Non-Backtracking Random Walks on Chess Networks

Candidate Number 1006289

Concepts from graph theory have been applied to chess for many years. The idea of a knight's tour - i.e. a Hamiltonian Tour of a chessboard using a knight's move - dates back to the 9th century AD (1). Non-backtracking random walks on networks are a comparatively new idea in network analysis. This paper studies non-backtracking walks on networks based on various chess pieces.

1. Summary of Contents and Outline of Problem

A. Summary of Contents.

Chapter	Chapter Summary
2	Markov chains and non-backtracking random walks.
3	Chess networks are defined and the clustering coefficient and average degree for chess networks are calculated.
4	Different types of random walks on chess networks are modelled, and the time taken to visit all vertices of the network is compared.
5	Random chess networks are defined, and the probability of being connected with minimum degree ≥ 2 is calculated.
6	The mixing rate of Rook networks is analysed, and the probability of a vertex being revisited for a non-backtracking random walk on a Slow Jumping Rook network is calculated.
7	The clustering coefficient for a Queen network is estimated using a non-backtracking random walk.
8	Modelling issues and possible extensions.

B. Current Study of Chess and Mathematics. There are three main types of mathematical chess problems that are currently studied:

- Piece Tour: Find a tour of a chess piece on an $n \times n$ chessboard.
- Independence: Find the maximum number of a given chess piece that can be placed on an $n \times n$ chessboard such that no two of the piece attack each other.
- Domination: Find the minimum number of a given chess piece that can be placed on an $n \times n$ chessboard such that all squares are attacked by at least one of the pieces.

This paper instead examines the networks formed by chess pieces on chessboards, and random walks on these networks.

2. Random Walks on Networks

Definition 1 *Let $G = (V, E)$ be an undirected graph. A* ***random walk*** *(RW) of length k on G, from some given vertex $w_0 \in V$ is a uniformly chosen member of: $W(k) = \{w_0, w_1, ..., w_k\} : w_t \in V, w_{t-1}w_t \in E$ for all $t \in [k]$.*

Definition 2 *Define G as above. A* ***non-backtracking random walk*** *(NBRW) of length k on G is a random walk of length k on G, with the additional restriction $w_{t-1} \neq w_{t+1}$ for all $t \in \{1, ..., k-1\}$.*

Definition 3 *A random walk $W(k)$ is said to* ***tour*** *a graph $G = (V, E)$ if $\forall v \in V, v \in W(k)$.*

Definition 4 *Define $S(n)$ to be the number of distinct vertices visited on a random walk of length n on a graph G.*

Definition 5 *The* ***stationary distribution*** *$\pi = \pi(u)$ for a state u of a Markov chain is defined to be a probability distribution π, such that $\pi = \pi P$, where P is the transition matrix of the Markov chain.*

Remark 1 *As (4) shows, a NBRW is not a Markov chain on V. A Markov chain must satisfy the property that "the future depends only on the current state and not on the past state", but a NBRW requires information about the previous vertex visited.*

However, it is possible to turn a NBRW into a Markov chain: To each edge $e \in E$, associate two oppositely oriented edges $e, \tilde{e}$, and denote the initial and terminal vertex of an oriented edge e by e^- and e^+, so $\tilde{\tilde{e}} = e$, $(\tilde{e})^+ = e^-$,$(\tilde{e})^- = e^+$. Now consider a NBRW to be a Markov chain on $\tilde{E}$, the set of oriented edges, with transition matrix $Q_{\tilde{E}} = (q_{\tilde{E}}(e,f))_{e,f\in\tilde{E}}$ defined by

$$q_{\tilde{E}}(e,f) = \begin{cases} \frac{1}{deg(e^+)+1}, & e \to f, \text{ i.e. } f^- = e^+ \text{ and } f \neq \tilde{e}. \\ 0, & \text{otherwise.} \end{cases} \qquad [1]$$

Define $q^{(n)}(x,y)$ and $q^{(n)}_{\tilde{E}}(e,f)$ to be the n-step transition probabilities, where x, y are vertices and e, f are oriented edges. The following equation relates these two transition probabilities:

$$q^{(n)}(x,y) = \frac{1}{deg(x)} \sum_{\substack{e,f\in\tilde{E} \\ e^+=x, f^+=y}} q^{(n)}_{\tilde{E}}(x,y) \qquad [2]$$

3. Chess Networks

This paper will focus chess networks for a given chess piece. This is an undirected graph, where the vertices correspond to the squares of a $n \times n$ chessboard, and there exists an edge between two vertices if and only if the chess piece can move from one square to another.

Definition 6 *A* ***standard chessboard*** *refers to an 8×8 chess network.*

Significance Statement

This paper defines the idea of chess networks and calculates important properties of these networks. It also models different types of random walks on these networks. In (2), it is shown how to calculate the mixing time for a non-backtracking walk on a regular graph. This paper applies this idea in order to find the mixing time for a Rook network. This paper then uses an concept from (3) to estimate the clustering coefficient for a Queen network using a non-backtracking random walk.

A. Movement of the Pieces. Here the movement of the pieces for chess networks is listed. Note that the Pawn (another chess piece) is not considered, because the resulting network is directed, and the following results apply to undirected networks only.

- The Slow Rook can move horizontally or vertically, one square at a time.
- The Bishop can move any positive number of squares diagonally.
- The Knight can move two squares horizontally and one square vertically, or two squares vertically and one square horizontally.
- The King can move one square diagonally, horizontally or vertically.
- The Rook can move any positive number of squares horizontally or vertically.
- The Queen can move like a Rook or a Bishop.
- The Fairy can move like a Queen or a Knight.

B. Diagrams of Movement. Figure 1 shows the diagram for the chess network of a Knight on a standard chessboard. Figure 2 shows the movement of the Knight. The movement of the other pieces is included in the Supplemental Information.

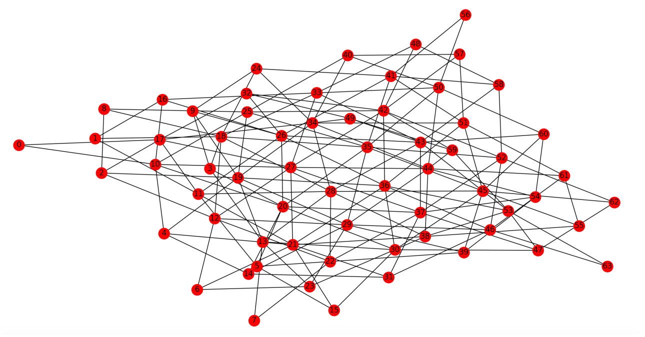


Fig. 1. Network for Knight on Standard Chessboard

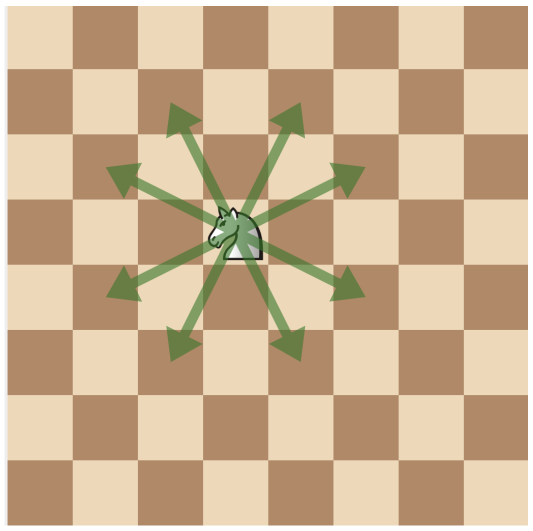


Fig. 2. Movement of Knight

C. Properties of Chess Networks. In this paper, the **clustering coefficient** always refers to the network average clustering coefficient, as defined by (5). Figure 3 shows the clustering coefficient for different chess pieces on chessboards of size less than 20. Since a Knight's move always changes the colour of its square, any cycle on a Knight network has even length, therefore this network contains no triangles. This means that the clustering coefficient for a Knight network is 0.

Remark 2 *The clustering coefficient for the Fairy network approaches the clustering coefficient of the Queen network as n increases. This might be because the move of the Fairy is equivalent to the move of a Queen or a Knight.*

Remark 3 *The piece with the highest clustering coefficient for large n is the Bishop, with clustering coefficient ≈ 0.55. This might be because the Bishop network has two connected components with an approximately equal number of vertices, since Bishops can only move on squares of one colour.*

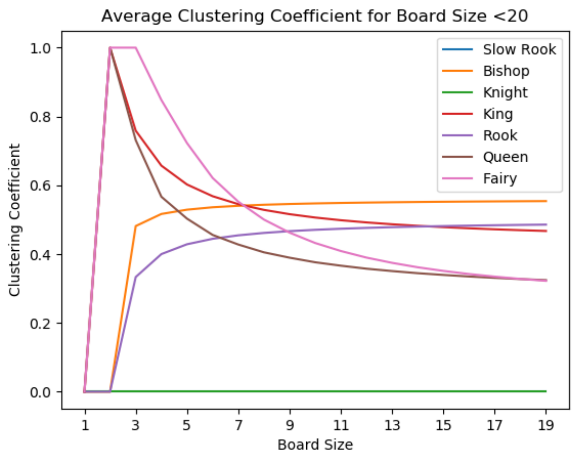


Fig. 3. Clustering Coefficient for the Pieces

Definition 7 *The girth of a graph is the length of its shortest cycle. The girth of an acyclic graph is defined to be ∞.*

Remark 4 *For a $n \geq 4$, the chess network for each piece contains a cycle and therefore has finite girth. For the Slow Rook and the Knight, the girth is 4, otherwise the girth is 3.*

D. Average Degree for Chess Networks. Figure 4 shows the average degree for different chess pieces on chessboards of size at most 20.

Remark 5 *For the $n \times n$ Knight network, there are $(n-2)^2$ vertices for which the degree is 8; remaining vertices have degree less than 8. Therefore, the average degree asymptotically approaches 8. Similar methods show that the average degree for the Slow Rook and King networks asymptotically approaches 4 and 8 respectively.*

4. Random Walks

In this section three different methods are used for modelling random walks on a graph and $S(n)$ is compared for different types of walk and different piece networks.

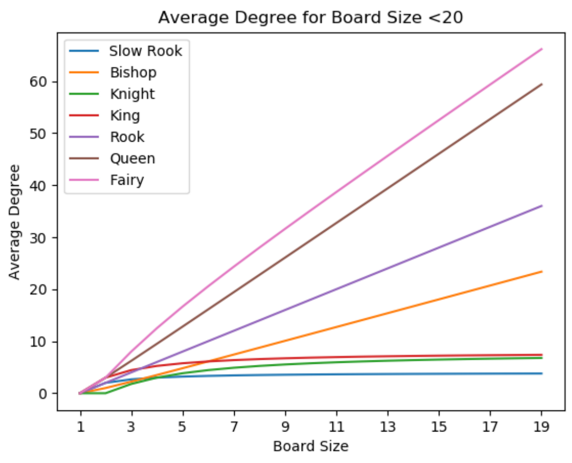


Fig. 4. Average Degree for the Pieces

A. Covering Times. It is useful to introduce the notion of covering time for a random walk.

Definition 8 *The cover time $C(G)$ for a random walk on a connected graph G is the expected number of steps for the random walk to visit all the vertices of G.*

Proposition 1 *(6) gives an upper bound for the cover time of a random walk:*

$$C(G) \leq 4|E|\left(|V|-1\right) \leq 2n(n-1)^2 \qquad [3]$$

Proof 1 *Define the **hitting time** from u to v,*

$$H(u,v) = \mathbb{E}[min\{n \in \mathbb{N}\backslash\emptyset : X_t = v\}|X_0 = u] \qquad [4]$$

Recall $H(u,u) = \pi(u)^{-1}$, the reciprocal of the stationary probability of u. Applying this result gives the equalities

$$\frac{2|E|}{deg(u)} = H(u,u) = \frac{1}{deg(u)} \sum_{v\in V, uv\in E} (H(u,v)+1) \qquad [5]$$

whence $H(u.v) < 2|E|$. Let T be a spanning tree of G, with a traversal $\left(v_0, v_1, ..., v_{2|V|-2} = v_0\right)$ which visits every vertex of T at least once, and every edge of T at most twice. Then the $C(G)$ is bounded by the expected time for this traversal of T:

$$C(G) \leq \sum_{i=0}^{2|V|-3} H(v_i, v_{i+1}) \leq (2|V|-2)(2|E|) \qquad [6]$$

Using the result that $|E| \leq \frac{|V|(|V|-1)}{2}$ completes the proof.

Remark 6 *This implies that the cover time for an $n \times n$ chess network should be at most $4n^6$. Any type of random walk that takes a longer time on average to tour the network is certainly slower than the standard random walk.*

B. Three Methods for Random Walks on a Graph. In each type of random walk, the starting vertex is picked at random.

- Backtracking Random Walk on a Graph (BRW): the next vertex of the walk is picked at random from the neighbours of the current vertex.
- Non-Backtracking Random Walk (NBRW): the next vertex of the graph is picked at random from all the neighbours of the current vertex, except for the previous vertex.
- Unvisited Vertices Random Walk (UVRW): the next vertex is picked at random from all the unvisited neighbours of the current vertex. If all the current vertices have been visited, the next vertex is picked at random from all the neighbours of the current vertex.

Remark 7 *Note that the maximum value of $S(n)$ is equal to the size of the connected component of the start point. Therefore a random walk on the Bishop network - which is the only chess network with more than one connected component - can only visit approximately half the vertices of the network.*

C. Comparison of Covering Time for Different Pieces. Figure 5 compares the number of distinct squares visited for the Knight networks for a board size of 30, with 20 trials; touring times are tabulated below. For other pieces, diagrams are included in the Supplemental Information. For all of the pieces, the UVRW toured the networks in the fewest number of steps.. For the Knight network, the number of steps to tour the graph using the NBRW was 34% less than for the BRW.

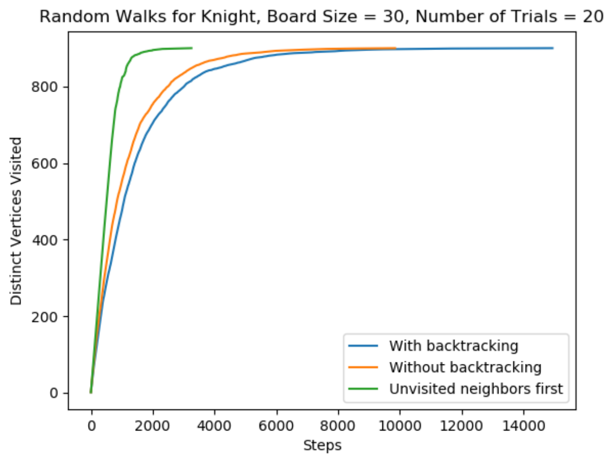


Fig. 5. Knight Comparison

Type of RW	Average Number of Steps to Tour 30x30 Knight Network
BRW	14800
NBRW	9700
UVRM	3500

D. Cover Times for Different Pieces. Figure 6 shows the time to visit all the vertices in the connected component of the start vertex, using a NBRW. Figures for other types of random walk are included in the Supplemental Information. In each case, the performance of the Rook, Queen and Fairy were similar. For the BRW and the UVRW, the Slow Rook network took the most number of steps to tour the network, but for the NBRW, the King took the most number of steps.

E. Covering All Edges of Chess Network. One variant of this problem is to measure the number of steps to visit every edge in the connected component of the start point. In Figure 7, the time taken for different pieces for a NBRW are compared. The

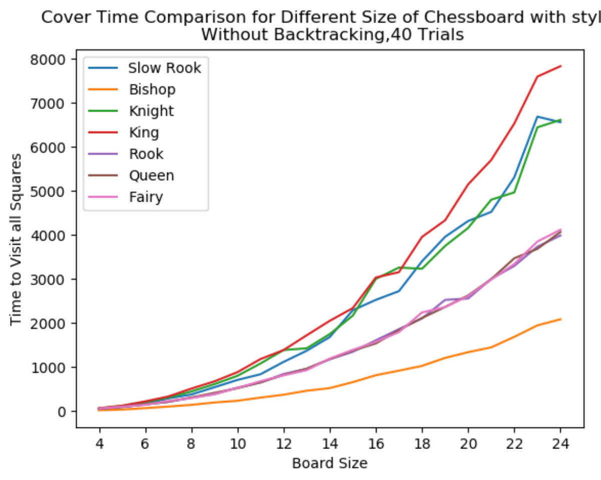


Fig. 6. Non-Backtracking Cover Times

graphs for the other styles are included in the Supplemental Information. In these simulations, the time taken to visit every edge in a random walk is proportional to the average degree of the graph. For any graph $G = (V, E)$,

$$avg(G) = \frac{2|E|}{|V|} \qquad [7]$$

where $avg(G)$ denotes the average degree of G. If $|V|$ is fixed, the number of edges is directly proportional to the average degree. In this model, the type of random walk does not significantly affect the time taken to visit all the edges of the graph.

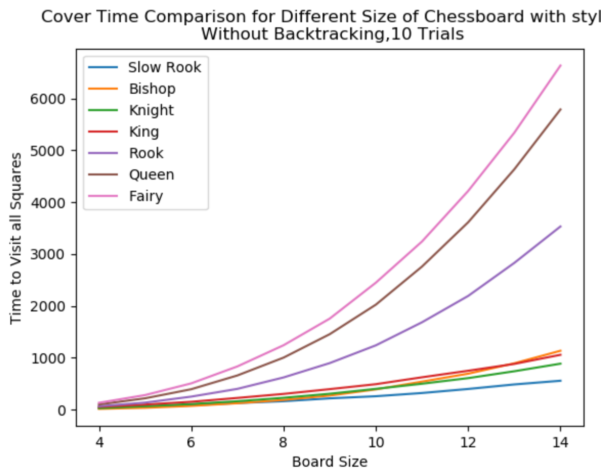


Fig. 7. Non-Backtracking Edge Cover Times

5. Random Ensembles

In this section a random ensemble for chess networks is presented.

Definition 9 *Let a $G(n, p)$* ***piece network****, for a given piece denote a graph where the vertices are the squares of a $n \times n$ chessboard, where each square is present with probability p. An edge exists between two vertices iff the given piece could move between the two corresponding squares.*

Example 1 *In Figure 8, a $G(8, 0.5)$ King network is illustrated, both using a chessboard and a network diagram.*

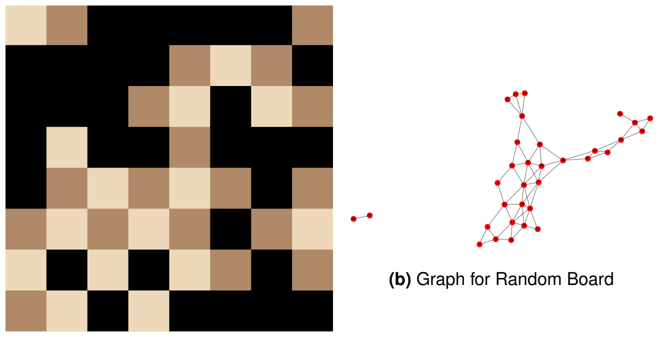


Fig. 8. Random Chessboard

For a fixed n and p, a NBRW will be able to visit all the vertices of $G(n, p)$ only when $G(n, p)$ is a connected graph with minimum degree ≥ 2. The second condition is needed to ensure that the random walk is not forced to backtrack. Figure 9 shows the probability of being able to cover all vertices of $G(n, p)$ with a non-backtracking random walk as p changes. The higher the average degree of the regular network, the more likely it is that a random chess network will be connected with average degree ≥ 2.

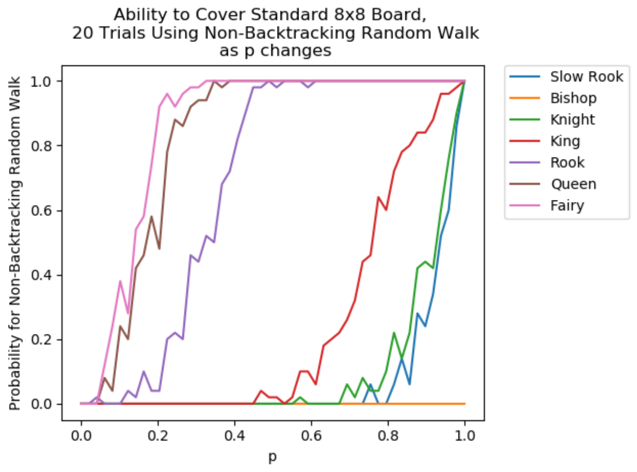


Fig. 9. Probability that $G(n, p)$ is connected with minimum degree ≥ 2

6. Rook Networks and Slow Rook Networks

A. Results on Rook Networks. Rook networks have an important property: they are strongly regular. Therefore results about a NBRW on strongly regular graphs can be applied. For the rest of this section let G_n denote a Rook network on an $n \times n$ board. For G_n the degree of each vertex is equal to $2n - 2$, because there are $n - 1$ squares each in the same row and column of any given square. Therefore G_n is a $(2n - 2)$-regular graph on n^2 vertices.

Definition 10 *Let $G = (V, E)$ be a k-regular graph on v vertices. G is **strongly regular** if there exist integers λ and μ such that every two adjacent vertices have λ common neighbours, and every two non-adjacent vertices have μ common neighbours. Denote such a graph by $srg(v, k, \lambda, \mu)$.*

Proposition 2 *G_n is $srg(n^2, 2n-2, n-2, 2)$.*

Proof 2 *Two adjacent vertices of G_n correspond to two squares in the same row or column, so they share $n-2$ neighbours. Furthermore, suppose (a, b) and (c, d) are the coordinates of 2 squares, with $a \neq c, b \neq d$. Then precisely (a, d) and (b, c) are their 2 common neighbours.*

Lemma 1 *Suppose G is $srg(v, k, \lambda, \mu)$. Then the spectrum of G is $\left\{\frac{1}{2}\left(\lambda - \mu \pm \sqrt{(\lambda-\mu)^2 + 4(k-\mu)}\right), k\right\}$ with respective multiplicities $\left\{\frac{1}{2}\left[(v-1) \mp \frac{2k+(v-1)(\lambda-\mu)}{\sqrt{(\lambda-\mu)^2+4(k-\mu)}}\right], 1\right\}$*

Proof 3 *See (7), Lemma 10.2.1.*

Corollary 1 *The spectrum of G_n is $\{-2, n-2, 2n-2\}$, with respective multiplicities $\{n^2 - 2n + 1, 2n - 2, 1\}$.*

B. Mixing Rates of Random Walks on Rook Networks. It is well-known that G is d-regular iff d is the element of largest absolute value in the spectrum of G. Let G be a d-regular graph on n vertices with second largest absolute eigenvalue k If $k < d$, call G a (m, d, k) graph.

Remark 8 *G_n is a (m, d, k) network, with $m = n^2$, $d = 2n-2$, $k = n-2$, because $k < d$.*

The following facts are well-known and can be found in (8). If G is a connected and non-bipartite undirected graph, then the Markov chain $\mathcal{M}$, which corresponds to a random walk on G, is irreducible and aperiodic. In this case, $\mathcal{M}$ converges to a unique stationary distribution, π, where

$$\pi(u) = \frac{deg(u)}{2|E|} \qquad [8]$$

independent of its starting position. (2) also proves that the stationary distribution for a non-backtracking walk on a non-bipartite d-regular graph, $d \geq 3$ converges to a uniform distribution.

Definition 11 *The **mixing rate** of the random walk on G measures how fast $\mathcal{M}$ converges to the stationary distribution. It is defined as follows:*

$$\rho = \rho(G) = \limsup_{k\to\infty} \max_{u,v\in V} |P^k_{uv} - \pi(v)|$$

where

$$P^k_{uv} = \mathbb{P}[X_{t+k} = v | X_t = u] \qquad [9]$$

For a non-backtracking walk, the mixing rate $\tilde{\rho}(G)$ is defined in a similar way, replacing P^k_{uv} with $\tilde{P}^k_{uv}$, the probability that a NBRW of length k, starting at u, ends at v.

C. Mixing Number for Rook Network on an n x n board. It is shown in (8) that for a (m, d, k) graph, $\rho = \frac{k}{d}$. Therefore $\rho(G_n) = \frac{n-2}{2n-2}$. (9) shows how to calculate $\tilde{\rho}$ for an (m, d, k) graph.

Define $\psi(x) : [0, \infty) \longrightarrow \mathbb{R}$ by

$$\psi(x) = \begin{cases} x + \sqrt{x^2 - 1}, & \text{if } x \geq 1. \\ 1, & \text{if } 0 \leq x \leq 1. \end{cases} \qquad [10]$$

Then a non-backtracking random walk on G converges to a uniform distribution, and its mixing rate, $\tilde{\rho}(G)$, satisfies

$$\tilde{\rho}(G) = \psi\left(\frac{k}{2\sqrt{d-1}}\right) / \sqrt{d-1} \qquad [11]$$

The following corollary is due to (9)

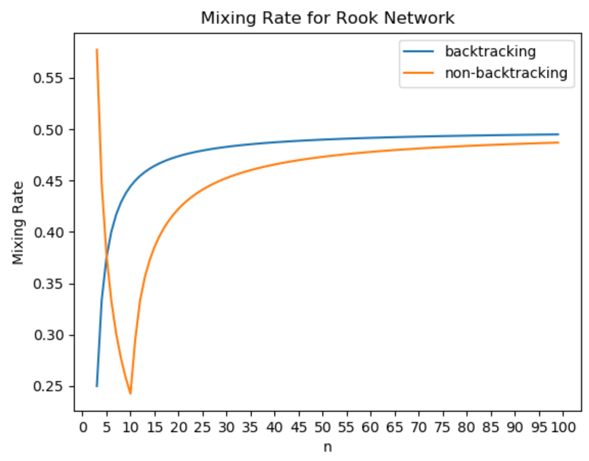
Corollary 2 *Suppose G is a (m, d, k) non-bipartite graph, with $d \geq 3$ and $k \geq 2\sqrt{d-1}$. Then*

$$\frac{d}{2(d-1)} \leq \frac{\tilde{\rho}(G)}{\rho(G)} \leq 1 \qquad [12]$$

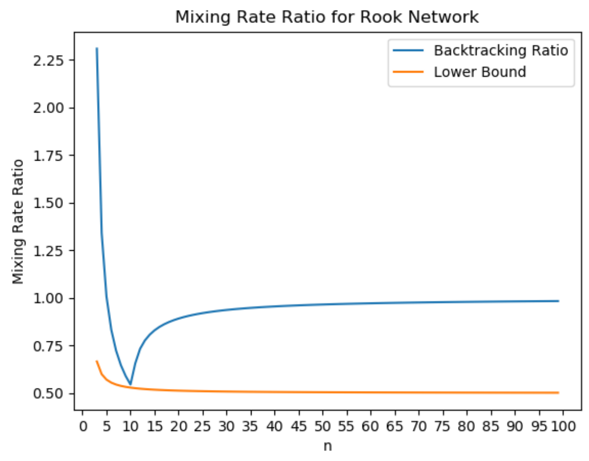
Remark 9 *Apply this result to G_n. Here $d \geq 3$ and $k \geq 2\sqrt{d-1}$ for $n \geq 11$. Explicitly,*

$$\tilde{\rho}(G_n) = \begin{cases} \frac{n-2+\sqrt{n^2-12n+16}}{2(2n-3)}, & \text{if } 11 \leq n. \\ \frac{1}{\sqrt{2n-3}}, & \text{if } 3 \leq n \leq 10. \end{cases} \qquad [13]$$

Figure 10 compares the mixing rate for RW versus NBRW on G_n, and verifies that the lower bound holds.



(a) Mixing Rate for Rook Network



(b) Mixing Rate Ratio: Non-Backtracking to Backtracking

Fig. 10. Mixing for Rook Network

D. Revisiting a Square in a Non-Backtracking Random Walk. Consider the following question: what is the probability that a NBRW on a chess network revisits the same square at least once? (2) gives the following lemma:

Lemma 2 *Let G be a d-regular graph on n vertices, in which each vertex is contained in a cycle of length $g = g(n)$. If $k = k(n)$ satisfies:*

$$k = \frac{\log_{d-1}(n/\log n) - \omega(1)}{g} \qquad [14]$$

then, with high probability, a NBRW of length n on G visits some vertex at least k times. In particular, such a walk almost surely visits some vertex $\Omega\left(\frac{\log n}{g}\right)$ times.

This lemma can be applied to a d-regular graph with small average degree and small girth.

Definition 12 *Define a Slow Jumping Rook to be a chess piece that moves like a Slow Rook, but can also jump to the other side of the board, as if the board were wrapped around a 2-torus. A diagram for the movement of this piece is included in the Supplemental Information.*

Note that for $n \geq 2$, a Slow Jumping Rook network for an $n \times n$ board is a 4-regular graph with n^2 vertices and a girth of 4.

Remark 10 *Applying the above lemma for a Slow Jumping Rook network gives a bound for $k(n)$: $\log_3\left(\frac{n^2}{\log(n^2)}\right) - 4k(n) = \omega(1)$.*

For example, setting $k(n) = 0.48\log_3(n)$ satisfies this equation. Note that $k(100) \approx 2.01 > 2$. Therefore with high probability, a NBRW of length n^2 on a Slow Jumping Rook network with board size $n \geq 100$ will visit the same square at least once.

7. Estimating Clustering Coefficient of Chess Networks Using Non-Backtracking Random Walks

In (3), the following method for estimating the clustering coefficient of a graph from a random walk is given.

- Perform a NBRW of length n on G: $R = (v_0, v_1, ...v_n)$ with $v_{k-1} \neq v_{k+1}$ for $k = 2, ..., n-1$.
- Define ϕ_k for $k = 2, ..., n-1$ to be 1 if v_{k-1} and v_{k+1} are connected, and 0 otherwise.
- Define d_k to be the degree of v_k.
- Define Φ, Ψ to be the following:

$$\Phi = \frac{1}{n-2}\sum_{k=2}^{n-1}\frac{\phi_k}{d_k}, \Psi = \frac{1}{n}\sum_{k=1}^{n}\frac{1}{d_k} \qquad [15]$$

- Then $\hat{C} = \frac{\Phi}{\Psi}$ is an estimator for the clustering coefficient C(G).

Let c_k be the local clustering coefficient of v_k. Firstly it is necessary to find the expected values of Ψ and Φ. Calculation shows that

$\mathbb{E}[\Phi] = \mathbb{E}[\frac{\phi_k}{d_K}] \sim \frac{1}{2|E|}\sum_{k=1}^{n} c_k$

$\mathbb{E}[\Psi] = \mathbb{E}[\frac{1}{d_K}] \sim \frac{n}{2|E|}$

Hence $\hat{C}$ is an unbiased estimator of $C = \bar{c}_k$ As (3) shows, this estimator has a lower variance than a similar estimator which uses backtracking random walks. Figure 11 demonstrates an application of this method to estimate the clustering coefficient of a Queen network on the standard chessboard.

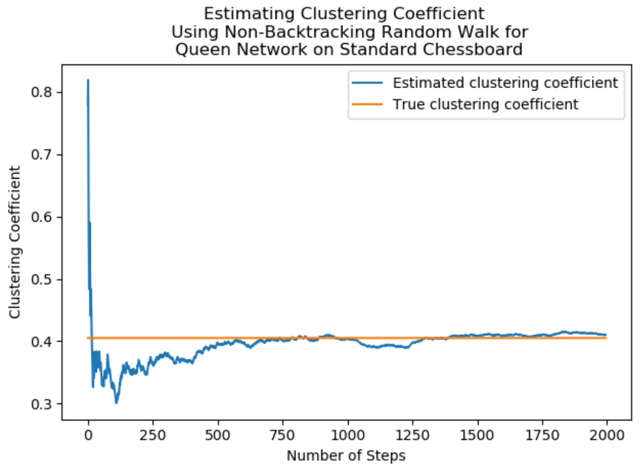


Fig. 11. Using Non-Backtracking Random Walks to estimate Clustering Coefficient

8. Modelling Issues and Extensions

A. Modelling Issues. Two issues with the given definition of random chess networks are that a random chess network might not be connected, or might not have a minimum degree of at least 2. In these cases, either the random walk will not visit every vertex of the random network, or the NBRW will be forced to backtrack. Here are two possible adaptations of the NBRW algorithm which avoid these issues.

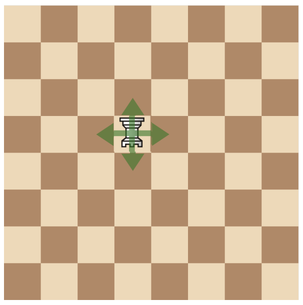
- Suppose G is not connected. Adapt the NBRW so that after the random walk has visited every vertex in the connected component of the starting vertex, the walk "jumps" to an unvisited connected component and continues the walk. The walk ends when every vertex of every connected component has been visited.
- Suppose G is connected but has some vertices with degree 1. Adapt the NBRW so that if the vertex has only one neighbour, that neighbour is selected and the walk continues.

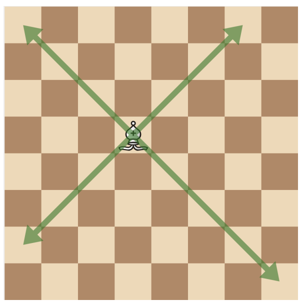
B. Extensions of Chess Networks.

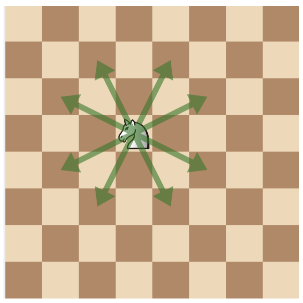
- Consider other chess pieces; for example a piece which can move like a rook and a knight.
- Consider a chessboard wrapped on another surface, e.g. a cylinder.
- Consider a chess network where the chessboard is tiled differently, for example a hexagonal lattice.
- Generalise a chess network to higher dimensions.

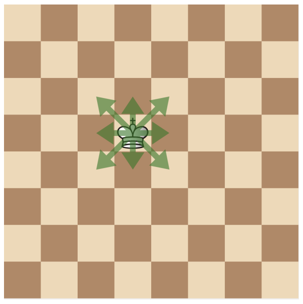
1. Satyadev C (2016) *Kavyalankara of Rudrata (Sanskrit text, with Hindi translation);. Delhitraversal: Parimal Sanskrit Series No. 30.*
2. Alon N, Benjamini I, Lubetzky E, Sodin S (2006) Non-backtracking random walks mix faster.
3. Iwasaki K, Shudo K (2018) Estimating the clustering coefficient of a social network by a non-backtracking random walk - ieee conference publication.
4. Ortner R, Woess W (2004) Non-backtracking random walks and cogrowth of graphs.
5. Watts DJ, Strogatz SH (1998). *Nature.*
6. Aleliunas R, M. Karp R, J. Lipton R, Lovász L, Rackoff C (1979) Random walks, universal traversal sequences, and the complexity of maze problems. pp. 218–223.
7. Godsil C, Royle GF (2013) *Algebraic Graph Theory (Graduate Texts in Mathematics Book 207).* (Springer).
8. Lovász L (1993) Random walks on graphs: A survey, combinatorics, paul erdos is eighty. *Bolyai Soc. Math. Stud.* 2.
9. Kempton M (2016) Non-backtracking random walks and a weighted ihara's theorem.

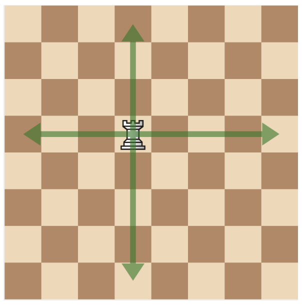
9. Supplemental Information and Additional Diagrams

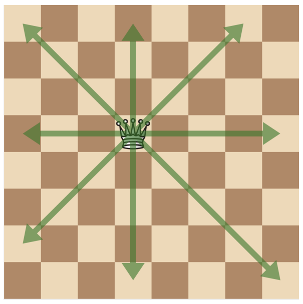

(a) Slow Rook Movement


(b) Bishop Movement


(c) Knight Movement.


(d) King Movement


(e) Rook Movement


(f) Queen Movement

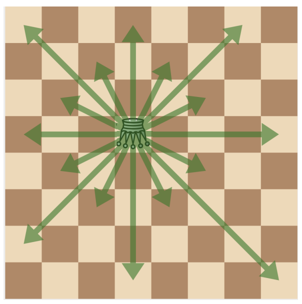
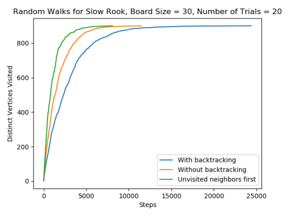
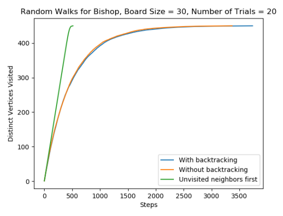

(g) Fairy Movement

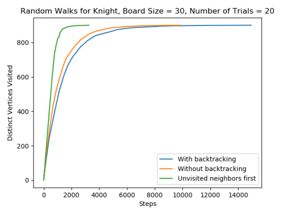
Fig. 12. Movement of Pieces



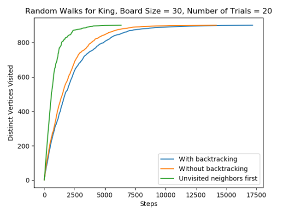
(a) Slow Rook Comparison



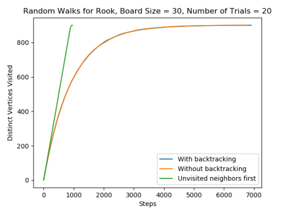
(b) Bishop Comparison



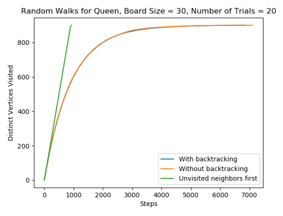
(c) Knight Comparison



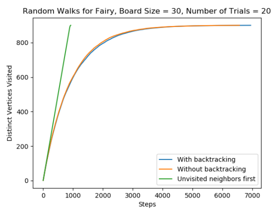
(d) King Comparison



(e) Rook Comparison

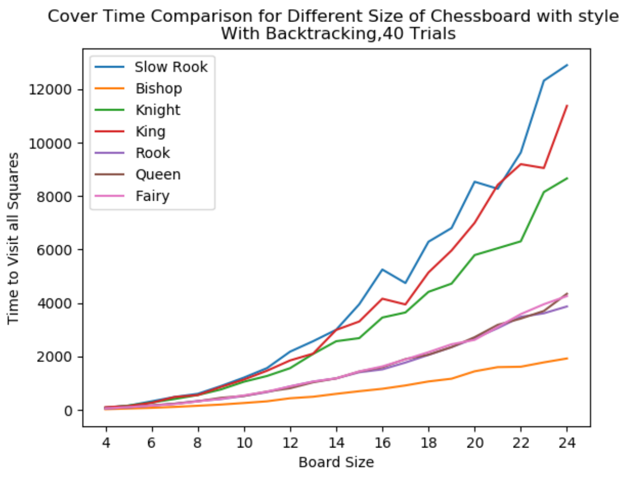


(f) Queen Comparison

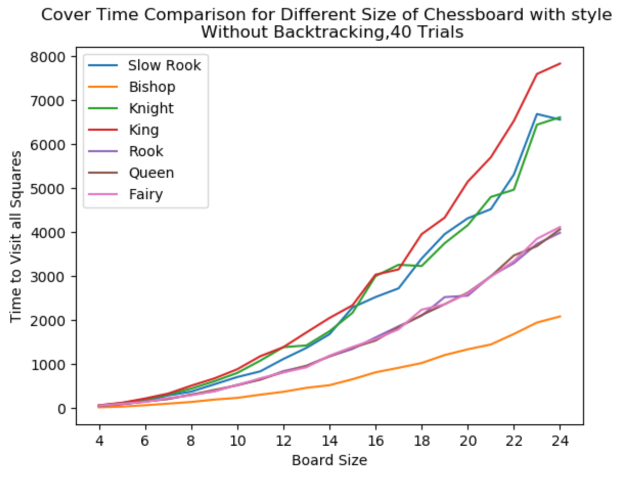


(g) Fairy Comparison

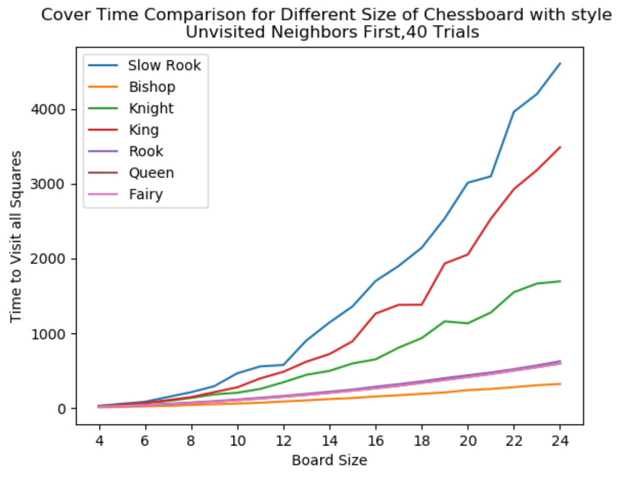
Fig. 13. Random Walk Comparison for 30×30 chessboard, 20 trials.



(a) Backtracking Cover Times



(b) Non-Backtracking Cover Times



(c) Unvisited Neighbour Cover Times

Fig. 14. Cover Times Comparison, 40 Trials

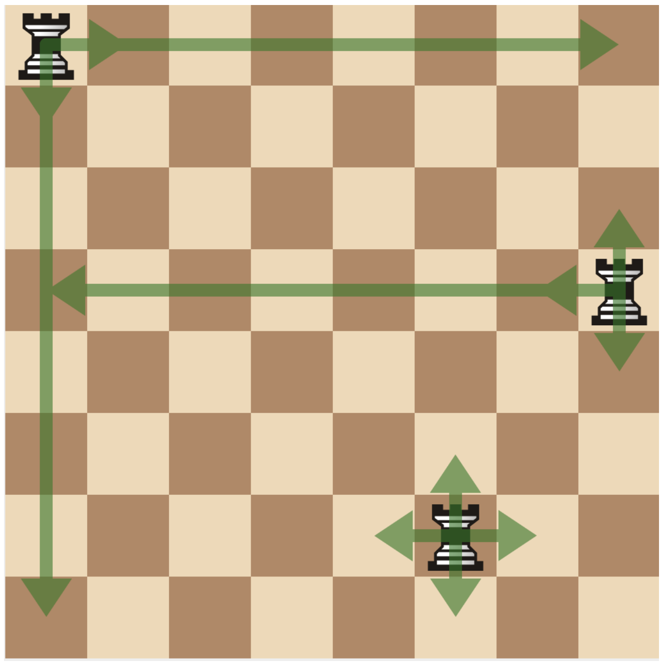


Fig. 15. Slow Jumping Rook

```
import networkx as nx
import chess
import matplotlib.pyplot as plt
import random
import numpy as np
import time
from collections import Counter
import math
def draw_graph(G,title="title"):
    plt.figure (figsize=(15,15))
    pos = nx.spring_layout(G)
    nx.draw(G, pos)
    nx.draw_networkx_labels(G,pos)
    plt.title(title)

    plt.show()
pieces =
["slow_rook","bishop","knight","king","rook
","queen","fairy"]
styles = ["bt","nbt","unvisited"]
def num_to_co(num,size):
    return [num//(size),num%(size)]
def prettify(piece_name):
    a = piece_name.replace("_"," ")
    return a.title()
def co_to_num(co_ordinate,size):
    return
size*co_ordinate[0]+co_ordinate[1]
# print(num_to_coordinate(23,8))
# print(coordinate_to_num([3,3],8))
def valid_rook(co1,co2):
    return
(co1[0]==co2[0])!=(co1[1]==co2[1])
def valid_slow_rook(co1,co2):
    return ((abs(co1[0]-co2[0])==1) and
(abs(co1[1]-co2[1])==0))\
     or ((abs(co1[0]-co2[0])==0) and
(abs(co1[1]-co2[1])==1))

def valid_knight(co1,co2):
    return (abs(co1[0]-co2[0])==1)\
     and (abs(co1[1]-co2[1])==2) \
     or \
     (abs(co1[0]-co2[0])==2)\
     and (abs(co1[1]-co2[1])==1)
def valid_bishop(co1,co2):
    return (co1[0]!=co2[0] or
co1[1]!=co2[1]) and \

(abs(co1[0]-co2[0])==abs(co1[1]-co2[1]))
def valid_king(co1,co2):
    return (co1[0]!=co2[0] or
co1[1]!=co2[1]) and \

max(abs(co1[0]-co2[0]),abs(co1[1]-co2[1]))<
=1
def valid_queen(co1,co2):
    return valid_rook(co1,co2) or
valid_bishop(co1,co2)
def valid_fairy(co1,co2):
    return valid_rook(co1,co2)\
     or valid_bishop(co1,co2)\
     or valid_knight(co1,co2)
def random_sample(n,prob):
    list = []
    for i in range(n):
        if random.random()<=prob:
            list.append(i)
    return list
def chess_network(piece,size,prob):

    squares = random_sample(size**2,prob)
    # squares = full
    edges = []
    if piece=="knight":
        for s1 in squares:
            for s2 in squares:
                if
valid_knight(num_to_co(s1,size),num_to_co(s
2,size)):
                    edges.append([s1,s2])
    elif piece=="rook":
        for s1 in squares:
            for s2 in squares:
                if
valid_rook(num_to_co(s1,size),num_to_co(s2,
size)):
                    edges.append([s1,s2])
    elif piece=="slow_rook":
        for s1 in squares:
            for s2 in squares:
                if
valid_slow_rook(num_to_co(s1,size),num_to_c
o(s2,size)):
                    edges.append([s1,s2])
    elif piece=="bishop":
        for s1 in squares:
            for s2 in squares:
                if
valid_bishop(num_to_co(s1,size),num_to_co(s
2,size)):
                    edges.append([s1,s2])
    elif piece=="queen":
        for s1 in squares:
            for s2 in squares:
                if
valid_queen(num_to_co(s1,size),num_to_co(s2
,size)):
                    edges.append([s1,s2])
    elif piece=="king":
        for s1 in squares:
            for s2 in squares:
                if
valid_king(num_to_co(s1,size),num_to_co(s2,
size)):
                    edges.append([s1,s2])
    elif piece=="fairy":
        for s1 in squares:
            for s2 in squares:
                if
valid_fairy(num_to_co(s1,size),num_to_co(s2
,size)):
                    edges.append([s1,s2])

    G = nx.Graph()
    G.add_nodes_from(squares)
    G.add_edges_from(edges)
    return G
def rwalk(G,style,n):
    route = []
    current_steps = 0
    current_visited_squares = 0
    steps = []
    visited_squares = []
    num_visited_squares = []
    new_reached = []
    degs = []
    start = random.choice(list(G))
    size_cc =
find_size_connected_component(G,start)
    current = start
    previous = start
    most_visited = []
    least_visited = []
    # while len(visited_squares)<size_cc:
    while current_steps<n:
        route.append(current)

most_visited.append(most_common(route))

least_visited.append(least_common(route))
        degs.append(G.degree(current))
        current_steps+=1
        steps.append(current_steps)
        if current not in visited_squares:
            visited_squares.append(current)
            current_visited_squares+=1

num_visited_squares.append(current_visited_
squares)


new_reached.append(current_steps)

        if style == "bt":
            current = random.choice([n for
n in G.neighbors(current)])
        elif style == "nbt":
            poss = [n for n in
G.neighbors(current)]
            if len(steps)>1 and
len(poss)>1:
                poss.remove(previous)
            previous = current
            current = random.choice(poss)
        elif style == "unvisited":
            neighbors = [n for n in
G.neighbors(current)]
            unvisited_neighbors = \
            list(filter(lambda x: x not in
visited_squares,\
                neighbors))
            if len(unvisited_neighbors)>0:
                current =
random.choice(unvisited_neighbors)
            else:
                current =
random.choice(neighbors)
        else:
            return ["no","answer"]
        phi = []
        for i in range(1,len(route)-1):
            if [route[i-1],route[i+1]] in
G.edges():
                phi.append(1)
            else:
                phi.append(0)
    return[steps,most_visited]
def
piece_comparer(pieces,size,num_trials,num_g
aps):
    probs = np.linspace(0.0,1.0,
num=num_gaps)
    for piece in pieces:
        out = []
        for prob in probs:
            prob_data = 0.0
            for i in range(num_trials):
                G =
chess_network(piece,size,prob)
                if len(G)>0:
                    if nx.is_connected(G)
and min_degree(G)>=2:

                        prob_data+=1

out.append(prob_data/num_trials)
        print(out)
        plt.plot(probs,out,label =
prettify(piece))
    plt.xlabel("p")
    plt.ylabel("Probability for
Non-Backtracking Random Walk")
    plt.legend(bbox_to_anchor=(1.05, 1),
loc=2, borderaxespad=0.)

    plt.title("Ability to Cover Standard
8x8 Board, \n \
        20 Trials Using Non-Backtracking
Random Walk\
        \n as p changes")
    plt.tight_layout()

    plt.show()
def find_size_connected_component(G,r):

    ccs = nx.connected_components(G)
    for cc in ccs:
        if r in cc:
            return(len(cc))
def
find_edge_size_connected_component(G,r):

    ccs = nx.connected_components(G)
    for cc in ccs:
        if r in cc:
            H = G.subgraph(cc)
            return len(H.edges)
def avg_degree(G):
    return np.mean([x[1] for x in
G.degree()])
```

Fig. 16. Python Code used for Project

```
def rwalk_edges(G,style):
    route = []
    current_steps = 0
    current_visited_squares = 0
    steps = []
    visited_squares = []
    num_visited_squares = []
    new_reached = []
    visited_edges = []

    start = random.choice(list(G))
    cc_edges =
find_edge_size_connected_component(G,start)
    current = start
    previous = start
    while len(visited_edges)<cc_edges:
        current_edge = [previous,current]
        route.append(current)
        current_steps+=1
        steps.append(current_steps)
        if current_edge not in
visited_edges:
            # print(current_edge)

visited_edges.append(current_edge)
            current_visited_squares+=1

num_visited_squares.append(current_visited_
squares)


new_reached.append(current_steps)

        if style == "bt":
            previous = current
            current = random.choice([n for
n in G.neighbors(current)])
        elif style == "nbt":
            poss = [n for n in
G.neighbors(current)]
            if len(steps)>1 and
len(poss)>1:
                poss.remove(previous)
            previous = current
            current = random.choice(poss)
        elif style == "unvisited":
            previous = current
            neighbors = [n for n in
G.neighbors(current)]
            unvisited_neighbors = \
            list(filter(lambda x: x not in
visited_squares,\
                neighbors))
            if len(unvisited_neighbors)>0:
                current =
random.choice(unvisited_neighbors)
            else:
                current =
random.choice(neighbors)
        else:
            return ["no","answer"]
    return current_steps
def
cover_time(pieces,size,prob,style,num_trial
s):
    start = time.time()
    for piece in pieces:
        sizes = []
        size_data = []
        for n in range(4,size):
            G = chess_network(piece,n,prob)
            data_for_size = []
            for i in range(num_trials):

data_for_size.append(rwalk_edges(G,style))
            sizes.append(n)

size_data.append(np.mean(data_for_size))
        plt.plot(sizes,size_data,label =
prettify(piece))
def d(n):
    return 2*n-2.0
def k(n):
    return n-2.0
def rho(n):
    return k(n)/d(n)
def psi(x):
    if 0<=x and x<=1:
        return 1
    elif x>1:
        return x+0.0+((x**2-1)**0.5)
def rho_tilde(n):
    t1 = k(n)/(2*((d(n)-1)**0.5))
    t2 = ((d(n)-1)**0.5)
    return psi(t1)/t2
def cc_estimate(degs,phi):
    weights = []
    avgs = []
    w2 = []
    psi = []
    ratio = []
    true = []
    for i in degs:
        w2.append(1.0/i)
    for i in range(len(w2)-2):
        psi.append(np.mean(w2[:i+2]))
    for i in range(len(phi)):

weights.append((phi[i]+0.0)/degs[i+2])
    for i in range(len(w2)-2):
        avgs.append(np.mean(weights[:i+2]))


    for i in
range(3,min(len(avgs),len(psi))):
        ratio.append(avgs[i]/psi[i])
    # return w2,psi,weights,avgs
    for i in ratio:
        true.append(actual)
    plt.plot(ratio,label = "Estimated
clustering coefficient")
    plt.plot(true,label = "True clustering
coefficient")
    plt.legend()
    plt.title("Estimating Clustering
Coefficient \n Using Non-Backtracking
Random Walk for\n Queen Network on
Standard Chessboard")
    plt.xlabel("Number of Steps")
    plt.ylabel("Clustering Coefficient")
    plt.tight_layout()

    plt.show()
def avg_random_walk(G,style,n):
    d1 = []
    d2 = []
    for i in range(n):
        results = rwalk(G,style)

        d1.append(results[0])
        d2.append(results[1])
    return
[np.mean(d2,axis=0),np.mean(d1,axis=0)]
def rwalk_chess(piece,size,prob,style):
    G = chess_network(piece,size,prob)
    out =  rwalk(G,style)
    plt.plot(out[0],out[1])
def
rwalk_compare(piece,size,prob,num_trials):
    G = chess_network(piece,size,prob)
    for style in styles:
        out =
avg_random_walk(G,style,num_trials)

plt.plot(out[0],out[1],label=long_style(sty
le))
    my_title = "Random Walks for " +
prettify(piece)+", Board Size = "\
        + str(size ) + ", Number of Trials
= " + str(num_trials)
    plt.title(my_title)
    plt.xlabel('Steps')
    plt.ylabel('Distinct Vertices Visited')
    plt.legend()
    plt.show()
def rw_compare_bt(piece,size,prob):
    not_found = True
    counter = 0
    G = nx.Graph
    while counter<100 and not_found:
        counter+=1
        H = chess_network(piece,size,prob)
        if nx.is_connected(H) and
min_degree(H)>=2:
            not_found = False
            print("found me",len(H))
            G = H
    d1 = rwalk(G,True)
    d2 = rwalk(G,False)

plt.plot(d1[0],d1[1],'r',d2[0],d2[1],'b')
    plt.show()
def
piece_graph(piece,max_size,prob,metric):
    sizes = []
    ccs = []
    avg_degs = []
    if metric == "cc":
        for i in range(1,max_size):
            sizes.append(i)
            G = chess_network(piece,i,prob)
            # G =nx.complete_graph(4)

ccs.append(nx.average_clustering(G))
            #
             avg_degs.append(avg_degree(G)
             )
        plt.xticks(np.arange(min(sizes),
max(sizes)+2, 2.0))


plt.plot(sizes,ccs,label=prettify(piece))
    elif metric == "avg_degree":
        for i in range(1,max_size):
            sizes.append(i)
            G = chess_network(piece,i,prob)
            # G =nx.complete_graph(4)


            #
             ccs.append(nx.average_cluster
             ing(G))
            avg_degs.append(avg_degree(G))
        plt.xticks(np.arange(min(sizes),
max(sizes)+2, 2.0))

plt.plot(sizes,avg_degs,label=prettify(piec
e))
```

Fig. 17. Python Code used for Project