phi', \phi)$ depends on the state ϕ' where $\phi'(x^*)$ and $\phi'(y^*)$ will be the same unless if switching with neighbors. Given an x, y-switch we accept the proposed state ϕ' with

$$\beta(\phi, \phi') = min\left(1, \eta \cdot \prod_{i \in E(x \vee y)} \frac{d(\phi(x_i), \phi(y_i))^k}{d(\phi'(x_i), \phi'(y_i))^k}\right) \tag{13}$$

where

$$\eta = \frac{r(x, y, \phi') + r(y, x, \phi')}{r(x, y, \phi) + r(y, x, \phi)} \tag{14}$$

4.6 Mixed selection

The idea of this method is to propose good switches between nodes most of the time, which we choose for a node to be positioned next to the position a random neighbor takes on the lattice. The method is to use a mix between directed switches (that minimize distances very well) and proposing switches uniformly at random. This also makes the chain irreducible. The method is a one-parameter model, where we first pick one node x uniformly random. Then with probability p (the parameter) we make a selection uniformly random for a node to involve in switching, and with probability $1 - p$ we make a directed switch to a node that sits next to one of the neighbors of x. To guarantee the chain to be irreducible it should be possible to select the parameter arbitrarily small but larger than 0.

Let us assume x and y are involved in an x, y-switch under this selection kernel. ϕ is the current state, ϕ' is the state where the positions of x and y are switched as described previously. Then α depends on whether the x and y proposed are next to each other's neighbors or not. [5]

Evaluating Metropolis-Hastings is done as

$$\alpha(\phi, \phi') = \begin{cases} \frac{1}{N}(w(x, y, \phi) + w(y, x, \phi)) & \text{if } \phi' \text{ is } x, y\text{-switch of } \phi \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where

$$w(a, b, \phi) = \begin{cases} \frac{p}{N-1} + \frac{1-p}{deg(a)} \frac{N_{a,b}}{D_k} & \text{if } \phi(b) \text{ next to graph neighbor of } a \\ \frac{p}{N-1} & \text{otherwise} \end{cases} \tag{16}$$

where *deg(a)* denotes the degree of node a, and $N_{a,b}$ is the number of neighbors of a that take positions next to $\phi(b)$ in the lattice. A factor of D_k appears depending on the number of neighbors a node can take in a k-dimensional lattice.

As before, we accept the proposed state ϕ' with

$$\beta(\phi, \phi') = min \left(1, \sigma \cdot \prod_{i \in E(x \vee y)} \frac{d(\phi(x_i), \phi(y_i))^k}{d(\phi'(x_i), \phi'(y_i))^k} \right) \tag{17}$$

where

$$\sigma = \frac{w(x, y, \phi') + w(y, x, \phi')}{w(x, y, \phi) + w(y, x, \phi)} \tag{18}$$

4.7 Method

We initialized a number of small-worlds graphs where edges were added according to the Kleinberg model. After initial performance measurement, the graphs were shuffled and given to the Metropolis-Hastings algorithm. We then run the chain with uniformly selected position switches, periodically evaluating greedy routing until we have reached our stop criteria (for both of the criteria defined in section 3).

[5] A node may also be assigned next to more than one of its graph neighbors in the lattice.

Our method to compare the different methods is to see how many steps it takes to reach the same performance (as the uniform method) where they stopped. For each of the criteria we measure performance of average length and success rate when we have reached the criteria. Then we take a copy of the initial graph and give this to each of our local selection methods, and run the algorithms on the graph until we have reached the same performance in average length and success rate. This will thus measure how fast the different selection methods reach the same performance as defined by the two criteria for the original algorithm. The period of evaluation is set to 50 rounds.

4.8 Selection width when using a Gaussian: from local towards uniform

Studying the impact of the parameter λ as given in sections 4.4 and 4.5 means weighting the distance between a node and the position(s) that minimizing distance to its neighbors. Making λ too small when proposing ϕ' from ϕ would propose better configurations, but on the other hand it may make proposing the reverse step with $\alpha(\phi', \phi)$ less likely. Thus this is a tradeoff for the greedyness of the search strategy, against keeping the chain irreducible and with a good accept rate.

We study the parameter with a fixed network size and run the Metropolis-Hastings algorithm for varying λ to measure performance as in section 2.1. We then compare the rounds needed, bounding on the ideal performance.

4.9 Rates in the mixed method

We also study the rate of uniformly selecting switches against to directing them next to a neighbor in the mixed method. Since our model gives each neighbor a number of neighbors that depends on the network size, the network size may have some effect on the efficiency of the method. Starting with networks of varying sizes, for each graph we run the mixed selection, bounding on the ideal criteria as before and varying the parameter.

4.10 Results and evaluation

The main results are given in figure 9 and 10. The results on the different criteria show slightly different things. First we can say that the min-square selection almost always takes largest number of steps to reach the performance of both the stop criteria, except for the smallest graph sizes we simulated (see the discussion of results in section 3). But interestingly it is still not prohibitively worser to use as we had expected. From this we can speculate that either a general approach to make switches close to neighbors of a node is quite robust (arranging nodes so that they indicate roughly where their neighbors are), or that this depends on a clustering effect of neighborhoods on the lattice. The clustering option, however, does not seem very likely since we have measured the clustering coefficient as given in [1] which is equal or less than 0.1 for our generated graphs (far lower than what has been reported in real social networks).

The other results indicate a somewhat split view between the approximate methods (*concave* and *mixed*) that try to get close to randomly selected neighbors. Using the first criteria the mixed method seems faster, but when we studied the results from the second criteria the concave selection seems faster (especially for large sizes). This was however easily attributed due to the requirement on *success rate* for the first criteria. Since the success rate was required to get as high as the one measured when stopping, this got more impact than in the second criteria (typically success rate was close to 1 when stopping, the second criteria requires maximally 90seemed to have just a slight better performance on the success rate (and the other methods seemed to need some time to just increase this a *little* bit).

Figures 12 and 13 shows results of evaluating the selection witdh parameter λ for the “Gaussian” selection methods and parameer p for the mixed selection. These have been studied when selecting the parameters for the simulations. We can see that when increasing λ the algorithm begins to give results more like switching with uniformly drawn nodes. This makes the effect of sampling switches with a bias seem robust since the effect can be seen over a relatively broad range of parameter values. For the mixed selection, we can see how the effect for a large probability of switching with next-to-neighbor nodes trails off for larger network sizes. This may be due to that we scale the number of friends logarithmically, but varying model parameters are needed to study this in larger detail. In practice we could see that the performance got very close (but above) to the bound on the ideal model used, so a different bound may be needed for studying this parameter better.

In summary, our methods improve the speed of the algorithm. We also have some evidence for that the concave method seems best fit to reach the stronger (bounding on ideal performance) stop criteria. All the runs seemed to indicate the same trend with a low variance, but the conclusions so far are bounded by the computational resources available to the author.

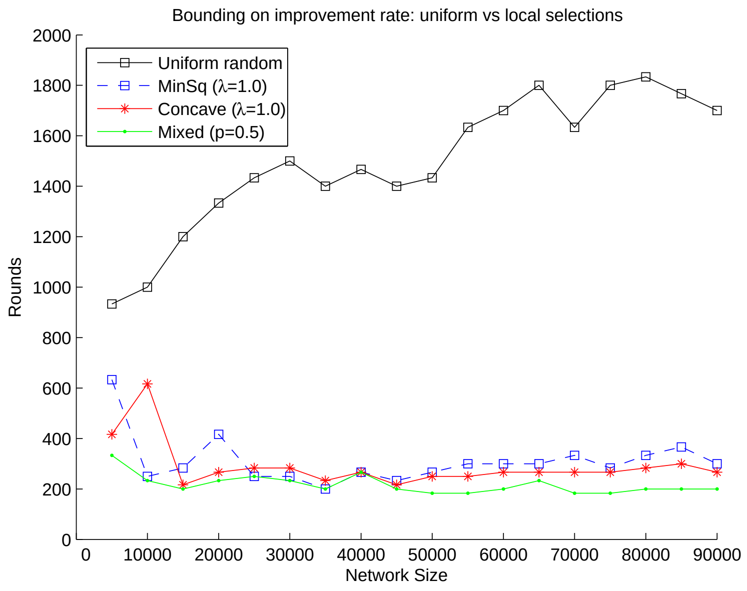


Figure 9: Bound on improvement rate, k=0.02, for local methods the number of rounds to reach same performance as when selecting uniformly stopped. Averaged over 10 runs.

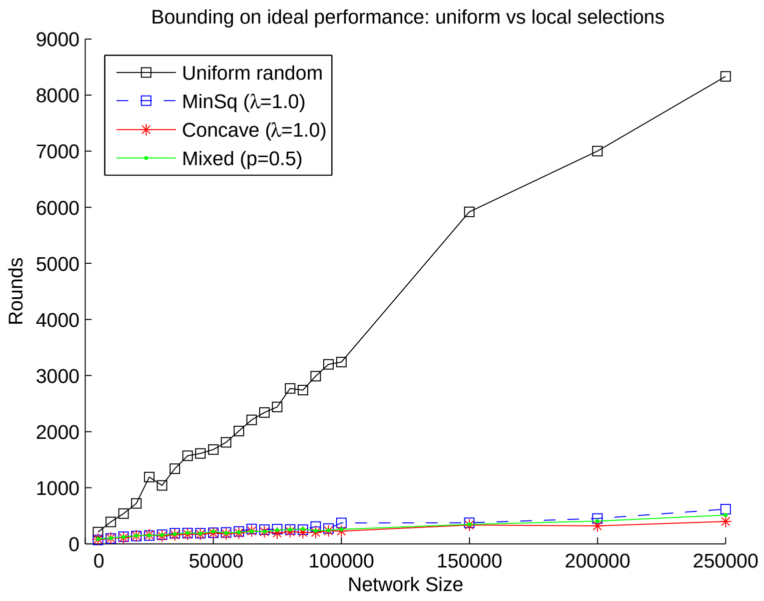


Figure 10: Bound on the ideal performance. Rounds until $L < 2L_{ideal}$ and $sr > 0.9sr_{ideal}$. Averaged over 10 runs.

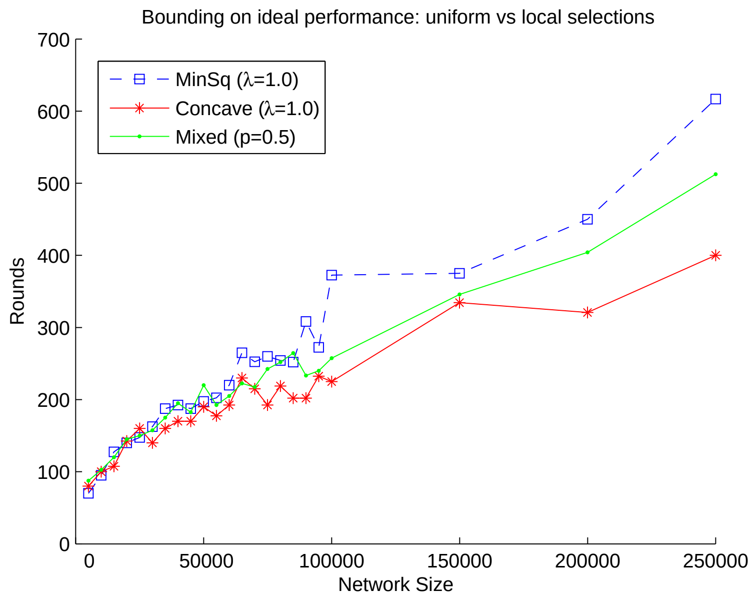
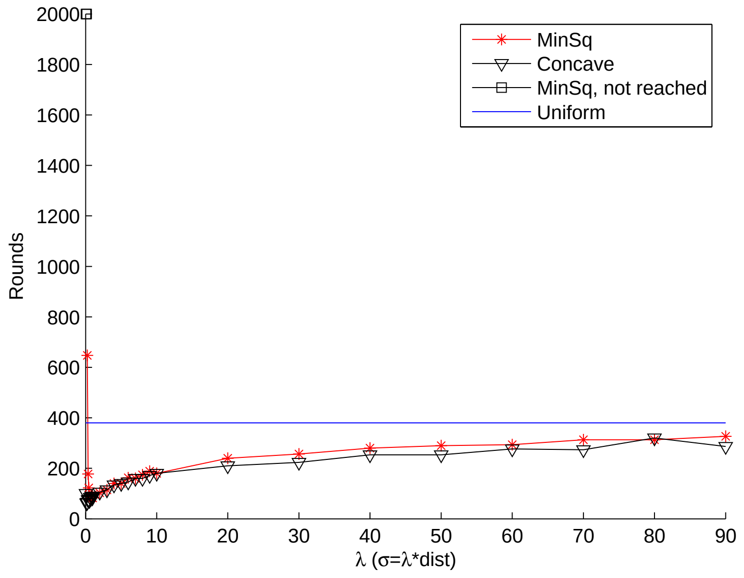


Figure 11: Bound on the ideal performance. Rounds until $L < 2L_{ideal}$ and $sr > 0.9sr_{ideal}$. Averaged over 10 runs.



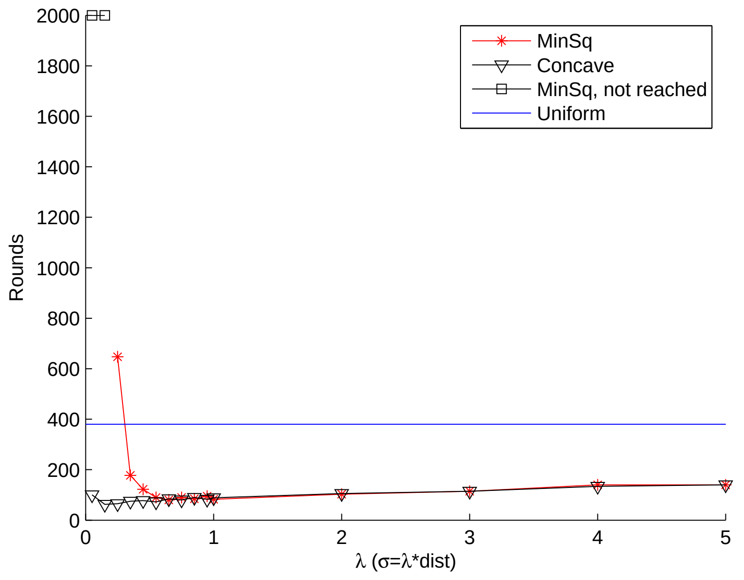


Figure 12: Impact of parameter λ for *concave* and *min-square* selection. $N = 10000$. Rounds until $L < 2L_{ideal}$ and $sr > 0.9sr_{ideal}$. Averaged over 10 runs.

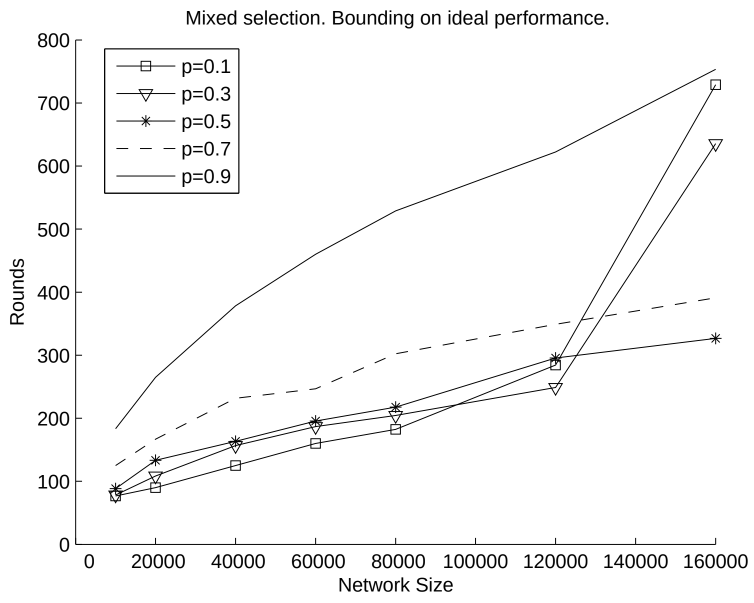


Figure 13: Impact of parameter p for mixed selection. Rounds to reach Rounds until $L < 2L_{ideal}$ and $sr > 0.9sr_{ideal}$. Averaged over 10 runs.

5 Generalizing position switching

It may also be possible to include more than two nodes in the position switches that have so far defined the transitions of the Markov chain. In this section we consider how this could work and if this may be practical.

Let us first consider to define the neighbor relation on the chain where k different nodes $x_0, x_1, ..., x_{k-1}$ are involved. This gives $k!$ configurations to select among for the next state of the chain (all different permutations of the positions). In the main problem discussed before we have considered an assignment on S^V of the values $S = \{0, 1, ..., N-1\}$ to each node in V, where each value may be associated only once to each node by ϕ. Now we consider a subset of assignments including k different values from S and k different nodes from V. The notation then is $V' = \{x_0, x_1, ..., x_k\}$ and a set of values $S' = \{\phi(x_0), \phi(x_1), ..., \phi(x_{k-1})\}$, and we are considering an assignment $S'^{V'}$. Note that this case also contains all the possible position switches containing *fewer* than k nodes changing positions. E.g. a chain with $k = 3$ also gives the possibility to only switch positions between 2 of the nodes involved (one of the nodes stays at its position in the proposed configuration).

5.1 Uniform k-switching

How do we design a selection kernel involving k nodes? One way may be draw k nodes uniformly at random, propose a switch based on the order in which we drew these nodes, and then consider all the $k!$ different ways in which the nodes may switch positions among each other. Another way which we will use here is to evaluate this the freedom of considering $k!$ different possible configurations (when k is relatively small).

Assume that k nodes have been selected uniformly at random. We are now free to evaluate each and all the different $k!$ configurations resulting from this switch, as long as the computation is practical. One way is to draw among these randomly, something that seems equivalent to the discussion above, but another approach that should take us closer to minimizing the log sum is to give bias for the configuration resulting in the smallest log sum of edge distances. This is also local information, as in previous examples, however the information is spread out over k nodes involved (instead of previously two nodes as in section 4.

With larger k, this quickly becomes heavier to simulate for a chain and the number of nodes involved becomes very large. The most direct way to sample a good configuration in this case seems to revert to the initial problem (with $k = N$), and probably to run the Metropolis-Hastings chain all over again.

5.2 Example: Uniform 3-switching

As an experiment we attempt letting 3 nodes switch at once. This does not seem unmanagable, even in a distributed implementation. When we have uniformly selected 3 nodes, we evaluate all of the 6 configurations and randomly draw a sample depending on which such configuration is best at minimizing the log sum. Our switching uses one parameter p between 0 and 1 which denotes the probability to draw one of the allowed configurations uniformly random. Otherwise we draw the best (minimizing) configuration.

Let the configuration ϕ be the current state of the chain as before, and let x,y,z be three uniformly drawn nodes. In all the different ways of drawing the nodes we will use the same switching scheme which makes the order unimportant. What guides the selection is what the nodes prefer *after evaluation* of the different configurations. Let ϕ' be the configuration proposed as above. To evaluate the Metropolis-Hastings chain we get

$$\alpha(\phi, \phi') = \begin{cases} \frac{6}{n(n-1)(n-2)} u(\phi, \phi') & \text{if } \phi' \text{ x,y,z-switch of } \phi \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

where $u(\phi, \phi')$ depends on whether the configuration proposed is best or not

$$u(\phi, \phi') = \begin{cases} p/6 + (1-p) & \text{if } \phi' \text{ is the best available configuration} \\ p/6 & \text{otherwise} \end{cases} \tag{20}$$

For each state proposed as with the now described α we accept it with

$$\beta(\phi, \phi') = min\left(1, \frac{u(\phi', \phi)}{u(\phi, \phi')} \prod_{i \in E(x \vee y \vee z)} \frac{d(\phi(x_i), \phi(y_i))^{\Delta}}{d(\phi'(x_i), \phi'(y_i))^{\Delta}}\right) \tag{21}$$

where Δ is the dimension of the lattice and $E(x \vee y \vee z)$ denotes the edges connected to the three nodes in the switch. Generalizing this scheme can be done easily, but in practice it quickly becomes heavy to simulate when k grows.

5.3 Method

First we compare the new method against uniformly selecting two nodes (as previously described in section 1.5.4). We run the dynamics with the stop criteria that bounds on the ideal performance. When reached we take a copy of the initial graph and run the dynamics again with 3-switching to reach the same goal as where the previous dynamics was stopped.

5.4 Results and evaluation

The results are shown in figures 14 and 14. We can see that giving two nodes biased selection of switching targets (for minimizing the log sum) gives more impact so far than considering one more node uniformly selected (but with bias for resulting configuration).

We note that fewer steps are typically required to reach performance at the second stop criteria. But the implications are more complicated in a distributed implementation, requiring more messages between nodes (and more nodes involved). Since each step involves more nodes it is also not obvious that the steps taken by the chain is a good measurement on the speed of the process.

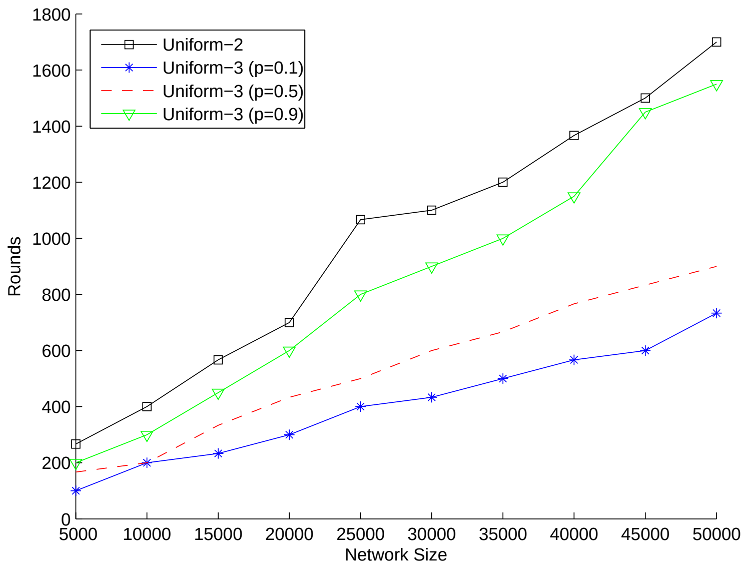


Figure 14: Comparing 3-uniform against 2-uniform. Rounds until $L < 2L_{ideal}$ and $sr > 0.9sr_{ideal}$. Averaged over 5 runs.

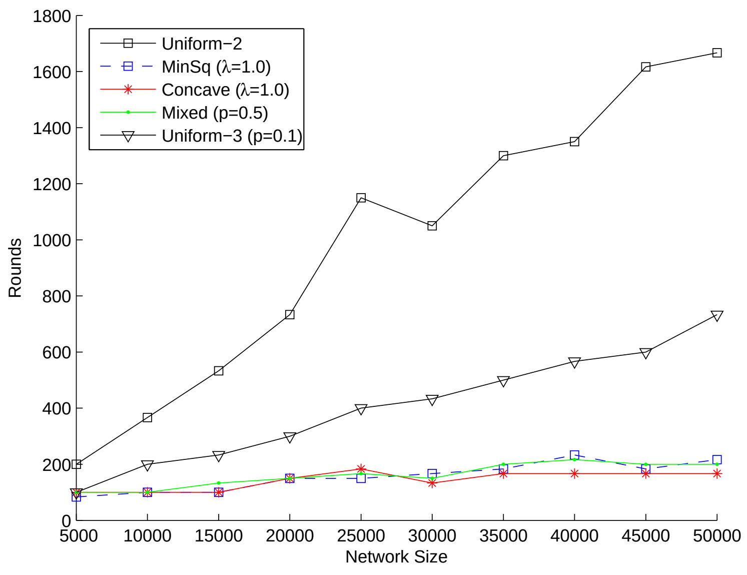


Figure 15: Comparing 3-uniform against local selection methods with 2 nodes. Rounds until $L < 2L_{ideal}$ and $sr > 0.9sr_{ideal}$. Averaged over 5 runs.

6 Conclusion

The main results of this thesis are three new selection kernels that depend on the position of a node and its neighbor positions in the lattice. Each of the three methods were shown to perform better than selecting switches uniformly random. The methods that sought to minimize the log sum of lengths covered by edges performed better than minimizing sum-of-squared distances, as expected. But minimizing sum-of-squared distances did not perform prohibitively worse and still clearly better than the uniform method.

We have also extended the Metropolis-Hastings selection kernel to include an additional node uniformly in each switch of the algorithm. The results improved in the number of chain steps, but did not improve as well as proposing switches with a bias for minimizing distances.

There are several topics that are open for further investigation. Our results depend on precisely being able to propose switches between the nodes according to the selection kernels. Investigating how well this would work in a distributed implementation, especially when the graph is not reasonable navigable, is important for practical considerations. There are still reasons to be optimistic since two of our methods are approximatively directed to produce better switches.

When involving more nodes in a switch other questions may appear on how these methods can be used in practice. We have used one rather direct way to involve more nodes, but there seems to be several other ways to do this, as well as the possibility to direct switches when including more nodes (as we have done with two nodes). Considering how to use this in a decentralized implementation, factors such as the number of messages that are needed between nodes in a graph to make switching efficient could be studied and be used as a comparison method.

The results so far are all using one specific method for switching until certain bounds were reached. It should also be possible to use different methods in different phases. Some methods may have more impact in the starting configuration of a network whereas another strategies may be better when the algorithm has made a network improve on navigability. Another way to vary the switching may be to use the Simulated Annealing approach with an annealing schedule.

We also believe that using these methods in a decentralized and distributed way to organize overlay networks may depend on cooperation with peers keeping to a switching protocol. Making a model with incentives for and against switching positions may open for studying game theoretic situations in such networks.

Finally, one should also be aware about that the results depend on networks generated by the Kleinberg model. Evaluating the methods on wider models and real-world data is important, since practical usability of these methods should depend a lot on how the underlying network has been formed.

A Metropolis-Hastings: role of the selection kernel

Theorem A. *The Markov chain of the Metropolis-Hastings algorithm given in section 1.5.3 has π as the stationary distribution, independent of the selection kernel α.*

Proof. We will show this by showing that the chain is reversible, that is $\forall i, j : \pi_i P_{i,j} = \pi_j P_{j,i}$. From this the equation for stationarity $\pi = \pi P$ follows directly.

Let a state k be the *neighbor* to i if it can be proposed from i directly by α. Using the form of Metropolis-Hastings as given in section 1.5.3 we get the transition matrix by

$$P_{i,j} = \begin{cases} \alpha(i,j)\beta(i,j) & \text{if } i \neq j \text{ are neighbors} \\ 1 - \sum_k \alpha(i,k)\beta(i,k) & i = j \text{ and } i,k \text{ are neighbors} \\ 0 & \text{otherwise} \end{cases}$$

This is rewritten as

$$P_{i,j} = \begin{cases} \alpha(i,j) min\left(1, \frac{\pi_j \alpha(j,i)}{\pi_i \alpha(i,j)}\right) & \text{if } i \neq j \text{ are neighbors} \\ 1 - \sum_k \alpha(i,k) min\left(1, \frac{\pi_k \alpha(k,i)}{\pi_i \alpha(i,k)}\right) & i = j \text{ and } i,k \text{ are neighbors} \\ 0 & \text{otherwise} \end{cases}$$

What we need to do is to show that the chain is reversible. For the second and third case this is easily shown; equality holds directly with $i = j$ and in the null case. For the first case, we rewrite $\beta(i,j) = min(1, z)$ where $z = \frac{\pi_j \alpha(j,i)}{\pi_i \alpha(i,j)}$. We need to check that $\pi_i P_{i,j} = \pi_j P_{j,i}$ holds in the two cases where $z < 1$ and $z \geq 1$.

Case $\mathbf{z} < \mathbf{1}$:

$$\begin{aligned} \pi_i P_{i,j} = \pi_i \alpha(i,j)\beta(i,j) = \\ \pi_i \alpha(i,j)\, min(1, z) = \\ \pi_i \alpha(i,j) z = \\ \pi_i \alpha(i,j) \frac{\pi_j \alpha(j,i)}{\pi_i \alpha(i,j)} = \\ \pi_j \alpha(j,i) \end{aligned} \tag{22}$$

$$\begin{aligned} \pi_j P_{j,i} = \pi_j \alpha(j,i)\beta(j,i) = \\ \pi_j \alpha(j,i)\, min(1, z^{-1}) = \\ \pi_j \alpha(j,i) \end{aligned} \tag{23}$$

Case $\mathbf{z} \geq \mathbf{1}$:
Follows similarly (by the same method). □

Thus it can be concluded that the chain has the stationary distribution π that does not depend on α, and that we are free to design it with respect to other constraints of the chain. If the chain needs to be ergodic (ie. to be able to start from any possible configuration (distribution) and converge towards the stationary distribution) the neighbor relation α still needs to allow for it to be irreversible and aperiodic.

References

[1] R. Albert and A-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74, 2002.

[2] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, 2001.

[3] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. *Theory of Computer Science, 1:237-267*, 1978.

[4] Gilks, Richardson, and Spiegelhalter. *Markov Chain Monte Carlo in practice*. Chapman and Hall/CRC, 1996.

[5] Olle Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2002.

[6] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[7] Judith Kleinfeld. Could it be a big world after all? *Society*, 2002.

[8] Stanley Milgram. The small world problem. *Psychology Today, 1:61-67*, 1967.

[9] Oskar Sandberg. Distributed routing in small-world networks. In *8th Workshop on Algorithm Engineering and Experiments (ALENEX06)*, 2006.

[10] D. Watts and S. Strogatz. Collective dynamics of small world networks. *Nature*, 1998.

[11] Duncan Watts. *Small Worlds*. Princeton University Press, 1999.