algorithms

Article

Linking and Cutting Spanning Trees

Luís M. S. Russo *, Andreia Sofia Teixeira and Alexandre P. Francisco

INESC-ID and the Department of Computer Science and Engineering, Instituto Superior Técnico, Universidade de Lisboa, Avenida Rovisco Pais 1, 1049-001 Lisboa, Portugal; sofia.teixeira@tecnico.ulisboa.pt (A.S.T.); aplf@tecnico.ulisboa.pt (A.P.F.)

* Correspondence: luis.russo@tecnico.ulisboa.pt; Tel.: +351-21-310-0272

Received: 12 March 2018; Accepted: 11 April 2018; Published: 19 April 2018

Abstract: We consider the problem of uniformly generating a spanning tree for an undirected connected graph. This process is useful for computing statistics, namely for phylogenetic trees. We describe a Markov chain for producing these trees. For cycle graphs, we prove that this approach significantly outperforms existing algorithms. For general graphs, experimental results show that the chain converges quickly. This yields an efficient algorithm due to the use of proper fast data structures. To obtain the mixing time of the chain we describe a coupling, which we analyze for cycle graphs and simulate for other graphs.

Keywords: spanning tree; uniform generation; Markov chain; mixing time; link-cut tree

PACS: 02.10.Ox; 02.50.Ga; 02.50.Ng; 02.70.Uu

MSC: 05C81; 05C85; 60J10; 60J22; 65C40; 68R10

1. Introduction

A spanning tree A of an undirected connected graph G is a connected set of edges without cycles that spans every vertex of G. Every vertex of G occurs at an edge of A. Figure 1 shows an example. The vertices of the graph are represented by circles. The set of vertices is denoted by V. The edges of G are represented by dashed lines, and the set of edges is represented by E. The edges of the spanning tree A are represented by thick gray lines. We also use V and E to denote the size of the set V and the size of set E, respectively, i.e., the number of vertices and the number of edges. In case the expression can be interpreted as a set instead of a number, we avoid the ambiguity by writing $|V|$ and $|E|$, respectively.

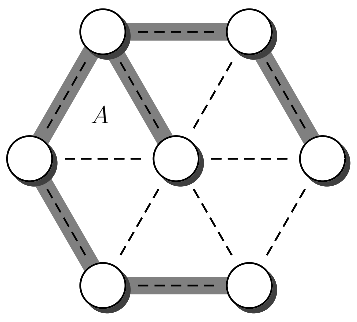

Figure 1. A Spanning tree A over a graph G.

We aim to compute one such spanning tree, A, uniformly among all possible spanning trees. The number of these trees may vary significantly, from 1 to V^{V-2}, depending on the underlying graph ([1–3], Chapter 22). Computing such a tree uniformly and efficiently is challenging for several reasons: the number of such trees is usually exponential; and the structure of the resulting trees is largely heterogeneous as the underlying graphs change. The contributions of this paper are as follows:

- We present a new algorithm, which given a graph G, generates a spanning tree of G uniformly at random. The algorithm uses the link-cut tree data structure to compute randomizing operations in $O(\log V)$ amortized time per operation. Hence, the overall algorithm takes $O(\tau \log V)$ time to obtain a uniform spanning tree of G, where τ is the mixing time of a Markov chain that is dependent on G. Theorem 1 summarizes this result.
- We propose a coupling to bound the mixing time τ. The analysis of the coupling yields a bound for cycle graphs (Theorem 2), and for graphs which consist of simple cycles connected by bridges or articulation points (Theorem 3). We also simulate this procedure experimentally to obtain bounds for other graphs. The link-cut tree data structure is also key in this process. Section 4.3 shows experimental results, including other classes of graphs.

This paper is structured as follows. In Section 2 we introduce the problem and explain its subtle nature. In Section 3 we explain our approach and point out that using the link-cut tree data structure is much faster than repeating depth first searches (DFSs). In Section 4 we thoroughly justify our results, proving that the underlying Markov chain has the necessary properties and providing experimental results of our algorithm. In Section 5 we describe the related work concerning random spanning trees, link-cut trees, and mixing times of Markov chains. In Section 6 we present our conclusions.

2. The Challenge

We start by describing an intuitive process for generating spanning trees that does not obtain a uniform distribution. It produces some trees with a higher probability than others. This serves to illustrate that the problem is more difficult than it may seem at first glance. Moreover, we explain why this process is biased, using a counting argument.

A simple procedure to build A consists in using a union-find data structure [4] to guarantee that A does not contain a cycle. Note that these structures are strictly incremental, meaning that they can be used to detect cycles but can not be used to remove an edge from the cycle. Therefore the only possible action is to discard the edge that creates the cycle.

Let us analyze a concrete example of the resulting distribution of spanning trees. We shall show that this distribution is not uniform. First, generate a permutation p of E and then process the edges in this order. Each edge that does not produce a cycle is added to A, edges that would otherwise produce cycles are discarded, and the procedure continues with the next edge in the permutation.

Consider the complete graph on four vertices, K_4, and focus on the probability of generating a star graph, centered at the vertex labeled 1. Figure 2 illustrates the star graph. The K_4 graph has 6 edges, hence there are $6! = 720$ different permutations. To produce the star graph, from one such permutation, it is necessary that the edges $(1,2)$ and $(1,3)$ are selected before the edge $(2,3)$ appears; in general the edges $(1,u)$ and $(1,v)$ must occur before (u,v). One permutation that generates the star graph is $(1,2),(1,3),(2,3),(1,4),(2,4),(3,4)$. Now, $(2,3)$ can be moved to the right to any of three different locations, so we know four sequences that generate the star graph. The same reasoning can be applied to $(2,4)$ which can be moved once to the right. In total we count 8 different sequences that generate the star graph, centered at 1. For each of these sequences it is possible to permute the vertices $2,\ 3,\ 4,$ amongst themselves. Hence, we multiply the previous count by $3! = 6$. In total we count $48 = 8 \times 6$ sequences that generate the star graph, and therefore the total probability of obtaining a star graph is $48/6! = 1/15$. According to Cayley's formula, the probability of obtain the star graph centered at 1 should be $1/4^2 = 1/16$. Hence, too many sequences generate the star graph centered at 1.

In the next section we fix this bias by discarding some edge in the potential cycle, not necessarily the edge that creates it.

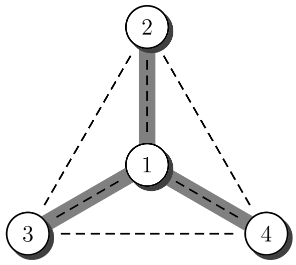

Figure 2. A star graph on K_4, centered at 1.

3. Main Idea

To generate a uniform spanning tree, start by generating an arbitrary spanning tree A. One way to obtain this tree is to compute a depth first search in G, in which case the necessary time is $O(V+E)$. In general, we want the mixing time of our chain to be much smaller than $O(E)$, especially for dense graphs. This initial tree is only generated once, and subsequent trees are obtained by the randomizing process. To randomize A, repeat the next process several times. Choose an edge (u,v) from E uniformly at random, and consider the set $A \cup \{(u,v)\}$. If (u,v) already belongs to A the process stops, otherwise $A \cup \{(u,v)\}$ contains a cycle C. To complete the process, choose an edge (u',v') uniformly from $C \setminus \{(u,v)\}$ and remove it. Hence, at each step the set A is transformed into the set $(A \cup \{(u,v)\}) \setminus \{(u',v')\}$. An illustration of this process is shown in Figure 3.

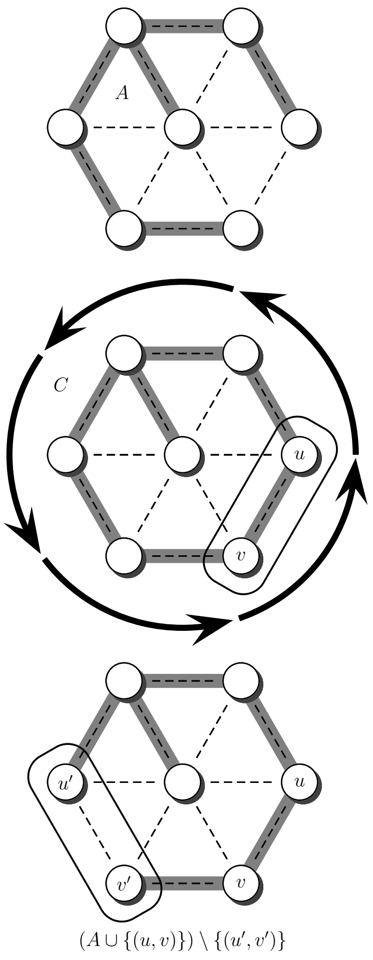

Figure 3. Edge-swap procedure. Inserting the edge (u,v) into the initial tree A generates a cycle C. The edge (u',v') is removed from C.

This edge-swapping process can be adequately modeled by a Markov chain, where the states corresponds to different spanning trees and the transitions among states correspond to the process we have just described. In Section 4.1 we study the ergodic properties of this chain. For now let us focus on which data structures can be used to compute the transition procedure efficiently. A simple solution to this problem would be to compute a depth first search (DFS) on A, starting at u and terminating whenever v is reached. This allows us to identify C in $O(V)$ time. Recall that A contains exactly $V-1$ elements. The edge (u', v') can then be easily removed. Besides G the elements of A also need to be represented with the adjacency list data structure. For our purposes, this approach is inefficient. This computation is central to our algorithm and its complexity becomes a factor in the overall performance. Hence, we will now explain how to perform this operation in only $O(\log V)$ amortized time with the link-cut tree data structure.

The link-cut tree (LCT) is a data structure that can used to represent a forest of rooted trees. The representation is dynamic so that edges can be removed and added. Whenever an edge is removed, the original tree is cut in two. Adding an edge between two trees links them. This structure was proposed by Sleator and Tarjan [5]. Both the link and cut operations can be computed in $O(\log V)$ amortized time.

The LCT can only represent trees; therefore the edge swap procedure must first cut the edge (u', v') and afterwards insert the edge (u, v) with the `Link` operation. The randomizing process needs to identify C and select (u', v') from it. The LCT can also compute this process in $O(\log V)$ amortized time. The LCT works by partitioning the represented tree into disjoint paths. Each path is stored in an auxiliary data structure, so that any of its edges can be accessed efficiently in $O(\log V)$ amortized time. To compute this process we force the path $D = C \setminus \{(u, v)\}$ to become a disjoint path. This means that D will be completely stored in one auxiliary data structure. Hence, it is possible to efficiently select an edge from it. Moreover the size of D can also be computed efficiently. The exact process, to force D into an auxiliary structure, is to make u the root of the represented tree and then access v. Algorithm 1 shows the pseudo-code of the edge-swapping procedure. We can confirm, by inspection, that this process can be computed in the $O(\log V)$ amortized time bound that is crucial for our main result.

Algorithm 1 Edge-swapping process

```
1:  procedure EdgeSwap($A$)                    ▷ $A$ is an LCT representation of the current spanning tree
2:      $(u, v) \leftarrow$ Chosen uniformly at random from $E$
3:      if $(u, v) \notin A$ then                                              ▷ $O(\log V)$ time
4:          ReRoot($A, u$)                                              ▷ Makes $u$ the root of $A$
5:          $D \leftarrow$ Access($A, v$)                   ▷ Obtains a representation of the path $C \setminus \{(u, v)\}$
6:          $i \leftarrow$ Chosen uniformly from $\{1, \ldots, |D|\}$
7:          $(u', v') \leftarrow$ Select($D, i$)                          ▷ Obtain the $i$-th edge from $D$
8:          Cut($A, u', v'$)
9:          Link($A, u, v$)
10:     end if
11: end procedure
```

Theorem 1. *If G is a graph and A is a spanning tree of G, then a spanning tree A' can be chosen uniformly from all spanning trees of G in $O((V + \tau)\log V)$ time, where τ is the mixing time of an ergodic edge-swapping Markov chain.*

In Section 4.1 we prove that the process we described is indeed an ergodic Markov chain, thus establishing the result. We finish this section by pointing out a detail in Algorithm 1. In the comment in line 3 we point out that the property $(u, v) \notin A$ must be checked in at most $O(\log V)$ time. This can be achieved in $O(1)$ time by keeping an array of Booleans indexed by E. Moreover it can also be achieved in $O(\log V)$ amortized time by using the LCT data structure, essentially by delaying the verification until D is determined and verifying if $|D| \neq 1$.

4. The Details

4.1. Ergodic Analysis

In this section, we analyze the Markov chain M_t induced by the edge-swapping process. It should be clear that this process has the Markov property because the probability of reaching a state depends only on the previous state. In other words the next spanning tree depends only on the current tree.

To prove that our procedure is correct we must show that the stationary distribution is uniform for all states. Let us first establish that such a stationary distribution exists. Note that, for a given finite graph G, the number of spanning trees is also finite. More precisely, for complete graphs, Cayley's formula yields V^{V-2} spanning trees. This value is an upper bound for other graphs, as all spanning trees of a certain graph are also spanning trees of the complete graph with the same number of vertices. Therefore, the chain is finite. If we show that it is irreducible and aperiodic, it follows that it is ergodic ([6], Corollary 7.6) and therefore it has a stationary distribution ([6], Theorem 7.7).

The chain is aperiodic because self-loops may occur, i.e., transitions where the underlying state does not change. Such transitions occur when (u,v) is already in A, and therefore their probability is at least $(V-1)/E$, because there are $V-1$ edges in a spanning tree A.

To establish that the chain is irreducible it is sufficient to show that for any pair of states i and j there is a non-zero probability path from i to j. First note that the probability of any transition on the chain is at least $1/(EV)$, because (u,v) is chosen out of E elements and (u',v') is chosen from $C \setminus \{(u,v)\}$, that contains at most $V-1$ edges. To obtain a path from i to j let A_i and A_j represent the respective trees. We consider the following cases:

- If $i = j$ we use a self-loop transition.
- Otherwise, when $i \neq j$, it is possible to choose (u,v) from $A_j \setminus A_i$, and (u',v') from $(C \setminus \{(u,v)\}) \cap (A_i \setminus A_j) = C \setminus A_j$; note that the set equality follows from the assumption that (u,v) belongs to A_j. For the last property, note that if no such (u',v') exists then $C \subseteq A_j$, which is a contradiction because A_j is a tree and C is a cycle. As mentioned above, the probability of this transition is at least $1/(EV)$. After this step the resulting tree is not necessarily A_j, but it is closer to that tree. More precisely $(A_i \cup \{(u,v)\}) \setminus \{(u',v')\}$ is not necessarily A_j, however the set $A_j \setminus ((A_i \cup \{(u,v)\}) \setminus \{(u',v')\})$ is smaller than the original $A_j \setminus A_i$. Its size decreases by 1 because the edge (u,v) exists on the second set but not on the first. Therefore, this process can be iterated until the resulting set is empty and therefore the resulting tree coincides with A_j. The maximal size of $A_j \setminus A_i$ is $V-1$, because the size of A_j is at most $V-1$. This value occurs when A_i and A_j do not share edges. Multiplying all the probabilities in the process of transforming A_i into A_j we obtain a total probability of at least $1/(EV)^{V-1}$.

Now that the stationary distribution is guaranteed to exist, we will show that it coincides with the uniform distribution by proving that the chain is time-reversible ([6], Theorem 7.10). We prove that for any pair of states i and j, with $j \neq i$, for which there exists a transition from i to j, with probability $P_{i,j}$, there exists, necessarily, a transition from j to i with probability $P_{j,i} = P_{i,j}$. If the transition from i to j exists it means that there are edges (u,v) and (u',v') such that $(A_i \cup \{(u,v)\}) \setminus \{(u',v')\} = A_j$, where (u',v') belongs to the cycle C contained in $A_i \cup \{(u,v)\}$. Hence, we also have that $(A_j \cup \{(u',v')\}) \setminus \{(u,v)\} = A_i$, which means that the tree A_i can be obtained from the tree A_j by adding the edge (u',v') and removing the edge (u,v). In other words, the process in Figure 3 is similar both top down or bottom up. This process is a valid transition in the edge-swap chain, where the cycle C is the same in both transitions, i.e., C is the cycle contained in $A_i \cup \{(u,v)\}$ and in $A_j \cup \{(u',v')\}$. Now we obtain our result by observing that $P_{i,j} = 1/(E(C-1)) = P_{j,i}$. In the transition from i to j, the factor $1/E$ comes from the choice of (u,v) and the factor $1/(C-1)$ from the choice of (u',v'). In the transition between j to i, the factor $1/E$ comes from the choice of (u', v') and the factor $1/(C-1)$ from the choice of (u, v). Hence, we established that the algorithm we propose correctly generates spanning trees uniformly, provided we can sample from the stationary distribution. Hence, we need to determine the mixing time of the chain, i.e., the number of edge swap operations that need to be performed on an initial tree until the distribution of the resulting trees is close enough to the stationary distribution.

Before analyzing the mixing time of this chain we point out that it is possible to use a faster version of this chain by choosing (u, v) uniformly from $E \setminus A$, instead of from E. This makes the chain faster, but proving that it is aperiodic is trickier. In this chain we have that $\Pr(M_{t+1} = i | M_t = i) = 0$, for any state i. We will now prove that $\Pr(M_{t+s} = i | M_t = i) \neq 0$, for any state i and $s > 1$. It is enough to show for $s = 2$ and $s = 3$, all other values follow from the fact that the greatest common divisor of 2 and 3 is 1. For the case of $s = 2$ we use the time reverse property and the following deduction: $\Pr(M_{t+2} = i | M_t = i) \geq P_{i,j} P_{j,i} \geq (1/EV)^2 > 0$. For the case of $s = 3$ we observe that the cycle C must contain at least three edges (u, v), (u', v') and (u'', v''). To obtain A_j we insert (u, v) and remove (u', v'); now we move from this state to state A_k by inserting (u'', v'') and removing (u, v). Finally we move back to A_i by inserting (u', v') and removing (u'', v''). Hence, for this case we have $\Pr(M_{t+3} = i | M_t = i) \geq (1/EV)^3 > 0$.

4.2. Coupling

In this section, we focus on bounding the mixing time. We did not obtain general analytical bounds from existing analysis techniques, such as couplings [6,7], strong stopping times [7] and canonical paths [8]. The coupling technique yielded a bound only for cycle graphs and moreover a simulation of the resulting coupling converges for ladder graphs.

Before diving into the reasoning in this section, we first need a finer understanding of the cycles generated in our process. We consider a closed walk to be a sequence of vertices $v_0, \ldots, v_n = v_0$, starting and ending at the same vertex, such that any two consecutive vertices v_i and v_{i+1} are adjacent; in our case $(v_i, v_{i+1}) \in A \cup \{(u, v)\}$. The cycles we consider are simple in the sense that they consist of a set of edges for which a closed walk can be formed that traverses all the edges in the cycle, and moreover no vertex repetitions are allowed, except for the vertex v_0, which is only repeated at the end. Formally this can be stated as: if $0 \leq i, j < n$ and $i \neq j$, then $v_i \neq v_j$.

The cycles that occur in our randomizing process are even more regular. A cordless cycle in a graph is a cycle such that no two vertexes of the cycle are connected by an edge that does not itself belong to the cycle. The cycles we produce also have this property, otherwise if such a chord existed then it would form a cycle on our tree A, which is a contradiction. In fact, a spanning tree over a graph can alternatively be defined as a set of edges such that for any pair of vertices v and v' there is exactly one path linking v to v'.

A coupling is an association between two copies of the same Markov chain X_t and Y_t, in our case the edge-swapping chain. The goal of a coupling is to make the two chains meet as fast as possible, i.e., to obtain $X_\tau = Y_\tau$, for a small value of τ. At this point we say that the chains have coalesced. The two chains may share information and cooperate towards this goal. However, when analyzed in isolation, each chain must be indistinguishable from the original chain M_t. Obtaining $X_\tau = Y_\tau$ with a high probability implies that at time τ the chain is well mixed. Precise statements of these claims are given in Section 5.

We use the random variable X_t to represent the state of the first chain, at time t. The variable Y_t represents the state of the second chain. We consider the chain X_t in state x and the chain Y_t in state y. In one step, the chain X_t will transition to the state $x' = X_{t+1}$ and the chain Y_t will transition to state $y' = Y_{t+1}$.

The set $A_x \setminus A_y$ contains the edges that are exclusive to A_x and likewise the set $A_y \setminus A_x$ contains the edges that are exclusive to A_y. The number of such edges provides a distance $d(x, y) = |A_x \setminus A_y| = |A_y \setminus A_x|$ that measures how far apart the two states are. We refer to this distance as the edge distance. We define a coupling when $d(x, y) \leq 1$, which can be extended for states x and y that are farther apart by using the path-coupling technique [9].

To use the path-coupling technique we cannot alter the behavior of the chain X_t as, in general, it is determined by the previous element in the path. We denote by i_x the edge that gets added to A_x, and by o_x the edge that gets removed from the corresponding cycle $C_x \subseteq A_x \cup \{i_x\}$, in the case such a cycle exists. Likewise, i_y represents the edge that is inserted into A_y and o_y the edge that gets removed from the corresponding cycle $C_y \subseteq A_y \cup \{i_y\}$, in the case such a cycle exists. The edge i_x is chosen uniformly at random from E and o_x is chosen uniformly at random from $C_x \setminus \{i_x\}$. The edges i_y and o_y will be obtained by trying to mimic the chain X_t, but still exhibiting the same behavior as M_t. In this sense the information flows from X_t to Y_t. Let us now analyze $d(x, y)$.

4.2.1. $d(x, y) = 0$

If $d(x, y) = 0$ then $x = y$, which means that the corresponding trees are also equal, $A_y = A_x$. In this case Y_t uses the same transition as X_t, by inserting i_x, i.e., set $i_y = i_x$, and by removing o_x, i.e., set $o_y = o_x$.

4.2.2. $d(x, y) = 1$

If $d(x, y) = 1$ then the edges $e_x \in A_x \setminus A_y$ and $e_y \in A_y \setminus A_x$ exist and are distinct. We also need the following sets: $I = C_x \cap C_y$, $E_x = C_x \setminus I$ and $E_y = C_y \setminus I$. The set I represents the edges that are common to C_x and C_y. The set E_x represents the edges that are exclusive to C_x, from the cycle point of view. This should not be confused with e_x which represents the edge that is exclusive to A_x, i.e., from a tree point of view. Likewise, E_y represents the edges that are exclusive to C_y. Also we consider the cycle C_e as the cycle contained in $A_x \cup \{e_y\}$, which necessarily contains e_x. The following Lemma describes the precise structure of these sets.

Lemma 1. *When $i_y = i_x$ we either have $C_x = C_y = I$ and therefore $E_x = E_y = \emptyset$, or E_x, E_y and I form simple paths, and the following properties hold:*

- $e_x \in E_x$, $e_y \in E_y$, $i_x \in I$
- $E_x \cap E_y = \emptyset$, $E_x \cap I = \emptyset$, $E_y \cap I = \emptyset$
- $E_x \cup I = C_x$, $E_y \cup I = C_y$, $E_x \cup E_y = C_e$.

Notice that in particular this means that, in the non-trivial case, E_x and E_y partition C_e. A schematic representation of this Lemma is shown in Figure 4.

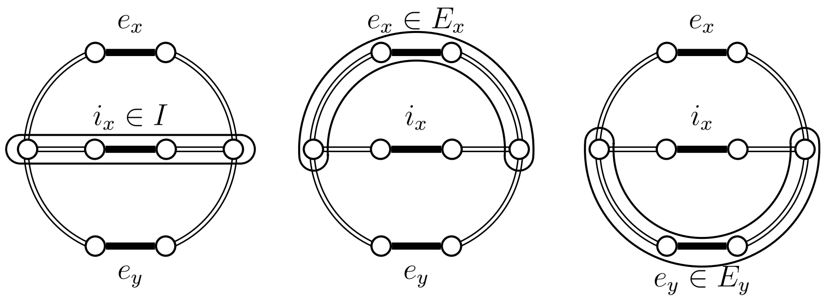

Figure 4. Schematic representation of the relations between E_x, E_y, and I.

We have several different cases described below. Aside from the lucky cases 2 and 3, we will usually choose $i_y = i_x$, as Y_t tries to copy X_t. Likewise, if possible, we would like to set $o_y = o_x$. When this is not possible we must choose $o_y \in E_y$, ideally we would choose $o_y = e_y$, but we must be extra careful with this process to avoid losing the behavior of M_t. To maintain this behaviour, we must sometimes choose $o_y \in E_y \setminus \{e_y\}$. Since X_t provides no information on this type of edges, we use

$o_y = s_y$ chosen uniformly from this $E_y \setminus \{e_y\}$, i.e., select from C_y but not e_y nor edges that are also in C_x.

There is a final twist to this choice, which makes the coupling non-Markovian, i.e., it does not verify the conditions in Equations (1a) and (1b). We can choose $o_y = e_y$ more often than would otherwise be permissible by keeping track of how e_y was determined. If e_y is obtained deterministically, for example by the initial selection of x and y, then this is not possible. In general e_y might be determined by the changes in X_t, in which case we want to take advantage of the underlying randomness. Therefore, we keep track of the random processes that occur. The exact information we store is a set of edges $U_y \subseteq C_e \setminus \{e_x\}$, such that $e_y \in U_y$, and moreover this set contains the edges that are equally likely to be e_y. This information can be used to set $o_y = e_y$ when $s_y \in U_y$, however after such an action the information on U_y must be purged.

To illustrate the possible cases we use Figures 5–15, where the edges drawn with double lines represent a generic path that may contain several edges, or none at all. The precise cases are as follows:

1. If the chain X_t loops ($x' = x$), because $i_x \in A_x$, then Y_t also loops and therefore $y' = y$. The set U_y does not change, i.e., set $U_{y'} = U_y$.
2. If $i_x = e_y$ and $o_x = e_x$ then set $i_y = e_x$ and $o_y = e_y$. In this case the chains do not coalesce; they swap states because $x' = y$ and $y' = x$ (see Figure 5). Set $U_{y'} = C_e \setminus \{e_{x'}\}$.

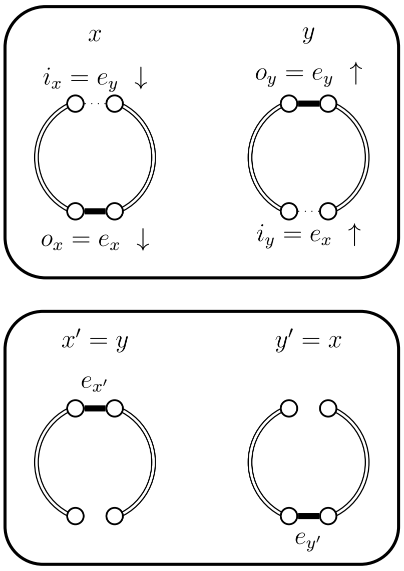

Figure 5. Case 2.

3. If $i_x = e_y$ and $o_x \neq e_x$ then set $i_y = e_x$ and $o_y = o_x$. In this case the chains coalesce, i.e., $x' = y'$ (see Figure 6). When the chains coalesce, the edges $e_{x'}$ and $e_{y'}$ no longer exist and the set $U_{y'}$ is no longer relevant.

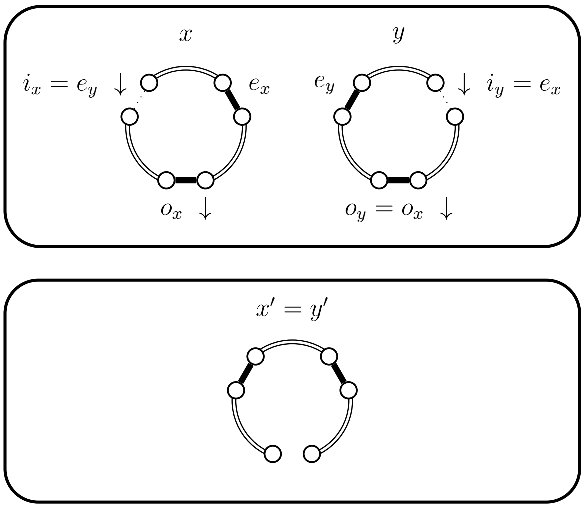

Figure 6. Case 3.

4. If $i_x \neq e_y$ set $i_y = i_x$. We now have three sub-cases, which are further sub-divided. These cases depend on whether $|C_x| = |C_y|$, $|C_x| < |C_y|$ or $|C_x| > |C_y|$. We start with $|C_x| = |C_y|$ which is simpler and establishes the basic situations. When $|C_x| < |C_y|$ or $|C_x| > |C_y|$ we use some Bernoulli random variables to balance out probabilities and whenever possible reduce to the cases considered for $|C_x| = |C_y|$. When this is not possible we present the corresponding new situation.

   (a) If $|C_x| = |C_y|$ we have the following situations:

      (i) If $o_x = e_x$ then set $o_y = e_y$. In this case the chains coalesce (see Figure 7).
      (ii) If $o_x \in I \setminus \{i_x\}$ then set $o_y = o_x$. In this case the chains do not coalesce, in fact the exclusive edges remain unchanged, i.e., $e_{x'} = e_x$ and $e_{y'} = e_y$ (see Figures 8 and 9). When $o_x \notin C_e$ the set $C_{e'}$ remains equal to C_e and likewise $U_{y'}$ remains equal to U_y (see Figure 8). Otherwise, when $o_x \in C_e$, the set $C_{e'}$ is different from C_e, and we assign $U_{y'} = U_y \cap C_{e'}$ (see Figure 9).
      (iii) If $o_x \in E_x \setminus \{e_x\}$ then select s_y uniformly from $E_y \setminus \{e_y\}$. If $s_y \in U_y$ then set $o_y = e_y$ (see Figure 10). In this case set $U_{y'} = E_x \setminus \{e_x\}$. The alternative, when $s_y \notin U_y$, is considered in the next case (4.*a*.iv).
      (iv) If $o_x \in E_x \setminus \{e_x\}$ and $s_y \notin U_y$, then set $o_y = s_y$. This case is shown in Figure 11. In this case the distance of the coupled states increases, i.e., $d(x', y') = 2$. Therefore we include a new state z' in between x' and y' and define $e_{z'}$ to be the edge in $A_{z'} \setminus A_{x'}$; $e_{y'}$ the edge in $A_{y'} \setminus A_{z'}$; and $e_{x'}$ the edge in $A_{x'} \setminus A_{z'}$. The set $U_{z'}$ should contain the edges that provide alternatives to $e_{z'}$. In this case set $U_{z'} = E_x \setminus \{e_x\}$ and $U_{y'} = (U_y \cap E_y) \setminus \{o_y\}$.

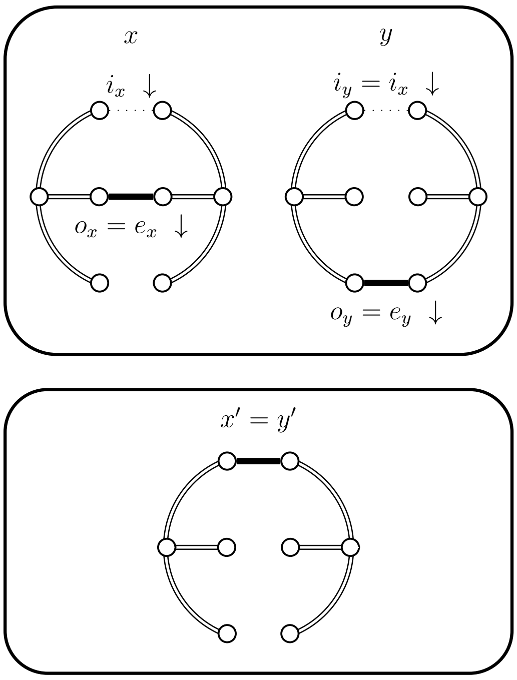

Figure 7. Case 4.*a*.i, case 4.*b*.i, and case 4.*c*.i.

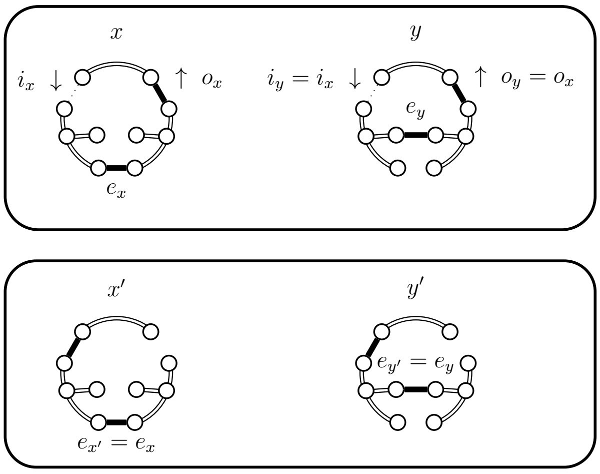

Figure 8. Case 4.*a*.ii, case 4.*b*.ii, and case 4.*c*.ii, when $o_x \notin C_e$.

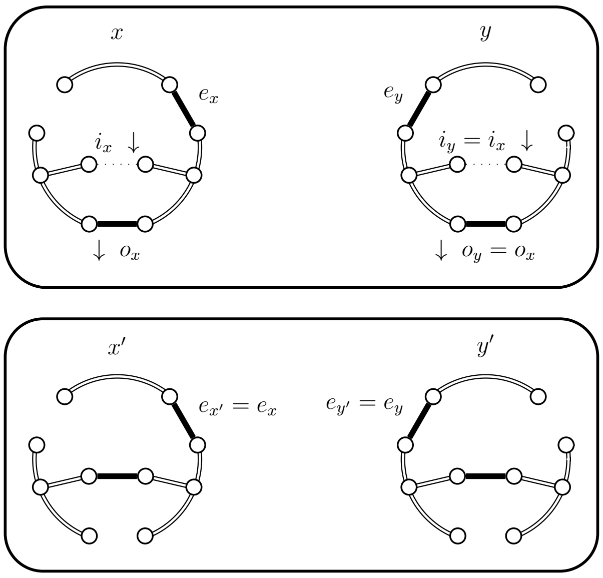

Figure 9. Case 4.*a*.ii, case 4.*b*.ii, and case 4.*c*.ii, when $o_x \in C_e$.

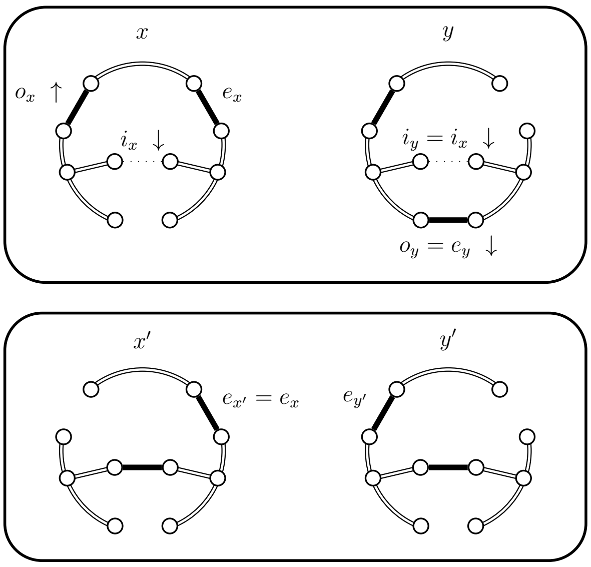

Figure 10. Case 4.*a*.iii, case 4.*b*.iv, and case 4.*c*.iv, when B' is true.

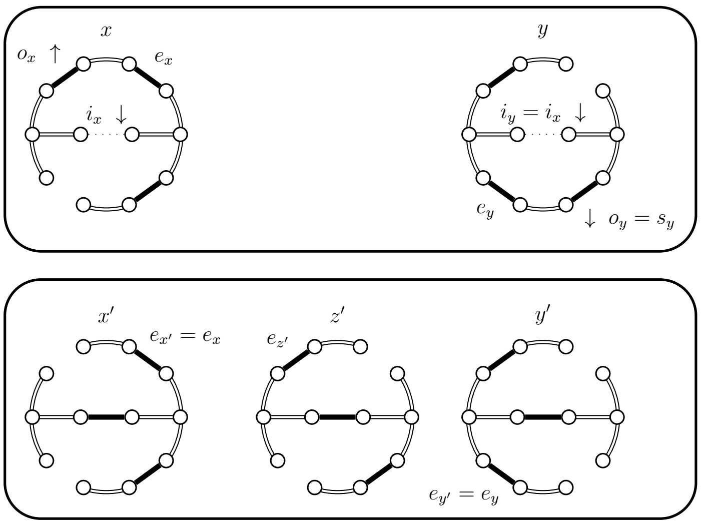

Figure 11. Case 4.*a*.iv, case 4.*b*.iv, and case 4.*c*.iv.

(b) If $|C_x| < |C_y|$ then X_t will choose $o_x \in I$ with a higher probability then what Y_t should. Therefore, we use a Bernoulli random variable B with a success probability p defined as follows:

$$p = \frac{C_x - 1}{C_y - 1}$$

In Lemma 2 we prove that p properly balances the necessary probabilities. For now note that when $|C_x| = |C_y|$ the expression for p yields $p = 1$. This is coherent with the following cases, because when B yields `true` we use the choices defined for $|C_x| = |C_y|$. The following situations are possible:

(i) If $o_x = e_x$ then we reduce to the case 4.*a*.i, both when B yields `true` or when B fails and $s_y \in U_y$. Set $o_y = e_y$ (see Figure 7). The new case occurs when B fails and $s_y \notin U_y$, in this situation set $o_y = s_y$ and $U_{y'} = (U_y \cap C_y) \setminus \{o_y\}$ (see Figure 12).

(ii) If $o_x \in I \setminus \{i_x\}$ then we reduce to the case 4.*a*.ii when B yields `true`. Set $o_y = o_x$ (see Figures 8 and 9). When B fails and $s_y \in U_y$ we have a new situation. Set $o_y = e_y$ and $U_{y'} = I \setminus \{i_x\}$. The chains preserve their distance, i.e., $d(x', y') = 1$ (see Figure 13). The alternative, when $s_y \notin U_y$, is considered in the next case (4.*b*.iii).

(iii) If $o_x \in I \setminus \{i_x\}$ and B fails and $s_y \notin U_y$. We have a new situation,;set $o_y = s_y$. The distance increases, $d(x', y') = 2$ (see Figure 14). Set $U_{z'} = I \setminus \{i_x\}$ and $U_{y'} = (U_y \cap E_y) \setminus \{o_y\}$.

(iv) If $o_x \in E_x \setminus \{e_x\}$ then if $s_y \in U_y$ use case 4.*a*.iii (Figure 10), otherwise, when $s_y \notin U_y$, use case 4.*a*.iv (Figure 11).

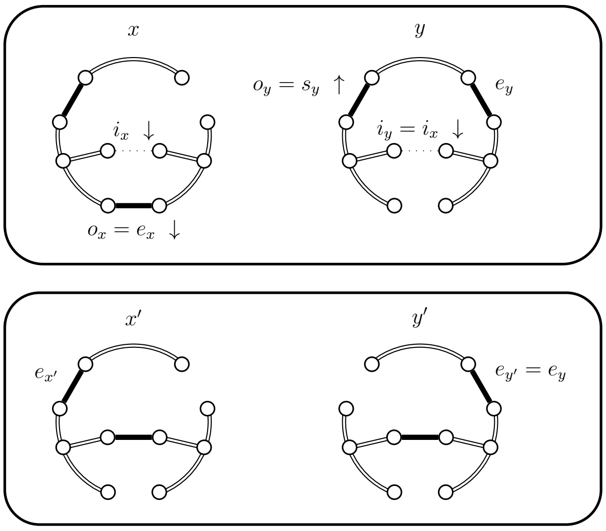

Figure 12. Case 4.*b*.i when B fails and $s_y \notin U_y$.

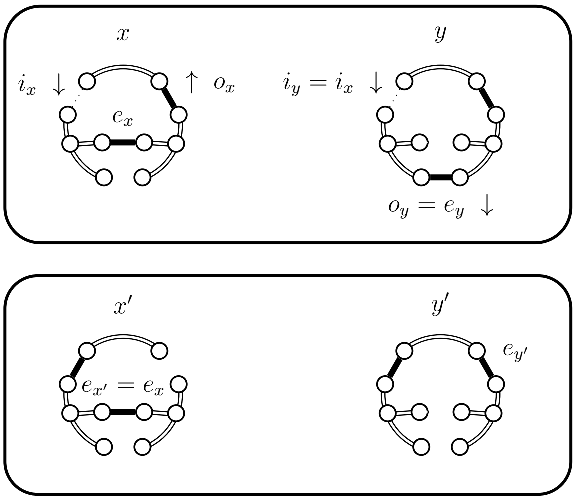

Figure 13. Case 4.*b*.ii.

(c) If $|C_x| > |C_y|$ we have the following situations:

(i) If $o_x = e_x$ then use case 4.*a*.i and set $o_y = e_y$ (see Figure 7). The chains coalesce.

(ii) If $o_x \in I \setminus \{i_x\}$ then use case 4.*a*.ii and set $o_y = o_x$ (see Figures 8 and 9).

(iii) If $o_x \in E_x \setminus \{e_x\}$ then we use a new Bernoulli random variable B^* with a success probability p^* defined as follows:

$$p^* = \left(\frac{1}{C_y - 1} - \frac{1}{C_x - 1}\right) \times \frac{(C_x - 1)(I - 1)}{E_x - 1}$$

In Lemma 2 we prove that B^* properly balances the necessary probabilities. For now, note that when $|C_x| = |C_y|$ the expression for p^* yields $p^* = 0$, because $1/(C_y - 1) - 1/(C_x - 1)$ becomes 0. This is coherent because when B^* returns `false` we will use the choices defined for $|C_x| = |C_y|$. The case when B^* fails is considered in the next case (4.*c*.iv).

If B^* is successful we have a new situation. Set $o_y = s_i$, where s_i is chosen uniformly from $I \setminus \{i_y\}$ (see Figure 15). We have $e_{y'} = e_y$, $U_{y'} = (U_y \cap E_y) \setminus \{o_y\}$, $e_{z'} = o_x$ and $U_{z'} = E_x \setminus \{e_x\}$.

(iv) If $o_x \in E_x \setminus \{e_x\}$ and B^* fails we use another Bernoulli random variable B' with a success probability p' defined as follows:

$$p' = 1 - \frac{(C_x - 1)(E_y - 1)}{(C_y - 1)(E_x - 1)(1 - p^*)}$$

In Lemma 2 we prove that B' properly balances the necessary probabilities. In case B' yields `true`, use case 4.*a*.iii and set $o_y = e_y$ (see Figure 10). Otherwise, if $s_y \in U_y$, use case 4.*a*.iii (Figure 10) or, if $s_y \notin U_y$, use case 4.*a*.iv (Figure 11).

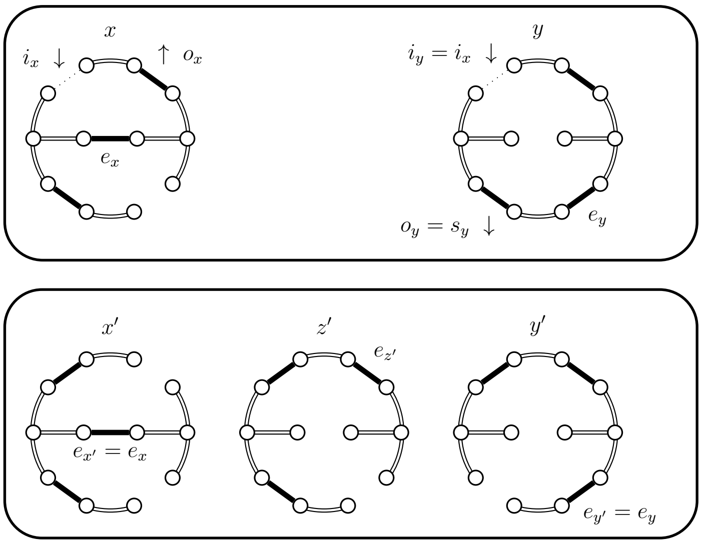

Figure 14. Case 4.*b*.iii.

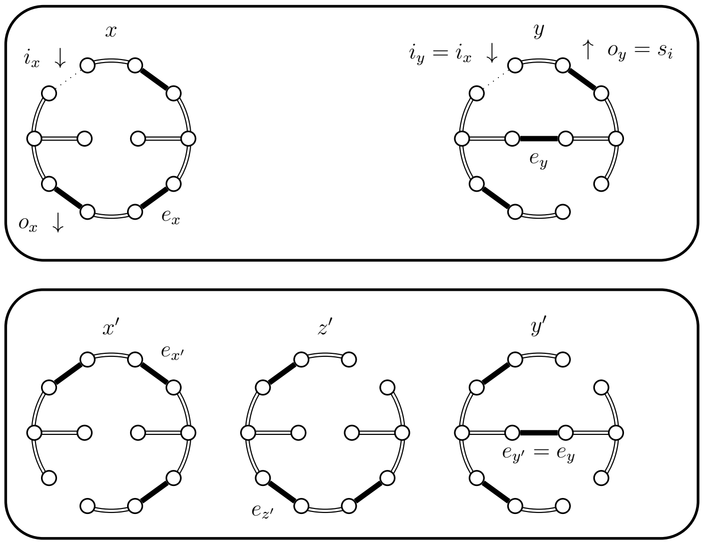

Figure 15. Case 4.*c*.iii, when B^* is true.

Notice that the case 4.*a*.ii applies when $C_x = C_y$, thus solving this situation as a particular case. This case is shown in Figure 8. It may even be the case that $C_x = C_y$ and C_x and C_e are disjointed, i.e., $C_x \cap C_e = \emptyset$. This case is not drawn.

Formally, a coupling is Markovian when Equations (1a) and (1b) hold, where Z_t is the coupling, which is defined as a pair of chains (X_t, Y_t). The chain M_t represents the original chain.

$$\Pr(X_{t+1} = x' \mid Z_t = (x, y)) = \Pr(M_{t+1} = x' \mid M_t = x) \tag{1a}$$

$$\Pr(Y_{t+1} = y' \mid Z_t = (x, y)) = \Pr(M_{t+1} = y' \mid M_t = y) \tag{1b}$$

To establish vital insight into the coupling structure we will start by studying it when it is Markovian.

Lemma 2. *When $U_y = \{e_y\}$, the process we described is a Markovian coupling.*

Proof. The coupling verifies Equation (1a), because we do not alter the behavior of the chain X_t. Hence the main part of the proof focuses on Equation (1b).

First, let us prove that for any edge $i \in E$ the probability that $i_y = i$ is $1/E$, i.e., $\Pr(i_y = i) = 1/E$. The possibilities for i_y are the following:

- $i \in A_y$: this occurs only in case 1, when $i_x \in A_x$. It may be that $i = e_y$; this occurs when $i_x = e_x$, in which case $i_y = e_y = i$ and this is the only case where $i_y = e_y$. In this case $\Pr(i_y = i) = \Pr(i_x = e_x) = 1/E$. Otherwise, $i \in A_y \cap A_x$, in these cases $i_y = i_x$, and therefore $\Pr(i_y = i) = \Pr(i_x = i) = 1/E$.
- $i = e_x$: this occurs in cases 2 and 3, i.e., when $i_x = e_y$, which is the decisive condition for this choice. Therefore $\Pr(i_y = i) = \Pr(i_x = e_y) = 1/E$.
- $i \in E \setminus A_y$: this occurs in case 4. In this case $i_y = i_x$ so again we have that $\Pr(i_y = i) = \Pr(i_x = i) = 1/E$.

Before focusing on o_x we will prove that the Bernoulli random variables are well defined, i.e., that the expressions on the denominators are not 0 and that the values of p, p^*, p' are between 0 and 1.

- Analysis of B. We need to have $C_y - 1 \neq 0$ for p to be well defined. Any cycle must contain at least three edges; therefore $3 \leq C_y$ and hence $0 < 2 \leq C_y - 1$. This guarantees that the denominator is not 0. The same argument proves that $0 < C_x - 1$, thus implying that $0 < p$, as both expressions are positive. We also establish that $p < 1$ because of the hypothesis of case 4.*b* which guarantees $C_x < C_y$ and therefore $C_x - 1 < C_y - 1$.
- Analysis of B^*. As in seen the analysis of B we have that $0 < C_y - 1$ and $0 < C_x - 1$, therefore those denominators are not 0. Moreover we also need to prove that $E_x - 1 \neq 0$. In general we have that $1 \leq E_y$, because $e_y \in E_y$. Moreover, the hypothesis of case 4.*c*.iii is that $C_y < C_x$ and therefore $E_y < E_x$, which is obtained by removing I from the both sides. This implies that $1 < E_x$ and therefore $0 < E_x - 1$, thus establishing that the last denominator is also not 0.
  Let us now establish that $0 \leq p^*$ and $p^* < 1$. Note that p^* can be simplified to the expression $(C_x - C_y)(I - 1)/((C_y - 1)(E_x - 1))$, where all the expressions in parenthesis are non-negative, so $0 \leq p^*$. For the second property we use the new expression for p^* and simplify $p^* < 1$ to $(E_x - E_y)(I - 1) < (E_x - 1)(C_y - 1)$. The deduction is straightforward using the equality $C_x - C_y = E_x - E_y$ that is obtained by removing I from the left side. The properties $E_x - E_y \leq E_x - 1$ and $I - 1 < C_y - 1$ establish the desired result.
- Analysis of B'. We established, in the analysis of B, that $C_y - 1$ is non-zero. In the analysis of B^* we also established that $E_x - 1$ is non-zero. Note that case 4.*c*.iv also assumes the hypothesis that $C_y < C_x$. Moreover, in the analysis of B^* we also established that $p^* < 1$, which implies that $0 < 1 - p^*$ and therefore the last denominator is also non-zero.
  Let us also establish that $0 \leq p'$ and $p' \leq 1$. For the second property we instead prove that $0 \leq 1 - p'$, where $1 - p' = (C_x - 1)(E_y - 1)/((C_y - 1)(E_x - 1)(1 - p^*))$ and all of the expressions

in parenthesis are non-negative. We use the following deduction of equivalent inequalities to establish that $0 \leq p'$:

$$\begin{aligned}
0 &\leq p' \\
-p' &\leq 0 \\
1 - p' &\leq 1 \\
(C_x - 1)(E_y - 1) &\leq (C_y - 1)(E_x - 1)(1 - p^*) \\
(C_x - 1)(E_y - 1) &\leq (C_y - 1)(E_x - 1)\left(1 - \frac{(C_x - C_y)(I - 1)}{(C_y - 1)(E_x - 1)}\right) \\
(C_x - 1)(E_y - 1) &\leq (C_y - 1)(E_x - 1) - (C_x - C_y)(I - 1) \\
(E_x - 1)(E_y - 1) + I(E_y - 1) &\leq (E_y - 1)(E_x - 1) + I(E_x - 1) - (E_x - E_y)(I - 1) \\
I(E_y - 1) &\leq I(E_x - 1) - (E_x - E_y)(I - 1) \\
I((E_y - 1) + E_x - E_y) &\leq I(E_x - 1) + E_x - E_y \\
I(E_x - 1) + E_y &\leq I(E_x - 1) + E_x \\
E_y &\leq E_x \\
C_y &\leq C_x
\end{aligned}$$

This last inequality is part of the hypothesis of case 4.*c*.iv.

Now, let us focus on the edge o_x. We wish to establish that for any $o \in C_y \setminus \{i_y\}$ we have that $\Pr(o_y = o) = 1/(C_y - 1)$. We analyze this edge according to the following cases:

1. When the cycles are equal $C_x = C_y$. This involves cases 2 and 3.
   - $o = e_y$: this occurs only in case 2 and it is determined by the fact that $o_x = e_x$. Therefore, $\Pr(o_y = o) = \Pr(o_x = e_x) = 1/(C_x - 1) = 1/(C_y - 1)$.
   - $o \neq e_y$: this occurs only in case 3 and it is determined by the fact that $o_x \neq e_x$, in this case $o_y = o_x$. Therefore $\Pr(o_y = o) = \Pr(o_x = o) = 1/(C_x - 1) = 1/(C_y - 1)$.
2. When the cycles have the same size $|C_x| = |C_y|$ (case 4.*a*). The possibilities for o are as follows:
   - $o = e_y$: this occurs only in the case 4.*a*.i. This case is determined by the fact that $o_x = e_x$. Therefore, $\Pr(o_y = o) = \Pr(o_x = e_x) = 1/(C_x - 1) = 1/(C_y - 1)$. Note that according to the Lemma's hypothesis, case 4.*a*.iii never occurs.
   - $o \in I \setminus \{i_y\}$: this occurs only in case 4.*a*.ii. This case is determined by the fact that $o_x \in I \setminus \{i_x\}$ and sets $o_y = o_x = o$. Therefore, $\Pr(o_y = o) = \Pr(o_x = o) = 1/(C_x - 1) = 1/(C_y - 1)$.
   - $o \in E_y \setminus \{e_y\}$: this occurs only in case 4.*a*.iv. This case is determined by the fact that $o_x \in X \setminus \{e_x\}$ and moreover sets $o_y = s_y$, which was uniformly selected from $E_y \setminus \{e_y\}$. We have the following deduction where we use the fact that the events are independent and that $|C_x| = |C_y|$ implies $|E_x| = |E_y|$:

$$\begin{aligned}
\Pr(o_y = o) &= \Pr(o_x \in E_x \setminus \{e_x\} \text{ and } s_y = o) \\
&= \Pr(o_x \in E_x \setminus \{e_x\}) \Pr(s_y = o) \\
&= \frac{E_x - 1}{C_x - 1} \times \frac{1}{E_y - 1} \\
&= 1/(C_x - 1) \\
&= 1/(C_y - 1)
\end{aligned}$$

3. When $C_x < C_y$ this involves case 4.*b*. The cases for o are as follows:

- $o = e_y$;: this occurs only in the case 4.*b*.i and when B is `true`. This case occurs when $o_x = e_x$. We make the following deduction, that uses the fact that the events are independent and the success probability of B:

$$\begin{aligned}
\Pr(o_y = o) &= \Pr(o_x = e_x \text{ and } B = \texttt{true}) \\
&= \Pr(o_x = e_x)\Pr(B = \texttt{true}) \\
&= \frac{1}{C_x - 1} \times \frac{C_x - 1}{C_y - 1} \\
&= 1/(C_y - 1)
\end{aligned}$$

- $o \in I \setminus \{i_y\}$: this occurs only in case 4.*b*.ii and when B is `true`. This case is determined by the fact that $o_x \in I \setminus \{i_x\}$ and sets $o_y = o_x = o$. We make the following deduction, that uses the fact that the events are independent and the success probability of B:

$$\begin{aligned}
\Pr(o_y = o) &= \Pr(o_x = o \text{ and } B = \texttt{true}) \\
&= \Pr(o_x = o)\Pr(B = \texttt{true}) \\
&= \frac{1}{C_x - 1} \times \frac{C_x - 1}{C_y - 1} \\
&= 1/(C_y - 1)
\end{aligned}$$

- $o \in E_y \setminus \{e_y\}$: this occurs in case 4.*b*.iv, but also in cases 4.*b*.iii and 4.*b*.i when B is `false`. We have the following deduction, that uses event independence, the fact that the cases are disjoint events, and the success probability of B:

$$\begin{aligned}
&\Pr(o_y = o) \\
&\quad = \Pr(4.b.\text{iv or } (4.b.\text{iii and } B = \texttt{false}) \text{ or } (4.b.\text{i and } B = \texttt{false})) \\
&\quad = \Pr(4.b.\text{iv}) + \Pr(4.b.\text{iii and } B = \texttt{false}) + \Pr(4.b.\text{i and } B = \texttt{false}) \\
&\quad = \Pr(o_x \in E_x \setminus \{e_x\} \text{ and } s_y = o) + \Pr(o_x \in I \text{ and } B = \texttt{false} \text{ and } s_y = o) \\
&\quad\quad + \Pr(o_x = e_x \text{ and } B = \texttt{false} \text{ and } s_y = o) \\
&\quad = \Pr(o_x \in E_x \setminus \{e_x\})\Pr(s_y = o) + \Pr(o_x \in I)\Pr(B = \texttt{false})\Pr(s_y = o) \\
&\quad\quad + \Pr(o_x = e_x)\Pr(B = \texttt{false})\Pr(s_y = o) \\
&\quad = \Pr(o_x \in E_x \setminus \{e_x\})\Pr(s_y = o) + \Pr(o_x \in I \cup \{e_x\})\Pr(B = \texttt{false})\Pr(s_y = o) \\
&\quad = [\Pr(o_x \in E_x \setminus \{e_x\}) + \Pr(o_x \in I \cup \{e_x\})(1 - \Pr(B = \texttt{true}))]\Pr(s_y = o) \\
&\quad = [\Pr(o_x \in C_x) - \Pr(o_x \in I \cup \{e_x\})\Pr(B = \texttt{true})]\Pr(s_y = o) \\
&\quad = [1 - \Pr(o_x \in I \cup \{e_x\})\Pr(B = \texttt{true})]\Pr(s_y = o) \\
&\quad = \left[1 - \frac{I - 1 + 1}{C_x - 1} \times \frac{C_x - 1}{C_y - 1}\right]\Pr(s_y = o) \\
&\quad = \left[1 - \frac{I}{C_y - 1}\right]\Pr(s_y = o) \\
&\quad = \frac{C_y - 1 - I}{C_y - 1} \times \frac{1}{E_y - 1} \\
&\quad = \frac{1}{C_y - 1}
\end{aligned}$$

4. When $C_x > C_y$, this concerns case 4.*c*. The cases for o are the following:

- $o = e_y$: this occurs in the case 4.c.i and case 4.c.iv when B' is `true`. We use the following deduction:

$$\begin{aligned}
&\Pr(o_y = o)\\
&= \Pr(\text{4.c.i or (4.c.iv and } B' = \texttt{true}))\\
&= \Pr(\text{4.c.i}) + \Pr(\text{4.c.iv and } B' = \texttt{true})\\
&= \Pr(o_x = e_x) + \Pr(o_x \in E_x \setminus \{e_x\} \text{ and } B^* = \texttt{false} \text{ and } B' = \texttt{true})\\
&= \frac{1}{C_x - 1} + \frac{E_x - 1}{C_x - 1}(1 - p^*)\left(1 - \frac{(C_x - 1)(E_y - 1)}{(C_y - 1)(E_x - 1)(1 - p^*)}\right)\\
&= \frac{1}{C_x - 1} + \frac{E_x - 1}{C_x - 1}\left(1 - p^* - \frac{(C_x - 1)(E_y - 1)}{(C_y - 1)(E_x - 1)}\right)\\
&= \frac{1}{C_x - 1} + \frac{E_x - 1}{C_x - 1} - \frac{E_x - 1}{C_x - 1}p^* - \frac{E_y - 1}{C_y - 1}\\
&= \frac{1}{C_x - 1} + \frac{E_x - 1}{C_x - 1} - \left[\frac{1}{C_y - 1} - \frac{1}{C_x - 1}\right](I - 1) - \frac{E_y - 1}{C_y - 1}\\
&= \frac{1}{C_x - 1} + \frac{E_x - 1}{C_x - 1} - \frac{I - 1}{C_y - 1} + \frac{I - 1}{C_x - 1} - \frac{E_y - 1}{C_y - 1}\\
&= \frac{E_x + I - 1}{C_x - 1} - \frac{E_y + I - 1}{C_y - 1} + \frac{1}{C_y - 1}\\
&= \frac{C_x - 1}{C_x - 1} - \frac{C_y - 1}{C_y - 1} + \frac{1}{C_y - 1}\\
&= \frac{1}{C_y - 1}
\end{aligned}$$

- $o \in I \setminus \{i_y\}$: this occurs in case 4.c.ii and case 4.c.iii when B^* is `true`. We make the following deduction:

$$\begin{aligned}
&\Pr(o_y = o)\\
&= \Pr(\text{4.c.ii or 4.c.iii})\\
&= \Pr(\text{4.c.ii}) + \Pr(\text{4.c.iii})\\
&= \Pr(o_x = o) + \Pr(o_x \in E_x \setminus \{e_x\} \text{ and } B^* = \texttt{true} \text{ and } s_i = o)\\
&= \Pr(o_x = o) + \Pr(o_x \in E_x \setminus \{e_x\}) \Pr(B^* = \texttt{true}) \Pr(s_i = o)\\
&= \frac{1}{C_x - 1} + \frac{E_x - 1}{C_x - 1} \times \left(\frac{1}{C_y - 1} - \frac{1}{C_x - 1}\right) \times \frac{(C_x - 1)(I - 1)}{E_x - 1} \times \frac{1}{I - 1}\\
&= \frac{1}{C_x - 1} + \frac{1}{C_y - 1} - \frac{1}{C_x - 1}\\
&= \frac{1}{C_y - 1}
\end{aligned}$$

- $o \in E_y \setminus \{e_y\}$: this occurs in case 4.c.iv when B' is `false`. We have the following deduction:

$$\begin{aligned}
&\Pr(o_y = o)\\
&= \Pr(\text{4.c.iv and } B' = \texttt{false})\\
&= \Pr(o_x \in E_x \setminus \{e_x\} \text{ and } B^* = \texttt{false} \text{ and } B' = \texttt{false} \text{ and } s_y = o)\\
&= \Pr(o_x \in E_x \setminus \{e_x\}) \Pr(B^* = \texttt{false}) \Pr(B' = \texttt{false}) \Pr(s_y = o)\\
&= \frac{E_x - 1}{C_x - 1}(1 - p^*)\frac{(C_x - 1)(E_y - 1)}{(C_y - 1)(E_x - 1)(1 - p^*)} \times \frac{1}{E_y - 1}\\
&= \frac{1}{C_y - 1}
\end{aligned}$$

□

Lemma 3. *The process we described is a non-Markovian coupling.*

Proof. In the context of a Markovian coupling we analyze the transition from y to y' given the information about x. In the non-Markovian case we will use less information about x. We assume that e_y is a random variable and that x provides only e_x and U_y; we know only that $e_y \in U_y$ and moreover that $\Pr(e_y = e) = 1/U_y$ for any $e \in U_y$ and $\Pr(e_y = e) = 0$ otherwise. Then the chain X_t makes its move and provides information about i_x and o_x. Let us consider only the cases when Y_t and choose $i_y = i_x$, because nothing changes in the cases where this does not happen. Now, i_x can be used to define C_x and C_y and, therefore, E_x and E_y. We focus our attention on $E_y \cap U_y$ because, except for the trivial cases, we must have $e_y \in (E_y \cap U_y)$. Hence, we instead alter our condition to $\Pr(e_y = e) = 1/|E_y \cap U_y|$, for any $e \in (E_y \cap U_y)$ and 0 otherwise.

Note that this is a reasonable process because we established in Lemma 1 that, in the non-trivial case, E_x and E_y partition C_e and, therefore, because $U_y \subseteq C_e \setminus \{e_x\}$, we have that E_x and E_y also partition U_y. This means that $U_y \cap I = \emptyset$ and so we are not losing any part of U_y in this process; we are only dividing it into cases. This process is also the reason why, even when $s_y \notin U_y$, we define $U_{y'} = (U_y \cap C_y) \setminus \{o_y\} = (U_y \cap E_y) \setminus \{o_y\}$.

Now the cases considered in Lemma 2 must be changed. Substitute the original cases of $o = e_y$ for $o \in U_y \cap E_y$. Also, substitute the case $o \in E_y \setminus \{e_y\}$ for $o \in E_y \setminus U_y$. The other cases remain unaltered. Except for the first case, the previous deductions still apply. We will exemplify how the deduction changes for $o \in U_y \cap E_y$. We consider only the situation when $|C_x| = |C_y|$. For the remaining situations, $C_x < C_y$ and $C_x > C_y$, we use a general argument. Hence our precise assumptions are: $o \in U_y \cap E_y$ and $|C_x| = |C_y|$.

$$\begin{aligned}
\frac{\Pr(o_y = o)}{\Pr(e_y = o)} &= \Pr(o_x \in E_x \setminus \{e_x\} \text{ and } s_y \in U_y) + \Pr(o_x = e_x) \\
&= \Pr(o_x \in E_x \setminus \{e_x\}) \Pr(s_y \in U_y) + \Pr(o_x = e_x) \\
&= \frac{E_x - 1}{C_x - 1} \times \frac{|U_y \cap E_y| - 1}{E_y - 1} + \frac{1}{C_x - 1} \\
&= \frac{|U_y \cap E_y|}{C_x - 1} \\
&= \frac{|U_y \cap E_y|}{C_y - 1}
\end{aligned}$$

This is correct according to our assumption.

The general argument follows the above derivation. Whenever the Markovian coupling produces $o_y = e_y$, we obtain a $1/(C_y - 1)$ probability of. Moreover, for every edge produced by the Markovian coupling such that $o_y \in E_y \setminus \{e_y\}$, we obtain another $1/(C_y - 1)$ probability. This totals $|U_y \cap E_y|/(C_y - 1)$, as desired. This also occurs in the cases when $C_x < C_y$ and $C_x > C_y$, only the derivations become more cumbersome.

Finally will argue why $\Pr(e_y = e) = 1/U_y$ when $e \in U_y$. The set U_y is initialized to contain only the edge e_y, i.e., $U_y = \{e_y\}$. As the coupling proceeds, $U_{y'}$ is chosen to represent the edges, from which $e_{y'}$ was chosen, or is simply restricted if e_y does not change. More precisely, in case 4.*a*.iii we choose $e_{y'} = o_x$; in case 4.*b*.iv we choose $e_{z'} = o_x$; and in case 4.*b*.ii we choose $e_{y'} = o_x$. □

We obtained no general bounds on the coupling we presented; it may even be that such bounds are exponential even if the Markov chain has a polynomial mixing time. In fact, Kumar and Ramesh [10] proved that this is the case for Markovian couplings of the Jerrum–Sinclair chain [11]. Note that according to the classification of Kumar and Ramesh [10] the coupling we present is considered as time-variant Markovian. Hence their result applies to the type of coupling we are using, although we are considering different chains so it is not immediately seen that indeed there exist no polynomial Markovian couplings for the chain we presented. Cycle graphs are the only class of graphs for which we establish polynomial bounds (see Figure 16).

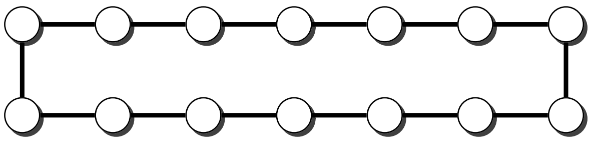

Figure 16. A cycle graph.

Theorem 2. *For any cycle graph G the mixing time τ of edge-swap chain is $O(V)$ for the normal version of the chain and $1/\log_4(V-1)$ for the fast version.*

Proof. For any two trees A_x and A_y we have a maximum distance of one edge, i.e., $d(A_x, A_y) \leq 1$. Hence our coupling applies directly.

For the fast version, case 1 does not occur, because i_x is chosen from $E \setminus \{A_x\}$. Hence, the only cases that might apply are cases 2 and 3. In first case, the chains preserve their distance and in the last case the distance is reduced to 0. Hence, $E[d(X_1, Y_1)] = 1/(V-1)$, which corresponds to the probability of case 2. Each step of the coupling is independent, which means we can use the previous result and Markov's inequality to obtain $\Pr(d(X_\tau, Y_\tau) \geq 1) \leq 1/(V-1)^\tau$. Then, we use this probability in the coupling Lemma 4 to obtain a variation distance of 1/4, by solving the following equation: $1/(V-1)^\tau = 1/4$.

For the slow version of the chain, case 1 applies most of the time, i.e., for $V-1$ out of V choices of i_x. It takes $V-1$ steps for the standard chain to behave as the fast chain and, therefore, the time should be $(V-1)/\log_4(V-1)$. □

This result is in stark contrast with the alternative methods, the random walk and Wilson's (see Section 5) algorithms, which require $O(V^2)$ time [7]. More recent algorithms are also at least $O(V^{4/3})$ for this case.

Moreover when a graph is a connected set of cycles linked by bridges or articulation points we can also establish a similar result. Figure 17 shows one such graph.

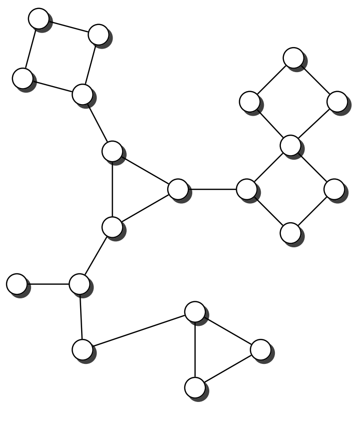

Figure 17. Cycles connected by bridges or articulation points.

Theorem 3. *For any graph G which consists of n simple cycles connected by bridges or articulation points, such that m is the size of the smallest cycle, then the mixing time τ of the fast edge swap chain is as follows:*

$$\tau = \frac{\log(4n)}{\log\left(\frac{n(m-1)}{n(m-1)-(m-2)}\right)}$$

The mixing time for the slow version is obtained by using $|E|$ instead of n in the previous expression.

Proof. To obtain this result we use a path-coupling argument. Then, for two chains at distance 1 we have $E[d(X_1, Y_1)] \leq \left(1 - \frac{m-2}{n(m-1)}\right)$.

We assume that the different edge occurs in the largest cycle. In general the edges inserted and deleted do not alter this situation, hence the term 1. However with probability $1/n$ the chain X_t inserts an edge that creates the cycle where the difference occurs. In that case with probability $(m-2)/(m-1)$ the chains coalesce. Hence applying path-coupling the mixing time must verify the following equation:

$$n\left(1 - \frac{m-2}{n(m-1)}\right)^{\tau} \leq \frac{1}{4}$$

For the slow version the chain choose the correct edge with probability $1/E$ instead of $1/n$. □

4.3. Experimental Results

4.3.1. Convergence Testing

Before looking at the performance of the algorithm we started by testing the convergence of the edge swap chain. We estimate the variation distance after a varying number of iterations. The results are shown in Figures 18–24. We now describe the structure of these figures. Consider for example Figure 20. The structure is as follows:

- The bottom left plot shows the graph properties, the number of vertices V in the x axis and the number edges E on the y axis. For the dense case graph 0 has 10 vertices and 45 edges. Moreover, graph 6 has 40 vertices and 780 edges. These graph indexes are used in the remaining plots.
- The top left plot shows the number of iterations t of the chain in the x axis and the estimated variation distance on the y axis, for all the different graphs.
- The top right plot is similar to the top left, but the x axis contains the number of iterations divided by $(V^{1.3} + E)$. Besides the data this plot also shows a plot of $\ln(1/\hat{\varepsilon})$ for reference.
- The bottom right plot is the same as the top right plot, using a logarithmic scale on the y axis.

To avoid the plots from becoming excessively dense, we do not plot points for all experimental values, instead we plot one point out of three. However, the lines pass through all experimental points, even those that are not explicit.

The different values at the end of **dmP**, namely in Figures 22–24, correspond to the choices of p.

The variation distance between two distributions D_1 and D_2 on a countable state space S is given by $\|D_1 - D_2\| = \sum_{x \in S} |D_1(x) - D_2(x)|/2$. This is the real value of ε. However, the size of S quickly becomes larger than we can compute. Instead, we compute a simpler variation distance $\|D_1 - D_2\|_d$, where S is reduced from the set of all spanning trees of G to the set of integers from 0 to $V-1$, which correspond to the edge distance, defined in Section 4.2, of the generated tree A to a fixed random spanning tree R. More precisely, we generate 20 random trees, using a random walk algorithm described in Section 5. For each of these trees, we compute $\|\pi - M_t\|_d$, i.e., the simpler distance between the stationary distribution π and the distribution M_t obtained by computing t steps of the edge-swapping chain. To obtain M_t, we start from a fixed initial tree A_0 and execute our chain t times. This process is repeated several times to obtain estimates for the corresponding probabilities. We keep two sets of estimates M_t and M'_t and stop when $\|M_t - M'_t\|_d < 0.05$. Moreover, we only estimate values where $\|\pi - M_t\|_d \geq 0.1$. We use the same criteria to estimate π, but in this case the trees are again generated by the random walk algorithm. The final value $\hat{\varepsilon}$ is obtained as the maximum value obtained for the 20 trees.

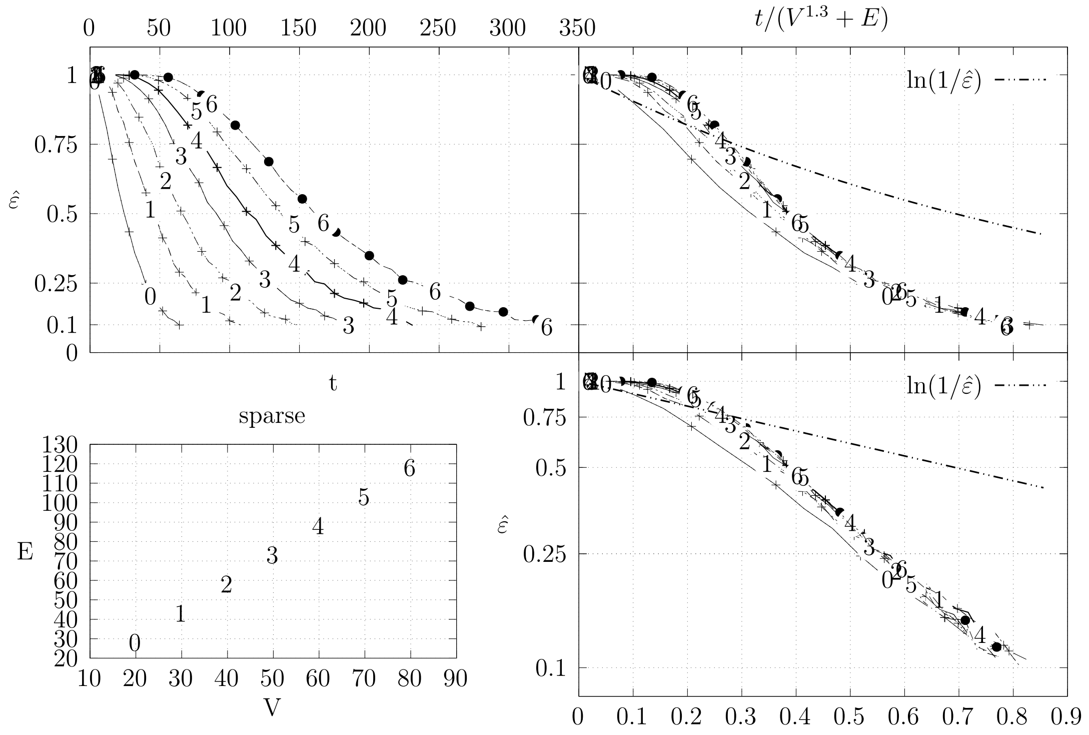

Figure 18. Estimation of variation distance as a function of the number of iterations for **sparse** graphs (see Section 4.3.1 for details).

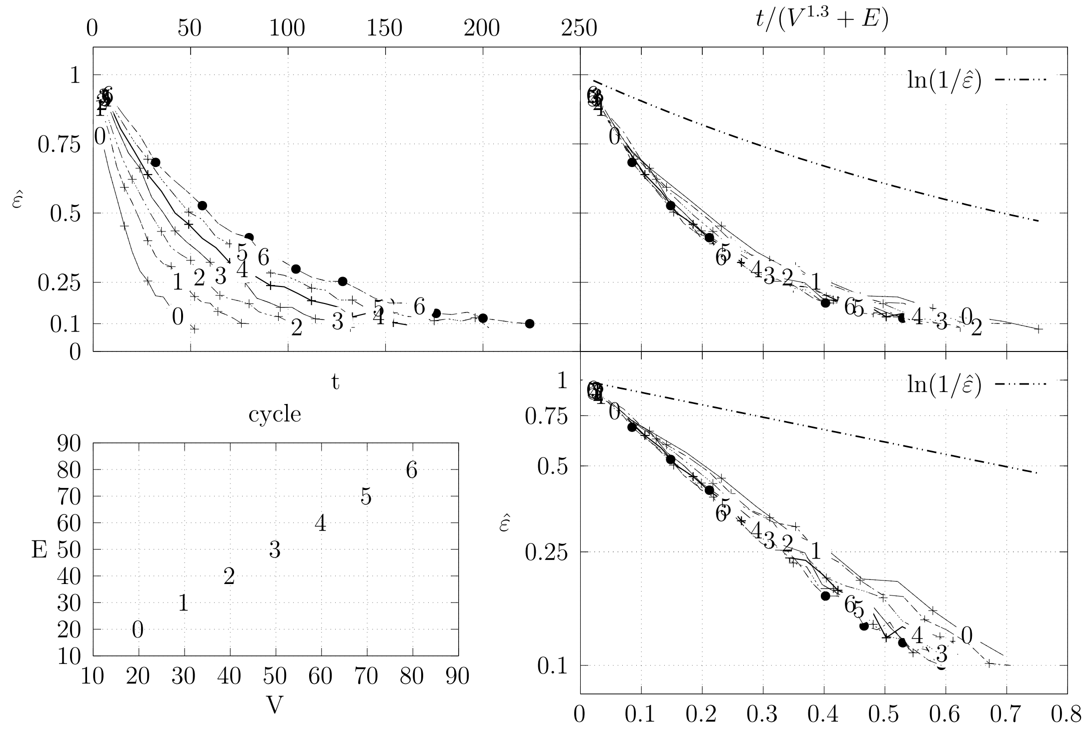

Figure 19. Estimation of variation distance as a function of the number of iterations for **cycle** graphs (see Section 4.3.1 for details).

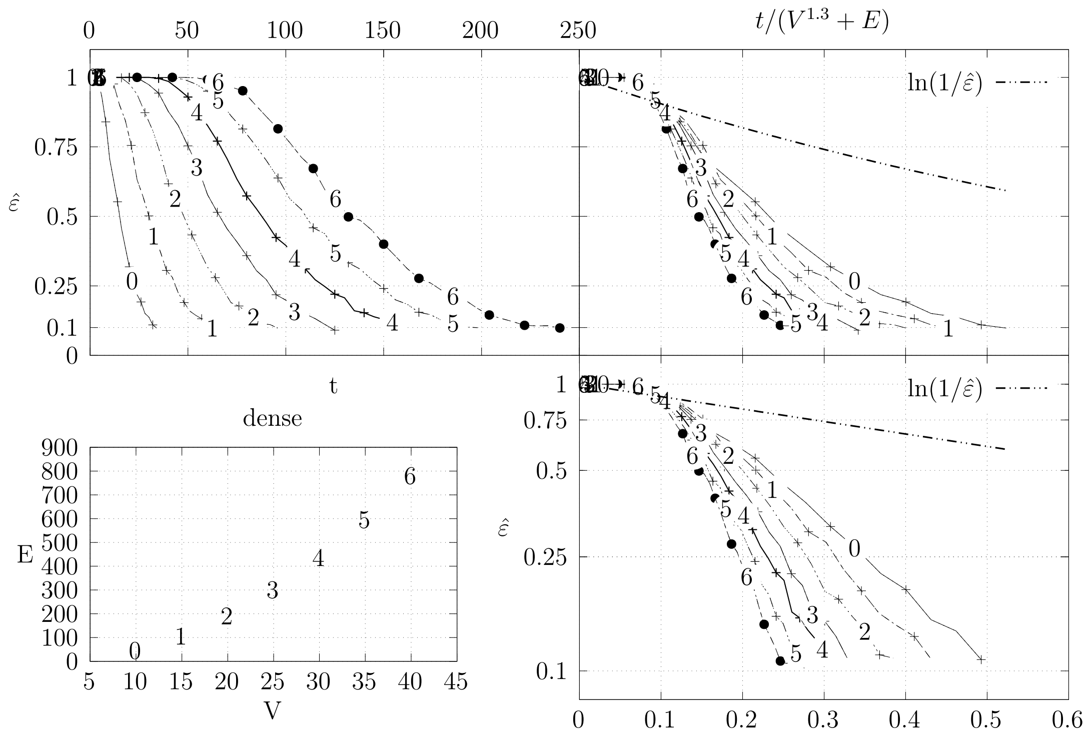

Figure 20. Estimation of variation distance as a function of the number of iterations for **dense** graphs (see Section 4.3.1 for details).

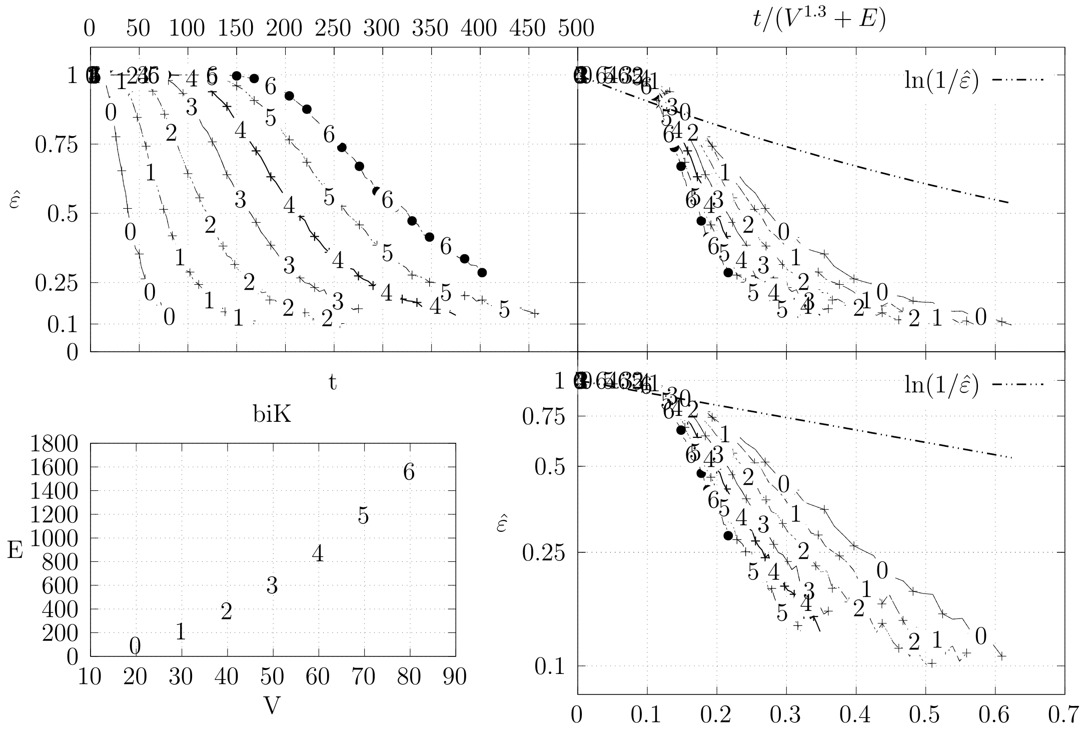

Figure 21. Estimation of variation distance as a function of the number of iterations for **biK** graphs (see Section 4.3.1 for details).

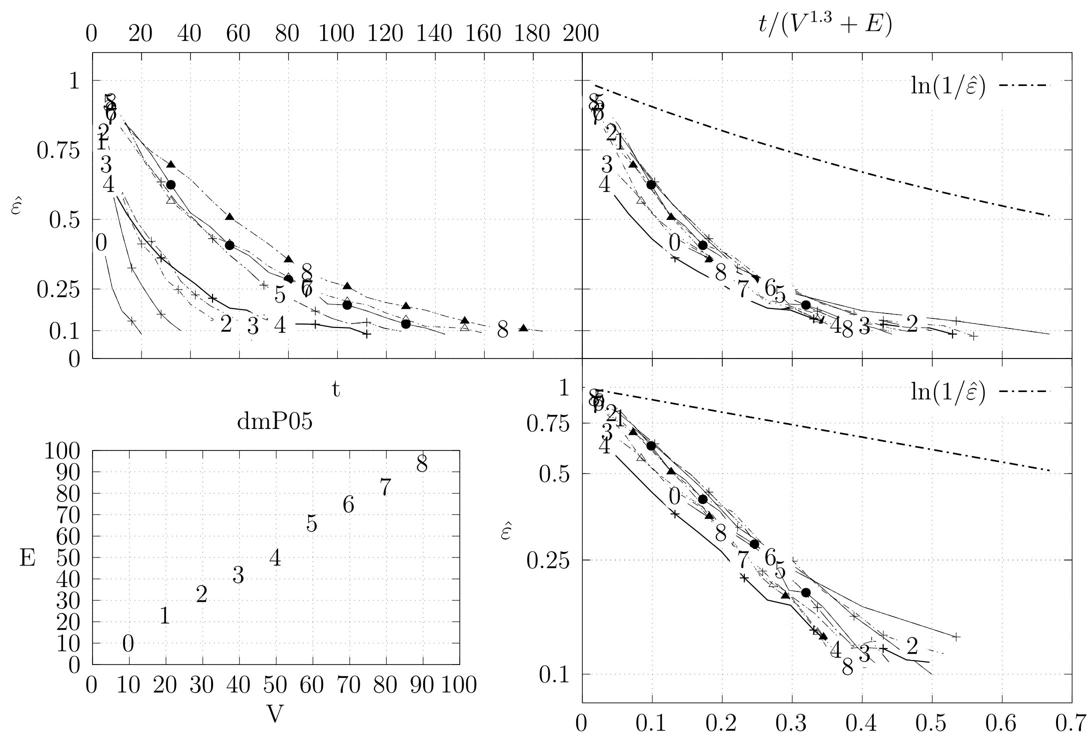

Figure 22. Estimation of variation distance as a function of the number of iterations for **dmP** graphs (see Section 4.3.1 for details).

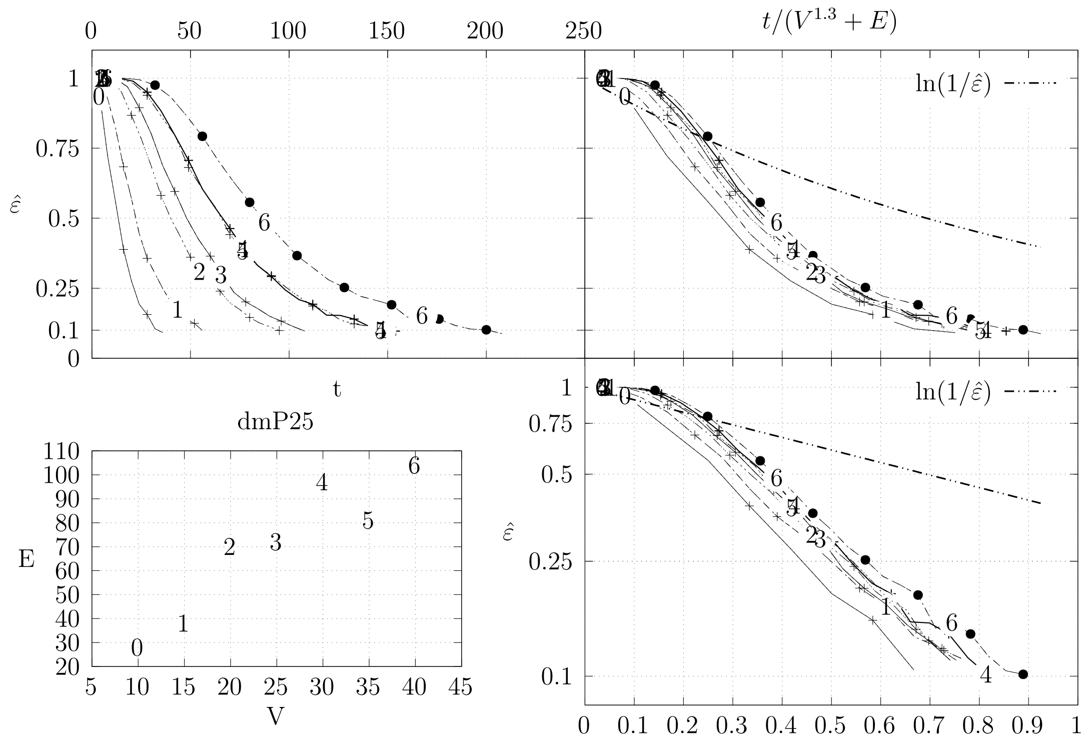

Figure 23. Estimation of variation distance as a function of the number of iterations for **dmP** graphs (see Section 4.3.1 for details).

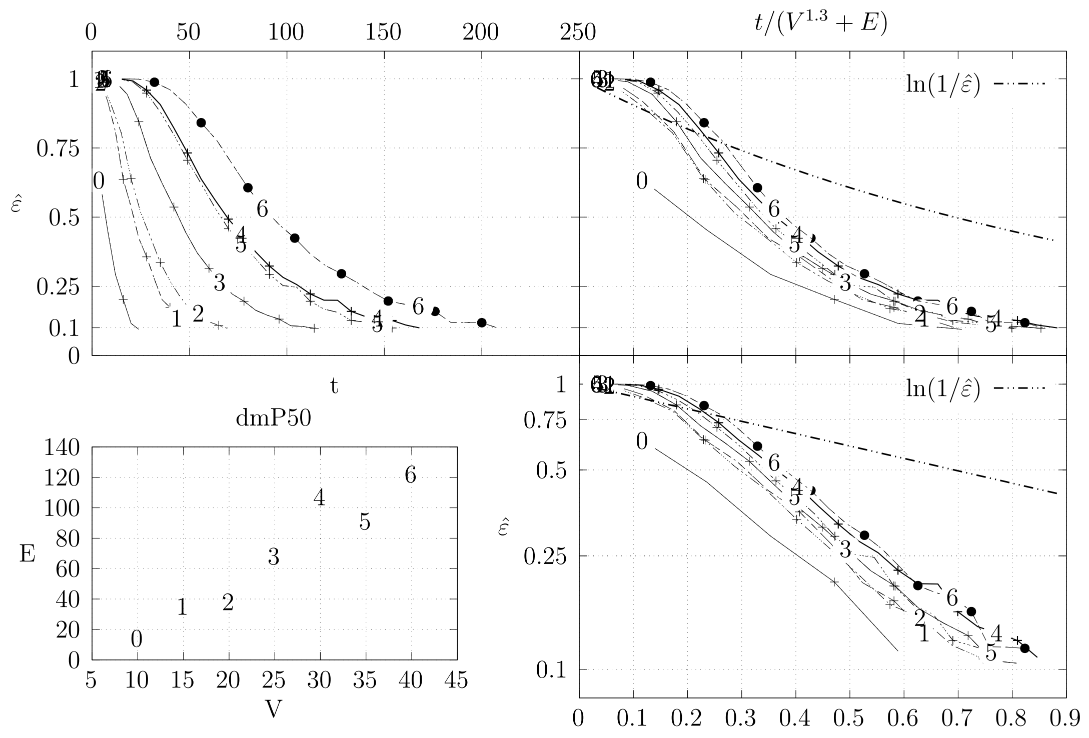

Figure 24. Estimation of variation distance as a function of the number of iterations for **dmP** graphs (see Section 4.3.1 for details).

We generated dense graphs, cycle graphs, sparse graphs, and some in-between graphs. The **cycle** graphs consist of a single cycle, as shown in Figure 16.The **sparse** graphs are ladder graphs; an illustration of these graphs is shown in Figure 25.

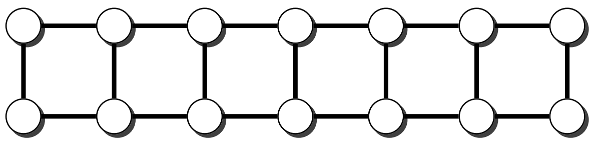

Figure 25. A ladder graph.

The **dense** graphs are actually the complete graphs K_V. We also generated other dense graphs labeled **biK** which consisted of two complete graphs connected by two edges. Graphs were also generated based on the duplication model **dmP**. Let $G_0 = (V_0, E_0)$ be an undirected and unweighted graph. Given $0 \leq p \leq 1$, the partial duplication model builds a graph $G = (V, E)$ by partial duplication as follows [12]: start with $G = G_0$ at time $t = 1$ and, at time $t > 1$, perform a duplication step:

1. Uniformly select a random vertex u of G.
2. Add a new vertex v and an edge (u, v).
3. For each neighbor w of u, add an edge (v, w) with probability p.

These graphs show the convergence of the Markov chain and moreover $V^{1.3} + E$ seems to be a reasonable bound for τ. Still, these results are not entirely binding. On the one hand the estimation of the variation distance groups several spanning trees into the same distance, which means that within a group the distribution might not be uniform, even if the global statistics are good. Hence, the actual variation distance may be larger and the convergence might be slower. On the other hand, we chose the exponent 1.3 experimentally by trying to force the data of the graphs to converge at the same point. The actual value may be smaller or larger.

4.3.2. Coupling Simulation

As mentioned before, we obtained no general bounds on the coupling we presented. In fact, experimental simulation for the coupling does not converge for all classes of graphs. We obtained experimental convergence for cycle graph, as expected from Theorem 2, and for ladder graphs. For the remaining graphs we used an optimistic version of the coupling which always assumes that $s_y \in U_y$ and that B^* fails. With these assumptions, all the cases which increase the distance between states are eliminated and the coupling always converges. Note that this approach does not yield sound coupling, but in practice we verified that this procedure obtained good experimental variation distance. Moreover, the variation distance estimation for these tests is not the simpler distance but the actual experimental variation distance, obtained by generating several experimental trees, such that in average each possible tree is obtained 100 times.

The simulation of the path coupling proceeds by generating a path with $e \ln V$ steps, essentially selecting two trees at distance $e \ln V$ from each other. This path is obtained by computing $e \ln V$ steps of the fast chain. Recall that our implementation and all simulations use the fast version of the chain. The simulation ends once this path contracts to size $\ln V$. Let t' be the number of steps in this process. Once this point is obtained, our estimate for mixing time is $\hat{\tau} = t' \ln V$. In general, we wish to obtain $\hat{\tau}$ such that the probability that the two general chains X_t and X_t coalesce is at least 75%. Hence, we repeat this process four times and choose the second largest value of $\hat{\tau}$ as our estimate.

Table 1 summarizes results for the experimental variation distance. The number of possible spanning trees for each graph was computed with Kirchhoff's theorem. Then, we generated 100 times the number of possible trees, and we computed the variation distance. As stated above, we obtained good results for the variation distance, getting a median well below 25% for all tested graph topologies.

Table 1. Variation distance (VD) for different graph topologies. Median and maximum VD computed over five runs for each network. Since **dmP** graphs are random, results for **dmP** were further computed over five different graphs for each size $|V|$.

Graph	$\|V\|$	Median VD	Max VD
dense	$\{5, 7\}$	0.060	0.194
biK	$\{8, 10\}$	0.065	0.190
cycle	$\{16, 20, 24\}$	0.001	0.004
sparse	$\{10, 14, 20\}$	0.053	0.110
torus	$\{9, 12\}$	0.094	0.383
dmP	$\{8, 10, 12\}$	0.069	0.270

We now present experimental results for larger graphs where we use the optimistic coupling. All experiments were conducted on a computer with an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40 GHz with 4 cores and 32 GB of RAM. We present running times for different graph topologies and sizes in Figures 26–32. Note beforehand that the coupling estimate needs only to be computed once for each graph. Once the estimate is known, we can generate as many spanning trees as we want. Although the edge-swapping method is not always the faster compared with the random walk and Wilson's algorithms, it is competitive in practice for **dmP** and **torus** graphs, and it is faster for **biK**, **cycle** and **sparse** (ladder) graphs. As expected, it is less competitive for **dense** graphs. Hence, experimental results seem to point out that the edge-swapping method is more competitive in practice for those instances that are harder for random walk-based methods, namely **biK** and **cycle** graphs. The results for **biK** and **dmP** are of particular interest as most real networks seem to include these kind of topologies, i.e., they include communities and they are scale-free [13].

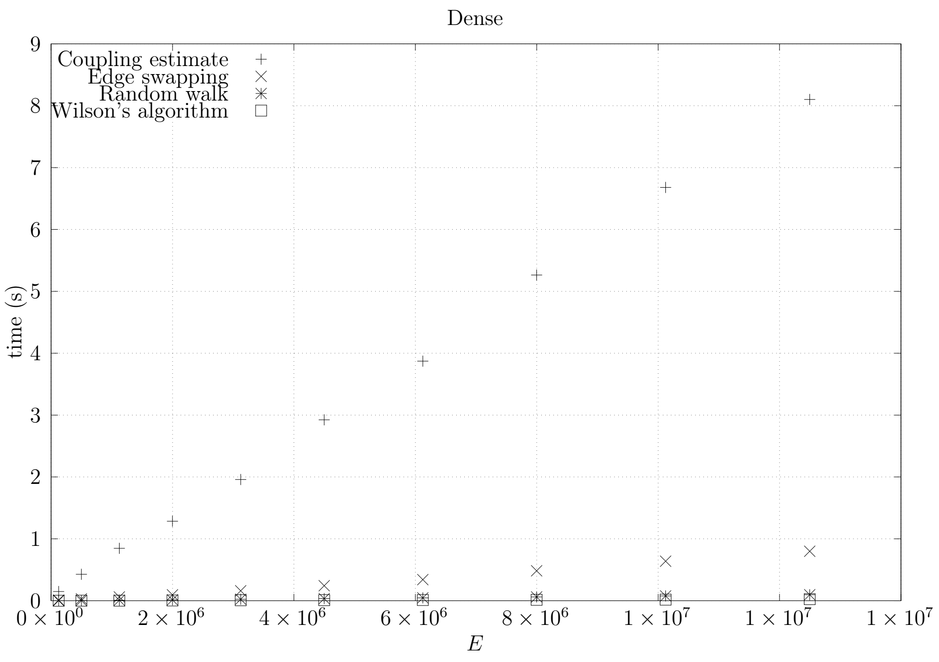

Figure 26. Running times for **dense** (fully connected) graphs averaged over five runs, including the running time for computing the optimistic coupling estimate and the running time for generating a spanning tree based on that estimate, as well as the edge-swapping algorithm, the running time for generating a spanning tree through a random walk, and the running time for Wilson's algorithm.

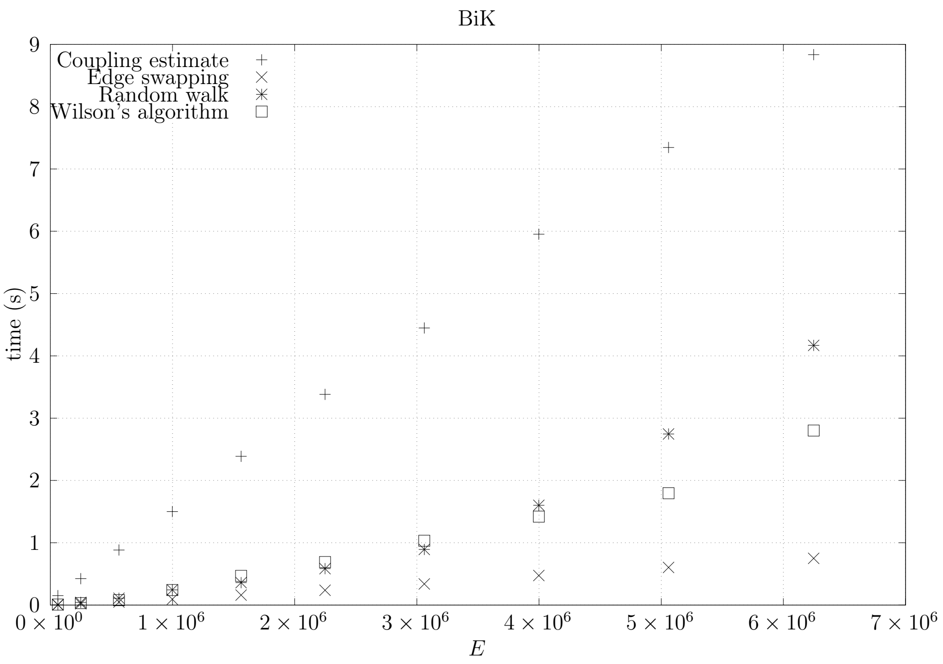

Figure 27. Running times for **biK** graphs averaged over five runs, including the running time for computing the optimistic coupling estimate, the running time for generating a spanning tree based on that estimate, the edge-swapping algorithm, the running time for generating a spanning tree through a random walk, and the running time for Wilson's algorithm.

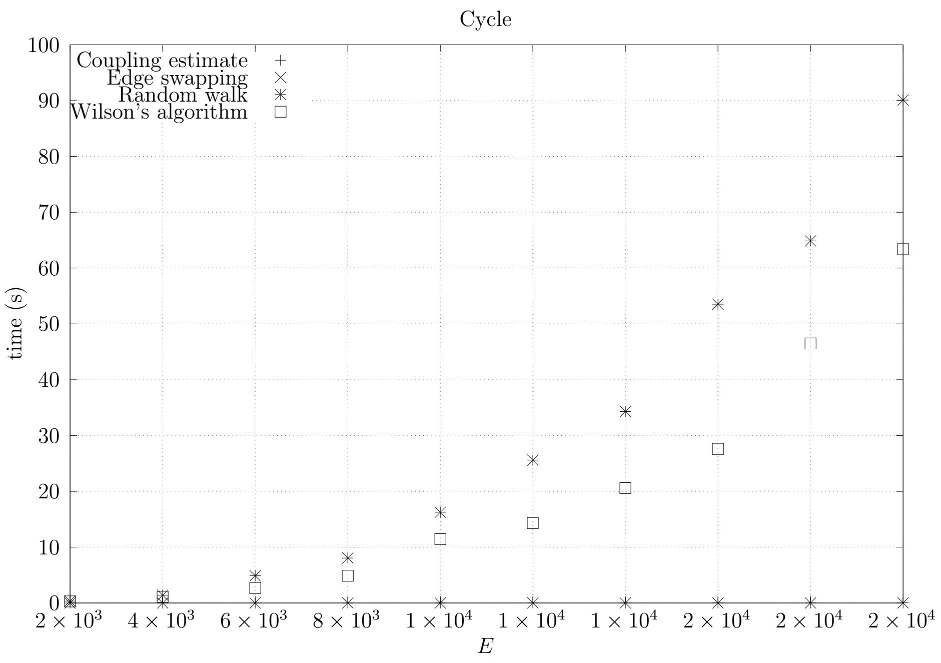

Figure 28. Running times for **cycle** graphs averaged over five runs, including the running time for computing the optimistic coupling estimate, the running time for generating a spanning tree based on that estimate, the edge-swapping algorithm, the running time for generating a spanning tree through a random walk, and the running time for Wilson's algorithm.

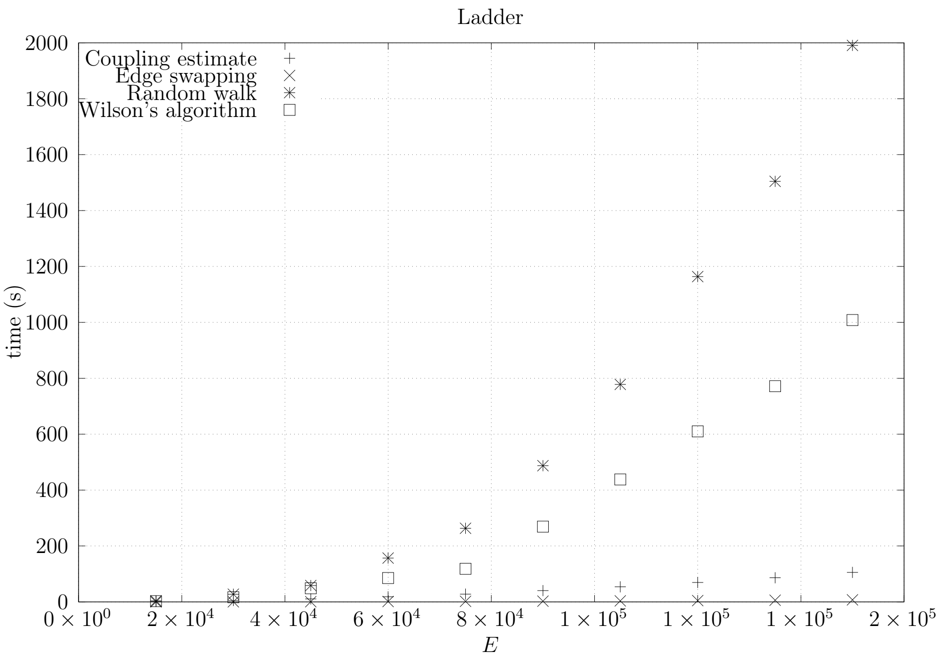

Figure 29. Running times for **sparse** (ladder) graphs averaged over five runs, including the running time for computing the optimistic coupling estimate, the running time for generating a spanning tree based on that estimate, the edge-swapping algorithm, the running time for generating a spanning tree through a random walk, and the running time for Wilson's algorithm.

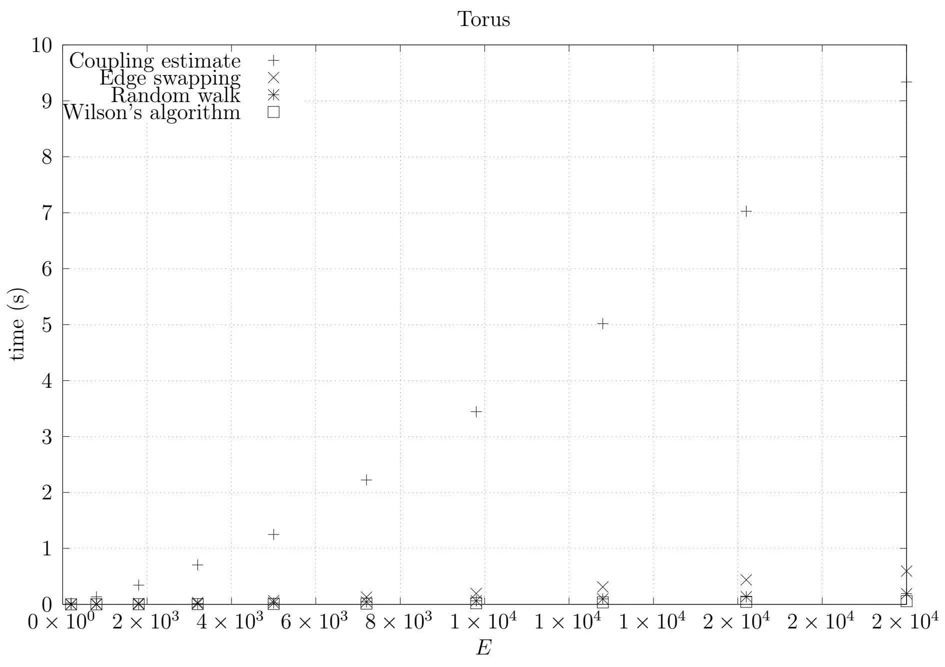

Figure 30. Running times for square **torus** graphs averaged over five runs, including the running time for computing the optimistic coupling estimate, the running time for generating a spanning tree based on that estimate, the edge-swapping algorithm, the running time for generating a spanning tree through a random walk, and the running time for Wilson's algorithm.

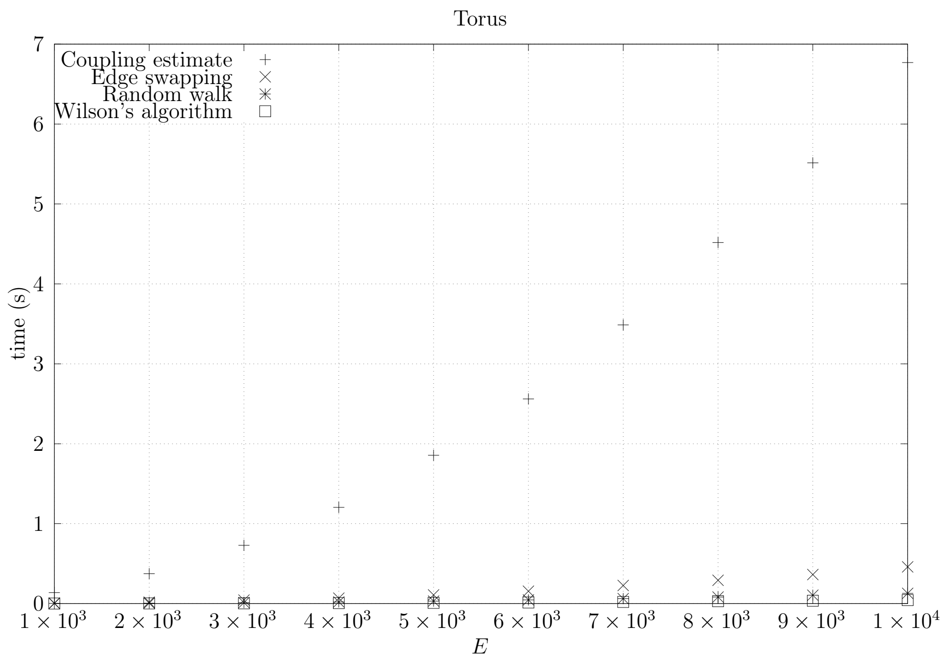

Figure 31. Running times for rectangular **torus** graphs averaged over five runs, including the running time for computing the optimistic coupling estimate, the running time for generating a spanning tree based on that estimate, the edge-swapping algorithm, the running time for generating a spanning tree through a random walk, and the running time for Wilson's algorithm.

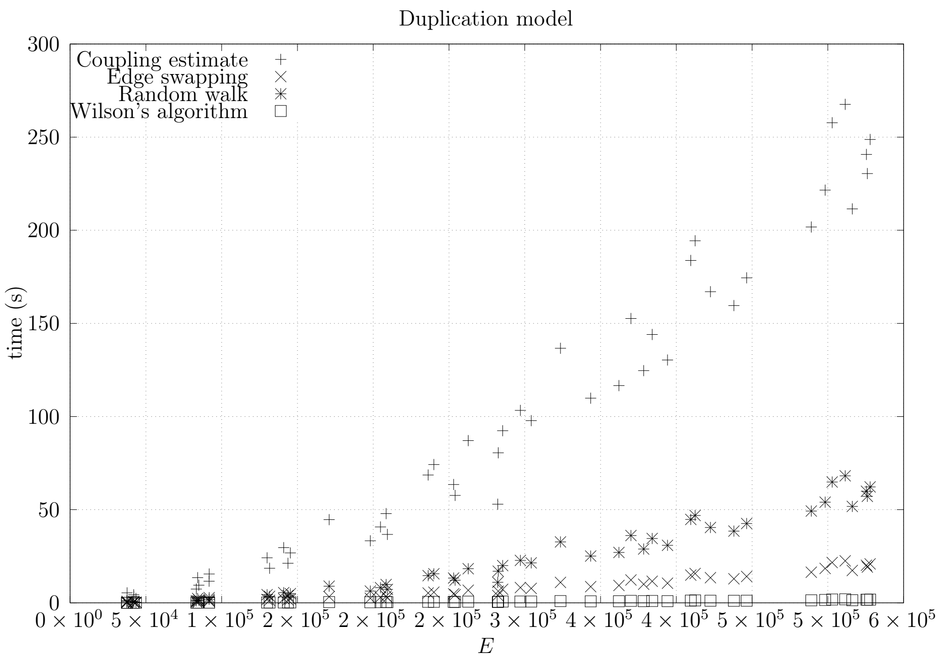

Figure 32. Running times for **dmP** graphs averaged over five runs, including the running time for computing the optimistic coupling estimate, the running time for generating a spanning tree based on that estimate, the edge-swapping algorithm, the running time for generating a spanning tree through a random walk, and the running time for Wilson's algorithm.

5. Related Work

For detailed information on probabilities for trees and networks, see Lyons and Peres [14] (Chapter 4). As far as we know, the initial work on generating uniform spanning trees was performed by Aldous [15] and Broader [16], who obtained spanning trees by performing a random walk on the underlying graph. The former author also further studied the properties of such random trees [17], namely giving general closed formulas for the counting argument we presented in Section 2. In the random walk process a vertex v of G is chosen and at each step this vertex is swapped by an adjacent vertex, where all neighboring vertexes are selected with equal probability. Each time a vertex is visited by the first time, the corresponding edge is added to the growing spanning tree. The process ends when all vertices of G get visited at least once. This amount of steps is known as the cover time of G.

To obtain an algorithm that is faster than the cover time, Wilson [18] proposed a different approach. A vertex r of G is initially chosen uniformly and the goal is to hit this specific vertex r from a second vertex, also chosen uniformly from G. This process is again a random walk, but with a loop erasure feature. Whenever the path from the second vertex intersects itself, all the edges in the corresponding loop must be removed from the path. When the path eventually reaches r it becomes part of the spanning tree. The process then continues by choosing a third vertex and also computing a loop erasure path from it, but this time it is not necessary to hit r precisely: it is enough to hit any vertex on the branch that is already linked to r. The process continues by choosing random vertices and computing loop erasure paths that hit the spanning tree that is already computed.

We implemented the above algorithms as they are accessible. Although several theoretical results have been obtained in recent years, we are not aware of an implementation of such algorithms. We will now survey these results. Another approach to this problem relies on a Theorem by Kirchhoff [19] that counts the number of spanning trees by computing the determinant of a certain matrix, related to the graph G. This relation was studied by Guénoche [20] and Kulkarni [21], who yielded an $O(EV^3)$ time algorithm. This result was improved to $O(V^{2.373})$ by Colbourn et al. [22,23], where the exponent corresponds to the fastest algorithm to compute matrix multiplication. Improvements on the random walk approach were obtained by Kelner and Mądry [24], and Mądry [25], culminating in an $\tilde{O}(E^{o(1)+4/3})$ time algorithm by Madry et al. [26], which relies on insight provided by the effective resistance metric. Interestingly, the initial work by Broader [16] contains a reference to the edge-swapping chain we presented in this paper (Section 5, named the swap chain). The author mentions that the mixing time of this chain is $E^{O(1)}$, although the details are omitted. This chain was extensively studied by several authors, namely in the context of balanced matroids [27–29]. The most recent upper bound on the mixing time is $O(EV \log V)$ [30]. Using Theorem 1 yields an $O(EV \log^2 V)$ algorithm.

Even though link-cut trees are well known [5,31] their application to this problem was not established prior to this work. Their initial application was to network flows [32]. We also found another reference to the edge swap in the work of Sinclair [8]. In the proposal of the canonical path technique the author mentions this particular chain as a motivating application for the canonical path technique, although the details are omitted and we were not able to obtain such an analysis.

We considered an LCT version where the auxiliary trees are implemented with splay trees as per Sleator and Tarjan [5], i.e., whereby the auxiliary data structures we mentioned in Section 3 are splay trees. This means that in step 5 of Algorithm 1 all the vertices involved in the path $C \setminus \{(u,v)\}$ get stored in a splay tree. This path-oriented approach of link-cut trees makes them suitable for our goals, as opposed to other dynamic connectivity data structures such as Euler tour trees [33].

Splay trees are self-adjusting binary search trees, and therefore the vertices are ordered in such a way that the in-order traversal of the tree coincides with the sequence of the vertices that are obtained by traversing $C \setminus \{(u,v)\}$ from u to v. This also justifies why the size of this set can also be obtained in $O(\log V)$ amortized time. Each node simply stores the size of its sub-tree and these values are efficiently updated during the splay process, which consists of a sequence of rotations. Moreover, these values can also be used to `Select` an edge from the path. By starting at the root and comparing the tree sizes to i we can determine if the first vertex of the desired edge is on the left sub-tree, on the root, or on the right sub-tree. Likewise we can do the same for the second vertex of the edge in question. These operations splay the vertices that they obtain and therefore the total time depends on the `Splay` operation. The precise total time of the `Splay` operation is $O((V+1)\log n)$, however the $V \log V$ term does not accumulate over successive operations, thus yielding the bound of $O((V+\tau)\log V)$ in Theorem 1. In general the $V \log V$ term should not be a bottleneck because for most graphs we should have $\tau > V$. This is not always the case; if G consists of a single cycle then $\tau = 1$, but V may be large. Figure 16 shows an example of such a graph.

We finish this section by reviewing the formal definitions of variational distance and mixing time τ [6].

Definition 1. *The variation distance between two distributions D_1 and D_2 on a countable space S is given by*

$$||D_1 - D_2|| = \sum_{x \in S} \frac{|D_1(x) - D_2(x)|}{2} \tag{2}$$

Definition 2. *Let π be the stationary distribution of a Markov chain with state space S. Let p_x^t represent the distribution of the state of the chain starting at state x after t steps. We define*

$$\Delta_x(t) = |p_x^t - \pi| \tag{3}$$

$$\Delta(t) = \max_{x \in S} \Delta_s(t) \tag{4}$$

That is, $\Delta_x(t)$ is the variation distance between the stationary distribution, and p_x^t and $\Delta(t)$ is the maximum of these values over all states x. We also define

$$\tau_x(\varepsilon) = min\{t : \Delta_x(t) \leq \varepsilon\} \tag{5}$$

$$\tau(\varepsilon) = \max_{x \in S} \tau_x(\varepsilon) \tag{6}$$

When we refer only to the mixing time we mean $\tau(1/4)$. Finally the coupling Lemma justifies the coupling approach:

Lemma 4. *Let $Z_t = (X_t, Y_t)$ be a coupling for a Markov chain M on a state space S. Suppose that there exists a T such that, for every $x, y \in S$,*

$$\Pr(X_T \neq Y_T | X_0 = x, Y_0 = y) \leq \varepsilon \tag{7}$$

Then $\tau(\varepsilon) \leq T$. That is, for any initial state, the variation distance between the distribution of the state of the chain after T steps and the stationary distribution is at most ε.

If there is a distance d defined in S then the property $X_T \neq Y_T$ can be obtained using the condition $d(X_T, Y_T) \geq 1$. For this condition we can use the Markovian inequality $\Pr(d(X_T, Y_T) \geq 1) \leq E[d(X_T, Y_T)]$. The path-coupling technique [9] constructs a coupling by chaining several chains, such that the distance between then is 1. Therefore we obtain $d(X_T, Y_T) = d(X_T^0, X_T^1) + d(X_T^1, X_T^2) + \ldots + d(X_T^{D-1}, X_T^D) = 1 + 1 + \ldots + 1$, where $X_T = X_T^0$ and $Y_T = X_T^D$.

6. Conclusions and Future Work

In this paper we studied a new algorithm to obtain the spanning trees of a graph in a uniform way. The underlying Markov chain was initially sketched by Broader [16] in the early study of this problem. We further extended this work by proving the necessary Markov chain properties and using the link-cut tree data structure. This allows for a much faster implementation than repeating the DFS procedure. This may actually be the reason why this approach has gone largely unnoticed during this time.

Using link-cut trees it is possible to generate a spanning tree in $O(EV \log^2 V)$ time for any graph, according to the latest bound on the mixing time of the Markov chain we used. However this result is not competitive against existing alternatives. Instead we studied a coupling approach that yielded much better bounds for graphs consisting of cycles and can be simulated in practice for any graph. We implemented our approach and compared it against existing alternatives. The experimental results show that it is very competitive.

On the one hand, computing the mixing time of the underlying chain is complex, time-consuming and hard to analyze in theory. On the other hand the user of this process can fix a certain number of steps to execute. This is a very useful parameter, as it can be used to swap randomness for time. Depending on the type of application the user may sacrifice the randomness of the underlying trees to obtain faster results or on the contrary spend some extra time to guarantee randomness. Existing algorithms do not provide such a possibility.

As a final note, we point out that our approach can be generalized by assigning weights to the edges of the graph. The edge to be inserted can then be selected with a probability that corresponds to its weight, divided by the global sum of weights. Moreover, the edge to remove from the cycle should be removed according to its weight. The probability should be its weight divided by the sum of the cycle weights. The ergodic analysis of Section 4.1 generalizes easily to this case, so this chain also generates spanning trees uniformly, although the analysis of the coupling of Section 4.2 might need some adjustments. A proper weight selection might obtain a faster mixing timer, possibly something similar to the resistance of the edge.

Acknowledgments: This work was partly supported by the BacGenTrack (TUBITAK/0004/2014) project funded by FCT (Fundação para a Ciência e a Tecnologia) and TUBITAK (Scientific and Technological Research Council of Turkey), by PRECISE (LISBOA-01-0145-FEDER-016394) project co-funded by FEEI (Fundos Europeus Estruturais e de Investimento) from "Programa Operacional Regional Lisboa 2020" and by national funds from FCT

(UID/CEC/500021/2013 and Pest-OE/EEI/LA0021/2014) and European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 690941.

Author Contributions: Luís M. S. Russo and Alexandre P. Francisco conceived the study, performed the theoretical analysis and designed the experiments; Luís M. S. Russo implemented the prototype; Andreia Sofia Teixeira and Alexandre P. Francisco constructed the test cases, performed the experiments and the experimental analysis; Luís M. S. Russo and Alexandre P. Francisco wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Aigner, M.; Ziegler, G.M.; Quarteroni, A. *Proofs from the Book*; Springer: Berlin/Heidelberg, Germany, 2010; Volume 274.
2. Borchardt, C.W. *Über Eine Interpolationsformel für Eine Art Symmetrischer Functionen und über Deren Anwendung*; Math. Abh. der Akademie der Wissenschaften zu Berlin: Berlin, Germany, 1860; pp. 1–20.
3. Cayley, A. A theorem on trees. *Q. J. Math.* **1889**, *23*, 376–378.
4. Galler, B.A.; Fisher, M.J. An improved equivalence algorithm. *Commun. ACM* **1964**, *7*, 301–303. [CrossRef]
5. Sleator, D.D.; Tarjan, R.E. Self-adjusting binary search trees. *J. ACM* **1985**, *32*, 652–686. [CrossRef]
6. Mitzenmacher, M.; Upfal, E. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*; Cambridge University Press: New York, NY, USA, 2005.
7. Levin, D.A.; Peres, Y. *Markov Chains and Mixing Times*; American Mathematical Society: Providence, RI, USA, 2017; Volume 107.
8. Sinclair, A. Improved bounds for mixing rates of Markov chains and multicommodity flow. *Comb. Probab. Comput.* **1992**, *1*, 351–370. [CrossRef]
9. Bubley, R.; Dyer, M. Path coupling: A technique for proving rapid mixing in Markov chains. In Proceedings of the 38th Annual Symposium on Foundations of Computer Science, Miami Beach, FL, USA, 20–22 October 1997; pp. 223–231.
10. Kumar, V.S.A.; Ramesh, H. Markovian coupling vs. conductance for the Jerrum-Sinclair chain. In Proceedings of the 40th Annual Symposium on Foundations of Computer Science, New York, NY, USA, 17–19 October 1999; pp. 241–251.
11. Jerrum, M.; Sinclair, A. Approximating the permanent. *SIAM J. Comput.* **1989**, *18*, 1149–1178. [CrossRef]
12. Chung, F.R.K.; Lu, L.; Dewey, T.G.; Galas, D.J. Duplication models for biological networks. *J. Comput. Biol.* **2003**, *10*, 677–687. [CrossRef]
13. Chung, F.R.; Lu, L. *Complex Graphs and Networks*; American Mathematical Society: Providence, RI, USA, 2006; No. 107.
14. Lyons, R.; Peres, Y. *Probability on Trees and Networks*; Cambridge University Press: Cambridge, UK, 2016; Volume 42.
15. Aldous, D.J. The random walk construction of uniform spanning trees and uniform labelled trees. *SIAM J. Discret. Math.* **1990**, *3*, 450–465. [CrossRef]
16. Broader, A. Generating random spanning trees. In Proceedings of the IEEE Symposium on Fondations of Computer Science, Research Triangle Park, NC, USA, 30 October–1 November 1989; pp. 442–447.
17. Aldous, D. A random tree model associated with random graphs. *Random Struct. Algorithms* **1990**, *1*, 383–402. [CrossRef]
18. Wilson, D.B. Generating random spanning trees more quickly than the cover time. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (STOC '96), Philadelphia, PA, USA, 22–24 May 1996; ACM: New York, NY, USA, 1996; pp. 296–303.
19. Kirchhoff, G. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Ann. Phys.* **1847**, *148*, 497–508. [CrossRef]
20. Guénoche, A. Random spanning tree. *J. Algorithms* **1983**, *4*, 214–220. [CrossRef]
21. Kulkarni, V. Generating random combinatorial objects. *J. Algorithms* **1990**, *11*, 185–207. [CrossRef]
22. Colbourn, C.J.; Day, R.P.J.; Nel, L.D. Unranking and ranking spanning trees of a graph. *J. Algorithms* **1989**, *10*, 271–286. [CrossRef]
23. Colbourn, C.J.; Myrvold, W.J.; Neufeld, E. Two algorithms for unranking arborescences. *J. Algorithms* **1996**, *20*, 268–281. [CrossRef]

24. Kelner, J.A.; Mądry, A. Faster generation of random spanning trees. In Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science, Atlanta, GA, USA, 24–27 October 2009; pp. 13–21.
25. Mądry, A. From Graphs to Matrices, and Back: New Techniques For Graph Algorithms. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2011.
26. Mądry, A.; Straszak, D.; Tarnawski, J. Fast generation of random spanning trees and the effective resistance metric. In Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015), San Diego, CA, USA, 4–6 January 2015; Indyk, P., Ed.; pp. 2019–2036.
27. Feder, T.; Mihail, M. Balanced matroids. In Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, Victoria, BC, Canada, 4–6 May 1992; ACM: New York, NY, USA, 1992; pp. 26–38.
28. Jerrum, M.; Son, J.-B.; Tetali, P.; Vigoda, E. Elementary bounds on poincaré and log-sobolev constants for decomposable markov chains. *Ann. Appl. Probab.* **2004**, *14*, 1741–1765. [CrossRef]
29. Mihail, M. Conductance and convergence of markov chains-a combinatorial treatment of expanders. In Proceedings of the 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, NC, USA, 30 October–1 November 1989; pp. 526–531.
30. Jerrum, M.; Son, J.-B. Spectral gap and log-sobolev constant for balanced matroids. In Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, Vancouver, BC, Canada, 19 November 2002; pp. 721–729.
31. Sleator, D.D.; Tarjan, R.E. A data structure for dynamic trees. In Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing (STOC '81), Milwaukee, WI, USA, 11–13 May 1981; ACM: New York, NY, USA, 1981; pp. 114–122.
32. Goldberg, A.V.; Tarjan, R.E. Finding minimum-cost circulations by canceling negative cycles. *J. ACM* **1989**, *36*, 873–886. [CrossRef]
33. Henzinger, M.R.; King, V. Randomized dynamic graph algorithms with polylogarithmic time per operation. In Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, Las Vegas, NV, USA, 29 May–1 June 1995; ACM: New York, NY, USA, 1995; pp. 519–527.