Efficient Uniform Negative Edge Weights

Lukas Geis
Goethe University Frankfurt, Germany

Daniel Allendorf
Goethe University Frankfurt, Germany

Thomas Bläsius
Karlsruhe Institute of Technology (KIT), Germany

Alexander Leonhardt
Goethe University Frankfurt, Germany

Ulrich Meyer
Goethe University Frankfurt, Germany

Manuel Penschuck
University of Southern Denmark, Odense, Denmark

Hung Tran
Frankfurt Institute for Advanced Studies, Germany

Abstract

We consider a maximum entropy edge weight model that allows for negative weights. Given a graph G and possible weights $\mathcal{W}$ typically consisting of positive and negative values, the model selects edge weights $w \in \mathcal{W}^m$ uniformly at random from all weights that do not introduce a negative cycle. We propose an MCMC process and show that it converges to the required distribution. We then engineer an implementation of the process using a dynamic version of Johnson's algorithm in connection with a bidirectional Dijkstra search as well as an innovative resampling method. We empirically study the performance characteristics of these novel sampling algorithms as well as the output produced by the model.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Random Graphs, Shortest Path, Random Edge Weights, Negative Cycles

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.18

Related Version *Full Version*: https://arxiv.org/abs/2410.22717

Supplementary Material *Software (Generator source and analysis scripts / Experimental Data (Uncompressed 35GB / Compressed 12GB))*: https://doi.org/10.5281/zenodo.21128649

Funding This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG) – ME 2088/5-1, ME-2088/5-2 (FOR 2975 – Algorithms, Dynamics, and Information Flow in Networks), the LOEWE program of the state Hesse (CMMS), and the Novo Nordisk Foundation grant NNF21OC0066551.

Acknowledgements In memory of Daniel, whose sharp analytical mind and keen insights profoundly influenced this work. We thank Deepak Ajwani and Ryan O'Conner for valuable discussions on shortest path algorithms with negative weights that triggered this research, and Mia Ischebeck for her preliminary investigation of a related MCMC process.

1 Introduction

Shortest path problems are a fundamental algorithmic challenge with applications in network routing, geographical navigation, and other optimization tasks. Classic solutions such as Dijkstra's algorithm [14] and most research into practical routing tools operate under the constraint of non-negative edge weights.

Yet, real-world scenarios, such as the routing of recuperating electric vehicles with energy gains and expenditures, naturally present graphs with negative weights; other shortest path problems with negative weights arise, for instance, in the context of flow problems (e. g. [16]). For meaningful shortest paths to exist, graphs should not contain cycles of negative length, as otherwise, paths of unbounded negative weight arise.

Analytical and empirical studies of shortest path algorithms frequently involve random graph models (e. g. [7,18]) as they tend to have a well-understood and mathematically tractable structure [2, 39]. Typically, the graph topology and edge weights are considered separately. While considerable efforts were put towards "realistic" random graphs (e. g. [15] for a survey), maximum entropy network models are frequently used for hypothesis testing, baselines, and null models [21, 28, 32]. Applications in biology [12] and reinforcement learning [30] indicate that in practice they can improve inference quality in different domains, either by increasing the models' robustness or by decreasing their bias.

To maximize the entropy of random edge weights, we consider weights sampled independently and uniformly from some set $\mathcal{W}$; typically, the interval $\mathcal{W} = [a, b]$ is a plausible choice. In the non-negative case ($a \geq 0$), a straightforward sampling method is to process each edge in isolation. This does not hold if a significant probability mass goes towards negative weights ($a < 0$ and $|a|/(b - a) \gg 0$), since cycles of negative weights are likely to appear in many graphs. For example, consider a doubly-linked path, i. e., a graph of nodes $v_1, \ldots, v_{k+1}$ with edges (v_i, v_{i+1}) and (v_{i+1}, v_i) for all $1 \leq i \leq k$. Assuming random i.i.d. edge weights with $\mathcal{W} = \{-1, 0, 1\}$, each pair is randomly assigned one out of $|\mathcal{W}^2| = 9$ possibilities; exactly six of these choices are legal as they have non-negative sums. Hence, the probability of no negative cycles vanishes with an upper bound of $(6/9)^k$.

1.1 Our contribution

In Section 2, we extend the maximum entropy model for signed edge weights without introducing negative cycles. We then propose a basic Markov chain process to sample random negative edge weights in Section 3 and show that the process converges to a uniform distribution. In Section 4, we discuss multiple algorithmic optimizations to accelerate the sampling process. These involve a simple-to-implement dynamic variant of Johnson's algorithm including an application of a bidirectional SSSP search. We then also show how to further bias the process to force a faster convergence using a novel resampling method without losing uniformity. Lastly, in Section 5, we empirically study the performance and convergence of the MCMC process as well as the properties of the output produced. This yields a highly optimized sampling algorithm, which we provide ready-to-use on `https://codeberg.org/lukasgeis/unew`.

1.2 Preliminaries and notation

Let $G = (V, E)$ be a *directed graph* with $E \subseteq V \times V$. If unambiguous from context, we use n and m to denote the number of nodes $|V|$ and edges $|E|$, respectively. Given a graph $G = (V, E)$, we call $w \colon E \to \mathbb{R}$ an *edge weight* function (short weights); by fixing some implicit order on E, we represent the function as a vector $w \in \mathbb{R}^m$. We denote new weights, that differ from w only in one position corresponding to a single edge e, as $w_{e \leftarrow c}$.

We call $P = (v_1, \ldots v_k) \in V^k$ a k-path, if all edges (v_i, v_{i+1}) exist and "overload" the weight function for paths as $w(P) = \sum_i w((v_i, v_{i+1}))$. A path P is *negative* if $w(P) < 0$. A $(k + 1)$-path is a *k-cycle* if $v_{k+1} = v_1$.

For a graph G with weights w, we say (G, w) *is consistent* if no negative cycles exist. Two vertices $u, v \in V$ are *strongly connected* if there exists a directed $u \to v$ path and vice versa. Being strongly connected is an equivalence relation on V and the equivalence classes are called *strongly connected components* (SCC). A graph is *strongly connected* if it has only one strongly connected component.

1.3 Related work

Sampling negative edge weights was studied before (e. g. [10, 13]), typically as a secondary consideration to synthesize benchmark data sets. Yet, we are not aware of any existing maximum entropy models for negative edge weights in shortest path networks. However, given the conceptual simplicity of our sampling algorithm, similarities with existing work are expected. Notably, [13] empirically study shortest path under edge weight updates and, among others, use their `gen_seq` algorithm (described in five lines of prose) to produce update sequences. We expect that such sequences behave similarly to the transitions in our Markov chain proposed in Section 3. However, given the vastly different scope of their work, the authors did not analyse the process nor the properties of the output.

As we are unaware of previous work on maximum entropy models for negative edge weights, we mainly review the practical need for such networks and shortest path algorithm. We do, however, note that there is extensive research into Monte-Carlo-Markov-Chains (MCMC) for approximate uniform samplers of different graph problems (e. g. [8, 20, 33, 36]).

A clear strength of such MCMC solutions are their simplicity and relative ease with which additional constraints can be added (e. g. [1, 35, 40]). Notably, [40] augment Edge Switching with graph connectivity tests. This bears some resemblance to our proposal as both processes frequently execute reachability/shortest path computations. However, aside from addressing different problems, the methods also differ in technical details.

Networks with negative edge weights. A prominent application are road networks for electrical cars where downward-directed roads allow the vehicles to recuperate their batteries. A recent survey summarized a plethora of applications and research directed towards negative edge weights in social networks [26]. Here, positive edge weights might correspond to friendship or affection while negative edge weights relate to opposition and aversion. This yields new measures of power in politically charged networks [34] and centrality measures respecting negative ties [17].

Several "nature inspired" techniques produce shortest path networks with negative weights. For instance, a common technique derives edge weights from the gradient of random node potentials. This, however, needs further refinements (e. g. by adding positive random noise) since otherwise, each cycle has weight exactly 0 which seems implausible for physical networks.

Shortest path algorithms. The SSSP algorithms Dijkstra [14], and BellmanFord [3, 23] are classical, but still relevant solutions. With a runtime of $\mathcal{O}(m + n \log(n))$, Dijkstra is asymptotically faster than the $\mathcal{O}(nm)$ runtime of BellmanFord. Still, Dijkstra has the significant drawback of only supporting non-negative edge weights. Johnson's all-pair shortest path algorithm implements an ingenious workaround [22]. It translates general edge weights solely to non-negative weights, such that a shortest path in one network translates to a shortest path in the other (see Lemma 6 for details).

In addition, the recent theoretical breakthrough result of Bernstein et al. [4] presents an algorithm that solves the SSSP problem on networks with negative edge weights but without negative cycles in near-linear time. This recent addition stimulated subsequent (partial) implementations thereof [9] which had to resort to benchmarking their algorithmic engineering efforts on (biased) weight distributions in lack of a maximum-entropy generator.

Algorithm 1 RejSampler: Approximate uniform sampler.

Input: graph $G = (V, E)$, weight interval $\mathcal{W}$, number of steps τ
Output: consistent weights $w\colon E \to \mathcal{W}$

```
1 w ← arbitrary consistent weight function
2 for t ← 1 to τ do
3     e ← uniform random edge in E
4     c ← uniform random weight in W
5     if (G, w_{e←c}) is consistent then
6         accept w ← w_{e←c}                    // update weight of e
7 return w
```

2 The Negative Edge Weight Model

We want to uniformly sample from the set of all edge weights consistent with G:

▶ **Definition 1.** *Given graph G and weights $\mathcal{W}$, define the set of consistent edge weights as*

$$\mathbb{S}_{G,\mathcal{W}} = \{w \mid w\colon E \mapsto \mathcal{W} \ \wedge \ (G, w) \text{ is consistent}\}\,.$$

Then, $\mathcal{S}_{G,\mathcal{W}}$ is the uniform distribution over $\mathbb{S}_{G,\mathcal{W}}$. If unambiguous, we shorten to $\mathbb{S}$ and $\mathcal{S}$.

Assumptions. For ease of exposition, we assume without loss of generality that (A1) G is strongly connected and (A2) $\mathcal{W}$ contains positive and negative values. If (A1) does not hold, we may decompose G into strongly connected components and a residual acyclic graph in linear time [37], process the former independently, and trivially sample weights for the latter. As for (A2), observe that sampling from $\mathcal{S}_{G,\mathcal{W}}$ is trivial if G is acyclic or if $\mathcal{W}$ does only contain positive values, as neither case can have negative cycles. Further, if $\mathcal{W}$ consists only of negative values, consistent weights exist iff G is acyclic.

Variants. It is natural to ask whether – for some number $k \in \mathbb{N}$ – there exists a $w \in \mathbb{S}$ that has k negative edges. Unfortunately, this problem is NP-hard. This can be seen by a simple reduction from the NP-hard problem *Directed Feedback Arc Set* [25], which asks whether one can remove k edges from a directed graph to obtain an acyclic graph. This is the case iff there exists a consistent weight function w assigning negative weights to $m - k$ edges. By definition, the graph induced by the negative edges is acyclic; removing the k positive edges thus suffices to obtain an acyclic graph. Conversely, assume that removing k edges results in an acyclic graph. Then assigning large positive weights to these k edges and small negative weights to the remaining edges results in a consistent weight function.

3 The MCMC sampler

Unfortunately, a direct sampling method of $\mathcal{S}$ is highly non-trivial as each cycle imposes a constraint on the weight function. Since there are possibly an exponential number of cycles in a graph and only an exponentially small fraction of all weights in $\mathcal{W}^m$ appear in $\mathbb{S}$, we instead opt for an approximate sampler using a Markov-Chain Monte-Carlo (MCMC) process. We introduce Algorithm 1 to sample approximately uniformly from $\mathbb{S}_{G,\mathcal{W}}$ for a strongly connected graph G and partially negative values $\mathcal{W}$ (wlog; see Section 2).

We start with arbitrary weights $w \in \mathbb{S}_{G,\mathcal{W}}$; canonical choices include setting all edge weights to 0 (if $0 \in \mathcal{W}$), 1 (if $1 \in \mathcal{W}$), the largest value in $\mathcal{W}$, or a randomly chosen non-negative value in $\mathcal{W}$. Subsequently, we perturb the solution by repeating the following step τ times: we select a uniform random edge $e \in E$ and new weight $c \in \mathcal{W}$. Then, if $(G, w_{e \leftarrow c})$ remains consistent, we update the edge and reject otherwise.

For simplicity, we assume in the following that the weight interval $\mathcal{W}$ is discrete and finite, thus rendering the state space finite. The spirit of all arguments and results, however, carries over to continuous $\mathcal{W}$. The state space of the MCMC in Algorithm 1 consists of exactly all consistent weight functions in $\mathbb{S}$. The neighborhood $N[w]$ of state w consists of all weight functions that differ in at most one position:

$$N[w] = \{w' \mid \exists e \in E\ \exists c \in \mathcal{W}\colon w' = w_{e \leftarrow c} \in \mathbb{S}\}$$

▶ **Observation 2** (Symmetry). *Weights $w_1, w_2 \in \mathbb{S}$ satisfy $w_1 \in N[w_2] \Leftrightarrow w_2 \in N[w_1]$.*

▶ **Observation 3.** *Let $w_1 \in \mathbb{S}$ and $w_2 \in \mathcal{W}^m$, s.t. $\forall e \in E\colon\ w_1(e) \leq w_2(e)$. Then, $w_2 \in \mathbb{S}$. We can reach w_2 from w_1 by increasing the weight in all differing positions in any order.*

▶ **Theorem 4.** *Let $G = (V, E)$ be a directed graph and $\mathcal{W}$ a weight interval. Then, the MCMC process converges to a uniform distribution on $\mathbb{S}_{G,\mathcal{W}}$.*

Proof. We use the fact that any irreducible, aperiodic, and symmetric Markov chain converges to a uniform distribution [29, Th. 7.10]. We only show these properties:

The MC is irreducible iff its state graph is strongly connected. Let $w_1, w_2 \in \mathbb{S}$ be states and $w_{\max}$ the function that assigns every edge the maximum weight $\max(\mathcal{W})$. By Observation 3, we know that there exist paths P_i from w_i to $w_{\max}$. Then, by Observation 2, the opposite paths P_i^r from $w_{\max}$ to w_i also exist. We can therefore reach w_2 from w_1 by P_1 followed by P_2^r (and vice versa). △

An irreducible MC is aperiodic if there exists a self-loop. In fact, each state w has a loop: If we sample edge e and the old weight $c = w(e)$, no transition occurs. △

The MC is symmetric iff the probability of transition between any two adjacent states is identical in both directions. Let $w_1, w_2 \in \mathbb{S}, w_1 \in N[w_2]$. If $w_1 = w_2$, the property holds trivially. For $w_1 \neq w_2$, there is a unique $e \in E$ with $w_1(e) \neq w_2(e)$. To transition between w_1 and w_2, we need to draw e with probability $1/m$ and $w_1(e)$ or $w_2(e)$ with $1/|\mathcal{W}|$. ◀

We show in the full version (appendix) that (i) Theorem 4 also holds true for uncountable $\mathcal{W}$ and (ii) the process is rapidly mixing on the simple n-cycle graph.

4 Algorithmic insights

Most of Algorithm 1 can be efficiently implemented using standard techniques. We maintain the graph in the *compressed sparse row* (CSR) format [38], allowing fast neighborhood iteration as well as constant time uniform sampling of an edge.

The most expensive task of Algorithm 1 is to test whether the updated weight function w' remains consistent with G. To this end, we need to verify that w' induces no negative cycle. As discussed in Section 1.3, this problem has already been considered for several applications. However, we are unaware of solutions that utilize bidirectional searches, which often perform significantly better as shown in Section 5. In the following, we begin with a straightforward unidirectional implementation designed for the MCMC process and highlight connections to previous work. Subsequently, we extend the algorithm to incorporate a bidirectional search. Finally, we alter the algorithm to always sample an acceptable weight uniformly for an edge.

Consider a weight update $w' = w_{e \leftarrow c}$ on edge e. In the following, we focus on decreased weights (i. e., $w'(e) < w(e)$), since increased weights cannot introduce negative cycles by Observation 3. BellmanFord [3, 23] can detect negative cycles in w' directly. However, even an optimized implementation performs subpar. The primary issue is BellmanFord's lack of locality – the algorithm typically performs several global rounds despite the fact that negative cycles tend to be short.

Thus, we want to switch to Dijkstra and later its bidirectional variant; neither of which supports negative weights. As a first step, we show how to detect negative cycles in w' using only the previous weights w. The main insight is inductive in nature: as the current (G, w) is consistent (otherwise they would have been rejected), we know that the candidate (G, w') is consistent iff the updated edge e is not part of a negative cycle. This can be tested using:

▶ **Observation 5.** *Let P be a shortest $v \to u$ path in (G, w). Obtain $w' = w_{e \leftarrow c}$ by updating edge $e = (u, v)$. Then, e is in a negative cycle for w' iff $w'(u, v) + w(P) < 0$.*

4.1 An online version of Johnson's algorithm

If none but the new weight $w'(e)$ was negative, we could use Dijkstra based on Observation 5 with a running time of $\mathcal{O}(m + n \log n)$. Even better, we can heavily prune the search space and ignore all paths that have a weight of at least $-w'(e)$: Since all unprocessed edges have a non-negative weight, they cannot lead to strictly lighter paths than the ones already discovered (i. e., the smallest element in Dijkstra's priority queue).

However, since w may contain negative weights, we recast the shortest path problem into an equivalent non-negative analog using the potential method [4, 22]. The following lemma provides the core ingredient for this to work and also forms the basis for our algorithm. The function ϕ in the lemma statement is also called *potential function*.

▶ **Lemma 6** (based on [22]). *Let $G = (V, E)$ be a graph with weights $w \colon E \to \mathbb{R}$ and let $\phi \colon V \to \mathbb{R}$. Further, let $w_\phi(u, v) = w(u, v) + \phi(v) - \phi(u)$. Then, any path in G is a shortest path with respect to w if and only if it is a shortest path with respect to w_ϕ.*

Proof. Let $P = (v_1, \ldots, v_p)$ be a path in G. The path weight $w_\phi(P)$ follows as a telescope sum that cancels out all contributions but the first and the last potential:

$$\begin{aligned} w_\phi(P) &= \sum_{i=1}^{p-1} [w(v_i, v_{i+1}) - \phi(v_i) + \phi(v_{i+1})] \\ &= \sum_{i=1}^{p-1} w(v_i, v_{i+1}) - \sum_{i=1}^{p-1} \phi(v_i) + \sum_{i=2}^{p} \phi(v_i) = w(P) - \phi(v_1) + \phi(v_p) \end{aligned} \tag{1}$$

The remaining potentials $\phi(v_1)$ and $\phi(v_p)$ appear on all $v_1 \to v_p$ paths. Thus, a shortest path for w_ϕ remains a shortest path in terms of w (and vice versa). ◀

We call a potential function ϕ *feasible* if $w_\phi(u, v) \geq 0 \; \forall (u, v) \in E$. Further, we call $w_\phi(u, v)$ the *potential weight* of (u, v). We say edge (u, v) is *broken* if it has a negative *potential weight*.

4.2 Maintaining a feasible potential

A feasible potential function ϕ can be computed by adding a node r with outgoing edges of weight 0 to all existing nodes. Then, SSSP starting in r yields a feasible ϕ by setting $\phi(u)$ to minus the length of the shortest $r \to u$ path found [22]. We avoid this step by starting the MCMC process with non-negative edge weights. Then all potentials can be set to 0.

Input:
Since nodes u and v have same potential, the edge (u, v) is broken: $w_\phi(u, v) = w(u, v) = -2$.

Solution 1 (using Dijkstra):
increase only $\phi(v) + 2$. This large update causes "many" cascading updates

Solution 2 (using BiDijkstra):
split update to $\phi(u) - 1$ and $\phi(v) + 1$. This update causes fewer cascading updates as z and y remain unchanged.

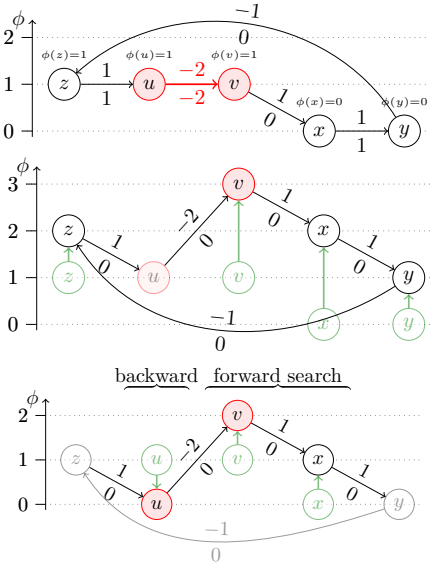

Figure 1 Fixing potential on a cycle graph. The number above edge (a, b) is $w(a, b)$ and below $w_\phi(a, b) = w(a, b) + \phi(b) - \phi(a)$; green nodes indicate the input potential (top panel). Edge (u, v) is broken in the input. It needs an increase of $\phi(v) - \phi(u)$ by at least $-w_\phi(u, v) = 2$ to make ϕ feasible.

We thus focus on the dynamic problems of (i) testing for negative cycles and (ii) maintaining feasible potentials after edge weight updates. The following lemma allows us to answer (i) as a direct byproduct of (ii) and is similar to [16, Thm. 3]:

▶ **Lemma 7.** *If ϕ is a feasible potential function for G and w, (G, w) is consistent.*

Proof. Consider any cycle C in G and observe that we can interpret C as a path with $v_1 = v_p$. Then, Equation (1) implies that the weight of C is independent of the potential function, i. e., $w_\phi(C) = w(C) + \phi(v_p) - \phi(v_1) = w(C)$. Then, if ϕ is feasible, every edge in G with respect to w_ϕ has a non-negative weight. Thus, every cycle must also have a non-negative total weight. But since $w_\phi(C) = w(C)$ for every cycle C, there cannot exist a negative weight cycle in G if ϕ is feasible. ◀

Thus, our optimized implementation maintains a potential function ϕ and checks whether an update is consistent by attempting to maintain a feasible ϕ. Further, we only update ϕ under weight decreases since a weight increase only increases the potential weight of the edge and thus cannot break *feasibility* of ϕ.

However, as illustrated in Figure 1, decreasing an edge weight $w(u, v)$ may *break* the edge. To make ϕ feasible again, we need to increase the gradient $\phi(v) - \phi(u)$. This can be achieved by (i) decreasing $\phi(u)$, (ii) increasing $\phi(v)$, or (iii) both. We begin with variant (ii) and discuss option (iii) in Section 4.3. In any case, changing one potential may cause "collateral damage" by breaking additional edges. The following lemma shows how to efficiently repair cascading breakages using a single (partial) shortest path query.

▶ **Lemma 8.** *Consider weights w' obtained from w by decreasing the weight of (u, v). If w' has no negative cycles, we can construct a feasible ϕ' with $\phi'(x) = \phi(x) + \max\{0, -w'_\phi(u, v) - D(x)\}$ where $D(x)$ is the length of a shortest $v \to x$ path with respect to w_ϕ.*

Proof. Note that if $w'_\phi(u,v) > 0$, we have $\max\{0, -w'_\phi(u,v) - D(x)\} = 0\ \forall x \in V$ since $D(x)$ is non-negative as ϕ is feasible for w and $w'_\phi(u,v)$ does not introduce a negative cycle. Therefore, ϕ remains feasible without updating.

Hence, assume that $w'_\phi(u,v) \leq 0$. Observe that $\phi'(x) = \phi(x)$ if $D(x) \geq -w'_\phi(u,v)$. Thus, we only need to consider nodes x which have a shortest $v \to x$ path of length at most $-w'_\phi(u,v)$. This allows us to heavily prune the shortest path tree T by only including nodes x with $D(x) \leq -w'_\phi(u,v)$. Because ϕ is feasible for w and due to Observation 5 and Lemma 7, T always exists and does not contain u.

We now show that all edges have non-negative edge weights $w_{\phi'}(\cdot)$. Since we set $\phi'(v) = \phi(v) - w'_\phi(u,v) - D(v) = \phi(v) - w'_\phi(u,v)$, we get $w'_{\phi'}(u,v) = 0$ and (u,v) is no longer *broken*. Now consider any edge $(x,y) \in E$, where the edge (x,y) is

- ... a tree edge in the shortest path tree T (if the weight change does not introduce a negative cycle). Then, the new potential weight $w'_{\phi'}(x,y)$ is

$$\begin{aligned} w'_{\phi'}(x,y) &= w_\phi(x,y) + [-w'_\phi(u,v) - D(y)] - [-w'_\phi(u,v) - D(x)] \\ &= w_\phi(x,y) + D(x) - D(y) \\ &= w_\phi(x,y) + D(x) - [D(x) + w_\phi(x,y)] = 0 \end{aligned} \tag{2}$$

- ... not a tree edge, but both nodes x and y are in T. Observe that $D(x) + w_\phi(x,y) \geq D(y)$ and Equation (2): the new potential weight is non-negative.
- ... not in T, and only x is in T. Then, we know

$$D(x) + w_\phi(x,y) \quad \geq \quad -w'_\phi(u,v) \qquad \Leftrightarrow \qquad w_\phi(x,y) \quad \geq \quad -w'_\phi(u,v) - D(x).$$

  Thus $w'_{\phi'}(x,y) \geq 0$ after only adjusting $\phi(x)$ since the decrease is less than the current value. Further, if we find y, but not x the potential weight can only increase and thus not break the edge.

Finally, if $x, y \notin T$, the non-negative weight $w'_{\phi'}(x,y) = w_\phi(x,y)$ stays unchanged. ◀

▶ **Lemma 9.** *Checking if $w_{(u,v)\leftarrow c} \in \mathbb{S}$ (Line 5 of Alg. 1) runs in time $\mathcal{O}(m + n \log n)$.*

Proof. If an edge weight increases or $w'_\phi(u,v) \geq 0$, we directly accept keeping ϕ unchanged. Otherwise, compute a shortest path tree T from node v and maximum distance $-w'_\phi(u,v)$, i. e., stop if every unvisited node x has a shortest $v \to x$ path of weight of at least $-w'_\phi(u,v)$. If we find the target u with $D(u) < -w'_\phi(u,v)$, reject the update as it causes a negative cycle (c. f. Observation 5 and Lemma 7). If u is not found, i. e., $D(u) \geq -w'_\phi(u,v)$, accept and update ϕ as in Lemma 8. Since ϕ is feasible before, every edge has a non-negative potential weight, and we can use DIJKSTRA to compute T or find a path that leads to rejection. ◀

4.3 Bidirectional search

The algorithm outlined in the previous section executes DIJKSTRA to obtain shortest path distances to nodes with distance at most $-w'_\phi(u,v)$. In practice, we can often further prune the search space: For exposition, assume we sampled weight c for edge (u,v) introducing a negative cycle. We detect this, by finding a $v \to u$ path P with $w(P) + c < 0 \Leftrightarrow w(P) < -c$. We may have found P faster through a bidirectional search, i. e., we run a forward search from node v, and a backward search from u (reversing each edge). Then, interleaving the forward- and backward searches eventually yields a node x with $w_\phi(v \to x) + w_\phi(x \to u) < -w'_\phi(u,v)$. It is known both in theory and practice, that this can reduce the search space dramatically [5,6,7].

As illustrated in Figure 1 (Solution 2), the bidirectional search also often accelerates accepted updates, especially if there is a broken edge. Consider an update of $w(u,v)$ that breaks the edge and requires increasing the gradient $\phi(v) - \phi(u)$ by $-w'_\phi(u,v)$. Following the

unidirectional pattern of DIJKSTRA, we previously only increased $\phi(v)$ and kept $\phi(u)$. The main insight is that we can achieve the same effect by distributing the changes over both endpoints, i. e., by reducing $\phi(u) - \Delta_u$ and increasing $\phi(v) + \Delta_v$ with $\Delta_u + \Delta_v = -w'_\phi(u, v)$.

This strategy directly repairs the broken edge (u, v) and may still break out-edges of v (and successors). As before, these cascading defects can be dealt with using Lemma 8. Additionally, the new strategy can also break in-edges of u (and predecessors). These can be fixed through a mirrored variant of Lemma 8 by decreasing (rather than increasing) potentials according to $x \to u$ paths (rather than $v \to x$ paths). We give empirical evidence in Section 5 that a bidirectional search results in a significantly faster algorithm.

▶ **Lemma 10.** *For a weight decrease of edge (u, v), let $w'_\phi(u, v)$ be the new potential weight. If w' remains free of negative cycles, we can construct a feasible ϕ' with $\phi'(x) = \phi(x) - \max\{0, -w'_\phi(u, v) - D(x)\}$ where $D(x)$ is the length of a shortest $x \to u$ path with respect to w_ϕ.*

Proof. This lemma is the symmetric version of Lemma 8. The proof is analogous to the proof of Lemma 8 with the inverted graph and potentials. ◀

The distribution between Δ_u and Δ_v does not affect correctness. Our implementation simply alternates between forward– and backward search until it is established that there is no $v \to u$ path of length at most $-w'_\phi(u, v)$. This also yields the disjoint partial shortest path trees T_u around u, and T_v respectively, used for potential updates.

4.4 Rejecting the rejection

As we will later see, the current chain eventually settles on a total rejection rate exceeding 40% in all parameter settings. Hence, in almost 50% of cases, we "discard" our search and do not change w, which significantly slows convergence, as we do nothing while incurring the cost of a search. We, however, can prevent this behavior entirely by extending Algorithm 1 to include an additional re-sampling step of c *after* "rejecting" in line 5. We again point to Observation 5 and note that we accept any new weight c for $e = (u, v) \in E$ iff $c \in \mathcal{W}_e$ where $\mathcal{W}_e$ is defined below.

▶ **Definition 11.** *For edge $e = (u, v) \in E$ and consistent weights w, define the set of acceptable weights as $\mathcal{W}_e := \{x \in \mathcal{W} \mid x \geq -w(P)\}$ where P is the shortest $v \to u$ path in (G, w).*

Hence, to resample an acceptable new weight c' for $w(e)$, we can use the path length $w(P)$, which we computed to reject the originally proposed weight c, to lower bound c'. We dub this altered scheme NOREJSAMPLER.

Unfortunately, in practice, one might not compute the true shortest $v \to u$ path P in (G, w) (and thus $\mathcal{W}_e$) to reject c but rather any $v \to u$ path P' with $c < -w(P')$ (e. g. reject at the first observed smaller path). Therefore, to guarantee that the newly resampled weight c' also does not induce a negative cycle, we continue with the (rejecting) search in two ways:

- **(i)** Continue the search until the truly shortest $v \to u$ path P is found and resample c' once afterwards in the correct set $\mathcal{W}_e$; or:
- **(ii)** Directly resample c' in a superset of $\mathcal{W}_e$ and continue the search until you either accept or reject and resample c' again with now better bounds before repeating this procedure.

While option (i) only requires one resampling, it also requires us to always find the shortest $v \to u$ path, which can be expensive. On the other hand, option (ii) might need multiple resampling steps, but also still allows for early termination by acceptance before fully

discovering P.[1] We will see in Section 5 that option (ii) is by far the most efficient version for our process. Note that in all cases (original chain, option (i), option (ii)), we only accept iff the (final) sampled weight lies in $\mathcal{W}_e$. We dub option (i) and (ii), MinCycle and Resamp, respectively.

For completeness, we briefly prove that these variations also converge to the uniform distribution on $\mathbb{S}_{G,\mathcal{W}}$.

▶ **Theorem 12.** *Let $G = (V, E)$ be a directed graph and $\mathcal{W}$ a weight interval. Then, the NoRejSampler process converges to a uniform distribution on $\mathbb{S}_{G,\mathcal{W}}$.*

Proof. The proof follows the same steps as the proof of Theorem 4. Since the process operates on the same state graph as Algorithm 1 and only varies in transition probabilities, properties of irreducibility and aperiodicity still hold, and we only need to show symmetry again:

Let $w_1, w_2 \in \mathbb{S}, w_1 \neq w_2$ with $w_1 \in N[w_2]$. There must exist a unique $e = (u, v) \in E$ with $w_1(e) \neq w_2(e)$. Since w_1 and w_2 only differ in e, and both yield consistent weight functions, the weight of the shortest $v \to u$ path P is identical in (G, w_1) and (G, w_2), and both weights share the same $\mathcal{W}_e$. Furthermore, by Observation 5, we have $w_1(e), w_2(e) \in \mathcal{W}_e$.

By definition of NoRejSampler, the new weight c (or c') is a uniform sample in a superset of $\mathcal{W}_e$, and thus also a uniform sample in $\mathcal{W}_e$ if it gets accepted. Thus, to transition from w_1 to w_2 (w_2 to w_1), we must first sample e with probability $1/m$ and then $w_2(e)$ ($w_1(e)$) with probability $1/|\mathcal{W}_e|$ (even while possibly not knowing $|\mathcal{W}_e|$). ◀

5 Empirical evaluation

We empirically study our chains and the properties of the produced output. We also give performance indications on the previously discussed implementations on random graph models. All performance-critical sections are implemented in Rust[2]. All runtime measurements were taken on a server with an `AMD EPYC 7702P 64-Core` processor with 512GB of RAM. Pseudo-random numbers are generated with the `Pcg64` generator [31].

Parameters. The implementations are relatively straightforward, with the exception of avoiding negative cycles. Here, we implement both the Dijkstra and BiDijkstra variants of RejSampler as well as both discussed variants of NoRejSampler. Since applying Lemma 8 leads to a high number of edges with *potential weight* 0, we modify Dijkstra as follows: If we settle a node, we also directly settle all neighbors reachable by zero-weight edges without pushing them into the priority queue.[3]

We consider Gilbert's $\mathcal{G}(n, p)$ model [19] with $p = \overline{d}/n$ to simulate a graph with average degree $\overline{d}$ which we dub $\mathcal{GNP}(n, \overline{d})$ and the hyperbolic graph model $\mathcal{RHG}(n, \overline{d})$ [27] with dispersion parameter $\alpha = 0$ and $T = 0$, where we replace an undirected edge by two directed edges. We also derive a real-world dataset $\mathcal{ROAD}$ from US state road networks obtained from the TIGER/Line dataset, available on the 9th DIMACS Implementation Challenge website.[4] We replace networks with $n \geq 100\,000$ with a connected induced subgraph of 100 000 nodes, collected via BFS. We also replace each undirected edge by two directed edges.

[1] Implementations of both methods are practically identical. Option (i) replaces the "resampling" step by always "resampling" the lower bound – guaranteeing a continuance of the search until P is found.
[2] Compiled with `rustc-1.96.0-nightly`. Performance roughly on par with C [11].
[3] In initial experiments, this heuristic proved to be **only** effective for Dijkstra.
[4] `https://www.diag.uniroma1.it/challenge9/data/tiger/`

We implement four different initial consistent weight functions: $w_{\max}$ (start with all weights set to $\max(\mathcal{W})$), w_{zero} (start with all weights set to 0), w_{one} (start with all weights set to 1),and w_{unif} (weights are independently drawn uniformly from $[0, \max(\mathcal{W})]$). Unless stated otherwise, we set $\mathcal{W} = [-100, 100] \cap \mathbb{Z}$ and $n = 10\,000$ for $\mathcal{GNP}$ and $\mathcal{RHG}$ graphs. All averages are computed via the arithmetic mean.

5.1 Selecting the best implementation

Our main goal is to study the performances of our different algorithms and chains to find the implementation that yields an approximate uniform distribution the fastest. For that, we first compare the different search algorithms for RejSampler before comparing them against implementations of NoRejSampler. For that, we ran experiments on $\mathcal{GNP}$ and $\mathcal{RHG}$ graphs with average degrees $\overline{d} \in \{10, 20, 50\}$ as well as graphs from $\mathcal{ROAD}$. We start with all four different initial weight functions, repeating each datapoint 100 times.

Table 1 Average speedup of BiDijkstra over Dijkstra after 10^8 rounds (RejSampler).

Graph	$\mathcal{GNP}(n = 10^5)$			$\mathcal{RHG}(n = 10^5)$			$\mathcal{ROAD}$
Degree	10	20	50	10	20	50	
Speedup	27.3052	29.0018	25.8080	13.9757	28.5909	38.7055	1.2643

Optimal Algorithm. First, we want to verify whether a bidirectional search via BiDijkstra does yield a better runtime than a one-directional search via Dijkstra. The implicit assumption that a naive implementation of RejSampler using BellmanFord performs by far the worst was confirmed in initial experiments and is thus no longer considered.

Table 1 supports our hypothesis as in almost all instances, BiDijkstra averages a speedup[5] of more than 25 over Dijkstra, showing a significant runtime improvement. For graphs in $\mathcal{ROAD}$, we observe a smaller speedup of roughly 25% since these graphs are very sparse, leading to narrow search trees in both algorithms.

Table 2 Comparison of BiDijkstra (RejSampler), Resamp, and MinCycle after 10^8 rounds: AccRate is the acceptance rate of RejSampler, Speedup the speedup of Resamp and MinCycle over BiDijkstra, and AccSpeedup the speedup per accepting round (i. e., Speedup/AccRate). Resamples is the average number of times Resamp and MinCycle had to "resample" in a search.

Graph		$\mathcal{GNP}(n = 10^5)$			$\mathcal{RHG}(n = 10^5)$			$\mathcal{ROAD}$
Degree		10	20	50	10	20	50	
AccRate (RejSampler)		0.5921	0.5472	0.5176	0.5465	0.5246	0.5102	0.6557
Speedup	Resamp	0.8620	0.8980	0.8655	0.8392	0.8374	0.8161	0.8580
	MinCycle	0.2861	0.2732	0.2499	0.3199	0.2401	0.1898	0.6959
AccSpeedup	Resamp	1.4561	1.6413	1.6722	1.5363	1.5974	1.5996	1.3087
	MinCycle	0.4832	0.4992	0.4829	0.5855	0.4578	0.3720	1.0612
Resamples (Resamp)		1.2485	1.2539	1.2550	1.1595	1.1890	1.2090	1.0150

[5] Speedup is computed via the total running time (all previous rounds) thus far.

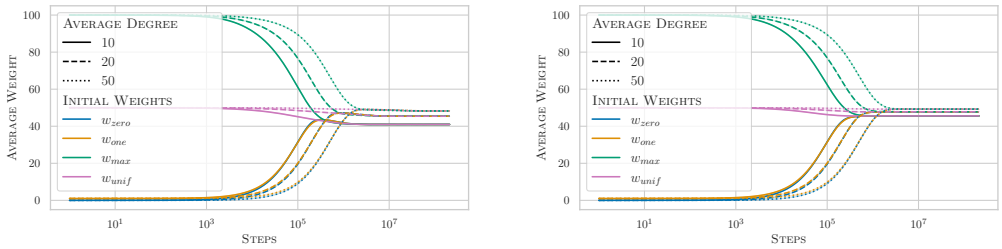

Figure 2 Average weight over time for $\mathcal{GNP}$ (l), $\mathcal{RHG}$ (r).

Optimal Chain. Similarly, we also want to determine whether eliminating rejections from RejSampler yields any practical benefit. While we do expect the average runtime per round to go up, as we potentially do more work per round (resampling in case of rejection, continuing the search), we anticipate that the average runtime per accepted weight change goes down. We show later that this is a reasonable metric as the asymptotic behaviour of the underlying markov chain is identical under accepted weight changes. Given the previously observed superiority of BiDijkstra, we consider only a bidirectional approach going forward.

Table 2 confirms our hypotheses. Firstly, more than one third of all rounds in RejSampler were rejected, sometimes even up to almost 50%. This shows that the underlying Markov chain of RejSampler is almost lazy in most cases. We also confirm that the additional overhead of resampling and continuing the search does slow down Resamp and MinCycle initially by a factor of ~ 0.85 and ~ 0.25, respectively. However, when averaging the runtime over accepted rounds, this flips for Resamp, which now achieves a roughly 50% speedup over RejSampler. Nonetheless, even then, MinCycle is still circa twice as slow as RejSampler. The number of average resamples needed reaffirms this as Resamp on average succeeds with almost only one resample.

Combining results from Tables 1 and 2, for a metric of accepted rounds, a bidirectional Resamp is more than 30× faster than the basic Dijkstra implementation for RejSampler in most cases – barring an even slower naive implementation of BellmanFord. Lastly, for $n = 10^5$, Resamp only requires $4.6\mu s \sim 5.6\mu s$ per round for $\mathcal{GNP}$ graphs and $0.7\mu s \sim 1.2\mu s$ per round for $\mathcal{RHG}$ graphs on average. For $\mathcal{ROAD}$, each round only takes $1\mu s$ on average.

5.2 Output Metrics

Apart from the runtime per round, the number of total rounds to reach an approximately uniform distribution is of interest. In practice, the *mixing time* of the MCMC is often investigated by considering a collection of metrics which are regarded as proxies for the mixing state of the underlying chain.

We consider two metrics as proxies: (i) the average weight of the graph and (ii) the fraction of edges that have negative weights. We again consider $\mathcal{GNP}$ and $\mathcal{RHG}$ graphs with $n = 10000$ and $\overline{d} \in \{10, 20, 50\}$ and run the chain for $2.1 \cdot 10^8$ (10^8 for $\mathcal{ROAD}$) rounds.

Average Weight. We first consider the average total weight of all edges over time. Figure 2 shows that the average weight stabilizes after at most $2 \cdot 10^6$ steps for both graph classes. There is also a slight bias towards a higher average weight if the average degree is higher. This is to be expected as a higher average degree equates to a higher number of cycles and thus more restrictions on negative weights.

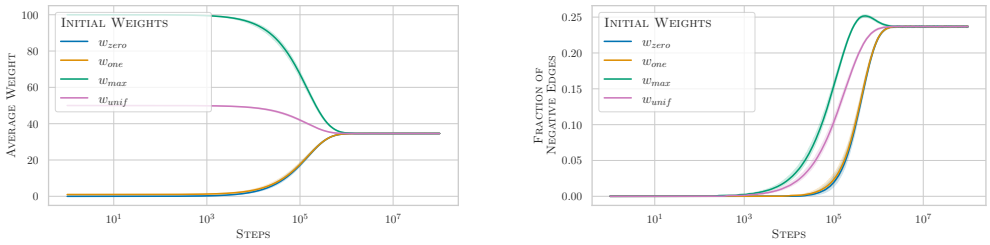

Figure 3 Average weight (l) and fraction of negative edges (r) over time on $\mathcal{ROAD}$ networks.

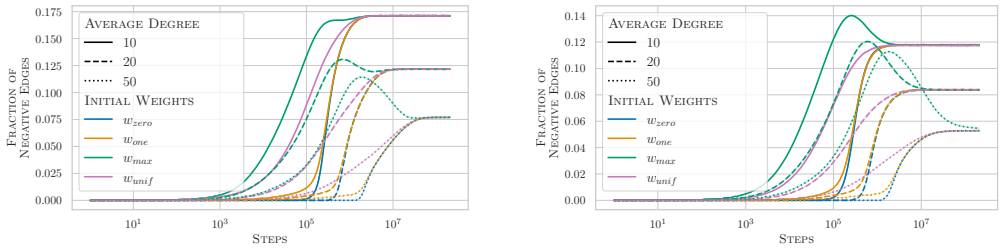

Figure 4 Fraction of negative edges over time for $\mathcal{GNP}$ (l), $\mathcal{RHG}$ (r).

Fraction of Negative Edges. We also consider the fraction of negative edges over time. Figure 4 summarizes the results. Similarly to the average weight, we observe both a bias towards less negative edges for bigger degrees as well as a stabilizing effect – although now after $10^6 \sim 10^8$ rounds, depending on average degree and graph type. Interestingly, for starting weights $w_{\max}$, the fraction of negative edges initially "overshoots" before settling down again into a stable state. This effect can be explained in part by the residual high-weight edges that have not yet been chosen as the target of an update. The chains require $\Theta(m \log m)$ rounds to ensure that each edge is selected at least once with high probability. Since for all considered $\mathcal{GNP}$ and $\mathcal{RHG}$ instances we have $10^6 \leq m \log m \leq 10^7$ it follows that, while these high-weight edges persist, they permit a significantly larger proportion of negative edges within any cycle containing them. Nonetheless, there still remains a lingering, slowly vanishing effect, well beyond the stated $m \log m$ bound. This effect seems to be driven by high-weight edges that persist well into the process and are prevented to significantly decrease their weight due to emerging constraints imposed by the, on average, decreasing cycle weight. This argument derives additional credence from two facts: (i) By Figures 2 and 3 the average weight stabilizes well before the fraction of negative edges and (ii) the expected edge weight in absence of constraints is 0 by the choice of $\mathcal{W}$, hence as soon as the average weight converges (bounded away from 0), constraints seem prevent a further decrease. The former suggests that there is still a significant amount of high-weight edges when average weight convergence is reached while the later indicates that these high-weight edges are unlikely to decrease their weight quickly due to the significant amount of constraints imposed by the (low-weight) cycles at this point. Despite this, in all observed cases, the chain stabilizes in $o(m^2)$ rounds.

5.3 Fast convergence on the n-cycle

Here, we aim to provide empirical evidence that, in practice, comparatively low mixing times suffice for the n-cycle and $\mathcal{W} = \{-1, 0, 1\}$.

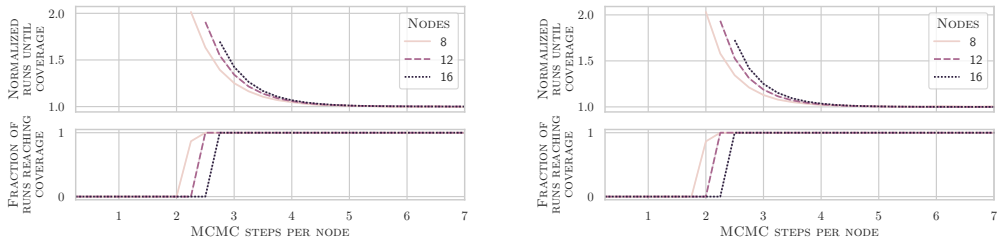

Figure 5 Runs to see 99% of $\mathbb{S}_{C_n,\mathcal{W}}$ on n-cycle for $\mathcal{W} = \{-1, 0, 1\}$ as function of MCMC steps for RejSampler (l) and NoRejSampler (r).

Methodology. The experimental design is inspired by the Coupon Collector's problem. We obtain a large number k of independent samples by running τ steps of the Markov Chain each starting from the initial weight $(0, \ldots, 0)$. Since each run is independent, some obtained weights are inevitably duplicates. However, we know from the Coupon collector's problem that for large n, $k = \Theta(|\mathbb{S}_{C_n,\mathcal{W}}| \log(|\mathbb{S}_{C_n,\mathcal{W}}|))$ samples are expected to be sufficient to observe every item in $\mathbb{S}_{C_n,\mathcal{W}}$ at least once. Since the last few elements incur significant noise, we stop earlier, namely when 99% of elements were seen.[6] We call this quantity *runs-until-coverage*.

Measurements. We use an exact sampling algorithm[7] to establish the correct baseline of $(4.6 \pm 0.1)|\mathbb{S}_{C_n,\mathcal{W}}|$ runs-until-coverage for $n \in \{8, 12, 16\}$. This linear dependency is plausible since each sample is a new item with probability of at least $1/100$. Then, we replace the exact sampler with the MCMC process for different numbers of steps τ and measure their runs-until-coverage. This allows us to reject the hypothesis that the MCMC process is identical to the exact sampler if τ is too small. If coverage does not occur within $10|\mathbb{S}_{C_n,\mathcal{W}}|$ steps (i. e., more than twice the baseline), we say that the run does not reach coverage. Figure 5 contains the findings as function of τ and normalized by the exact sampler's runs-until-coverage.

Between $2n \leq \tau \leq 3n$ steps, we observe a sharp transition from no runs achieving coverage, to all runs completing successfully for both chains. Observe that NoRejSampler reaches coverage slightly earlier than RejSampler since the MCMC steps of RejSampler contains both successes and rejections while NoRejSampler by design has only successful rounds. This provides strong evidence, at least on the n-cycle, that RejSampler and NoRejSampler display a similar convergence rate. Comparing $n = 8$ and $n = 16$, we see a slight increase in runs-until-coverage that is consistent with a mixing time of $\Theta(n \log n)$.

5.4 Identical Convergence

The previous experiment is only feasible for very small graphs and thus not practical for our main datasets. Thus, we want to provide further evidence that both RejSampler and NoRejSampler (i) converge relatively fast and (ii) converge at the same rate when only counting accepting rounds in RejSampler. Hence, we run statistical tests on meta-distributions of weights over time and show that convergence behaviour is practically identical for RejSampler and NoRejSampler.

[6] We also ensure that all samplers can *fully* cover $\mathbb{S}_{C_n,\mathcal{W}}$ to give additional credence to the correctness.
[7] We draw each edge weight $w(e)$ i.i.d. from $\mathcal{W} = \{-1, 0, 1\}$, and restart if $\sum_e w(e)$ is negative. This yields a constant acceptance probability on the n-cycle.

Weights. We restrict ourselves to $\mathcal{GNP}$ and $\mathcal{RHG}$ graphs with $n = 100$ and average degrees $\overline{d} \in \{10, 20\}$. We then simulate both RejSampler (BiDijkstra) and NoRejSampler (Resamp) on each graph $n \cdot \overline{d} \cdot 10$ ($\in \{10\,000, 20\,000\}$) times per possible starting weight $(w_{\max}, w_{\text{zero}}, w_{\text{one}}, w_{\text{unif}})$. After 2^i **accepting** rounds in each chain, we take a snapshot of the entire current weight vector of the graph.

Analysis. For each type of graph and time step s, we have up to 40 000/80 000 weight distributions of length 1000/2000 and thus a distribution over distributions. To compare two such distributions using common statistical tests, we need to find a representative value for each weight vector yielding a distribution over representatives. Natural examples are minimum, maximum, mean, and taking the weight of a specific edge (index).

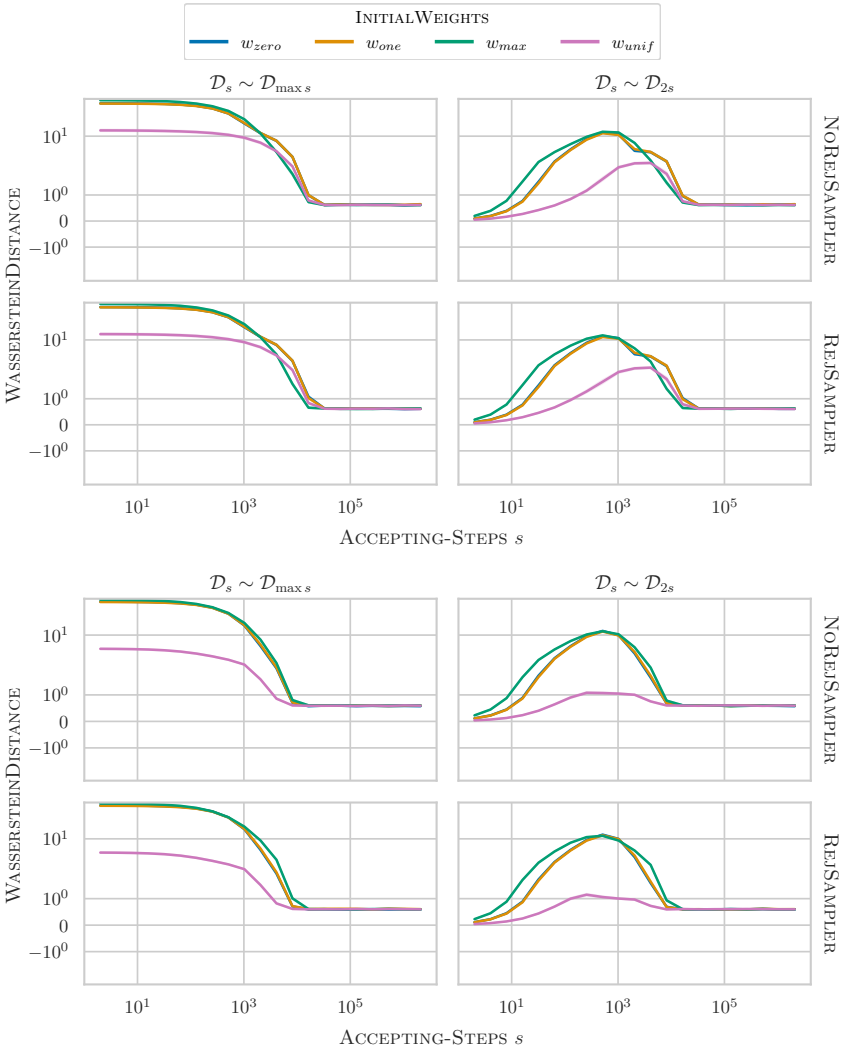

Figure 6 Wasserstein-Distances for weight-distributions of $\mathcal{GNP}(n = 100, \overline{d} = 10)$ (top) and $\mathcal{RHG}(n = 100, \overline{d} = 10)$ (bottom) graphs. Weights are categorized by sampler and initial weight function; meta-distributions are then constructed for each edge (index), evaluated, and then aggregated in these plots.

We then compare two meta-distributions using statistical tests to determine how close they are. We do this in two ways: (i) comparing distributions at step s with distributions at step $2s$, and (ii) comparing distributions at step s with distributions at step s_{max} where s_{max} is the highest recorded step. (i) determines how many changes occur between s accepting rounds, whereas (ii) determines how close a distribution at step s is to a final distribution at step s_{max}, which we assume to be a fairly good approximation of a uniform distribution.

Results. Figure 6 shows the results of a statistical analysis using the Wasserstein distance [24] for graphs with average degree $\overline{d} = 10$ (see Appendix for $\overline{d} = 20$: Figure 7). Although different types of graphs, we observe an almost indistinguishable behaviour in all cases. For the comparison of s and s_{max} (left), meta-distributions differ entirely from the final target distribution before converging around 10^4 accepting steps. On the other hand, when comparing distributions at s and $2s$ steps (right), we see a clear trend where initially similar distributions first stray away from each other before converging again at 10^4 steps.

In both cases, distributions start converging after 10^3 steps, coinciding with the number of edges. After circa 10^4 steps, the process enters a stable state. While this is not a full measure of the converged mixing time of the chains, it does show that the two chains, in fact, converge at a similar rate when measured on a metric of accepting steps. Paired with previous results, this reaffirms that (i) the chain converges relatively fast in practice and that (ii) Resamp is the most efficient algorithm to sample uniform negative edge weights.

6 Summary and Future Work

We introduce a maximum entropy model for signed edge weights, propose an MCMC sampler, and show that it converges to the model's distribution. We then engineer an implementation for the MCMC using a bidirectional search and a novel resampling technique. We empirically study the model, the sampler's convergence, as well as trade-offs in the implementation to provide an optimized, ready-to-use generator for uniform negative edge weights.

Future work might entail better theoretical bounds on the mixing time of our chains. The upper bound for the n-cycle, while polynomial, is highly impractical and not in line with experimentally verified bounds in Section 5.3.

It might also be interesting to further investigate the impact of asymmetric weight intervals (around 0); while initial experiments indicated no significant impact on running time, convergence, or even some output metrics, a more rigorous evaluation is needed.

Finally, it might also be interesting to study how recent advances in negative weight shortest path algorithms fare on weights generated by our maximum entropy model.

References

1 N. A. Arafat, D. Basu, L. Decreusefond, and S. Bressan. Construction and random generation of hypergraphs with prescribed degree and dimension sequences. In *DEXA (2)*, volume 12392 of *LNCS*, pages 130–145. Springer, 2020. doi:10.1007/978-3-030-59051-2_9.

2 A.-L. Barabasi. *Network Science*. Cambridge University Press, Cambridge, England, July 2016.

3 R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958. URL: http://www.jstor.org/stable/43634538.

4 A. Bernstein, D. Nanongkai, and C. Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *FOCS*, pages 600–611. IEEE, 2022. doi:10.1109/FOCS54457.2022.00063.

5 T. Bläsius, C. Freiberger, T. Friedrich, M. Katzmann, F. Montenegro-Retana, and M. Thieffry. Efficient shortest paths in scale-free networks with underlying hyperbolic geometry. *ACM Trans. Algorithms*, 18(2):19:1–19:32, 2022. doi:10.1145/3516483.

6 T. Bläsius and M. Wilhelm. Deterministic performance guarantees for bidirectional BFS on real-world networks. In *IWOCA*, volume 13889 of *LNCS*, pages 99–110. Springer, 2023. doi:10.1007/978-3-031-34347-6_9.

7 M. Borassi and E. Natale. KADABRA is an adaptive algorithm for betweenness via random approximation. *ACM J. Exp. Algorithmics*, 24(1):1.2:1–1.2:35, 2019. doi:10.1145/3284359.

8 C. J. Carstens. *Topology of Complex Networks: Models and Analysis*. PhD thesis, RMIT University, 2016.

9 A. Cassis, A. Karrenbauer, A. Nusser, and P. L. Rinaldi. Algorithm engineering of SSSP with negative edge weights. In *SEA*, LIPIcs, pages 10:1–10:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.SEA.2025.10.

10 B. V. Cherkassky and A. V. Goldberg. Negative-cycle detection algorithms. *Math. Program.*, 85(2):277–311, 1999. doi:10.1007/S101070050058.

11 M. Costanzo, E. Rucci, M. R. Naiouf, and A. D. Giusti. Performance vs programming effort between rust and C on multicore architectures: Case study in n-body. In *CLEI*, pages 1–10. IEEE, 2021. doi:10.1109/CLEI53233.2021.9640225.

12 A. De Martino and D. De Martino. An introduction to the maximum entropy approach and its application to inference problems in biology. *Heliyon*, 4(4):e00596, 2018. doi:10.1016/j.heliyon.2018.e00596.

13 C. Demetrescu, D. Frigioni, A. M. Spaccamela, and U. Nanni. Maintaining shortest paths in digraphs with arbitrary arc weights: An experimental study. In *WAE*, volume 1982 of *LNCS*, pages 218–229. Springer, 2000. doi:10.1007/3-540-44691-5_19.

14 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.

15 M. Drobyshevskiy and D. Turdakov. Random graph modeling: A survey of the concepts. *ACM Comput. Surv.*, 52(6):131:1–131:36, 2020. doi:10.1145/3369782.

16 J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, April 1972. doi:10.1145/321694.321699.

17 M. G. Everett and S. P. Borgatti. Networks containing negative ties. *Soc. Networks*, 38:111–120, 2014. doi:10.1016/J.SOCNET.2014.03.005.

18 A. M. Frieze and G. R. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discret. Appl. Math.*, 10(1):57–77, 1985. doi:10.1016/0166-218X(85)90059-9.

19 E. N. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4), 1959. doi:10.1214/aoms/1177706098.

20 C. Gkantsidis, M. Mihail, and E. W. Zegura. The markov chain simulation method for generating connected power law random graphs. In *ALENEX*, pages 16–25. SIAM, 2003.

21 N. J. Gotelli and G. R. Graves. *Null models in ecology*. Smithsonian Institution, 1996.

22 D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.

23 L. R. F. Jr. *Network Flow Theory*. RAND Corporation, Santa Monica, CA, 1956.

24 L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960. URL: http://www.jstor.org/stable/2627082.

25 R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

26 M. Kaur and S. Singh. Analyzing negative ties in social networks: A survey. *Egyptian Informatics J.*, 17(1):21–43, 2016. doi:10.1016/j.eij.2015.08.002.

27 D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, September 2010. doi:10.1103/PhysRevE.82.036106.

28 R. Milo. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594), 2002. doi:10.1126/science.298.5594.824.

29 M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

30 J. Moos, K. Hansel, H. Abdulsamad, S. Stark, D. Clever, and J. Peters. Robust reinforcement learning: A review of foundations and recent advances. *Mach. Learn. Knowl. Extr.*, 4(1):276–315, 2022. doi:10.3390/MAKE4010013.

31 M. E. O'Neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, September 2014.

32 T. P. Peixoto. Model selection and hypothesis testing for large-scale network models with overlapping groups. *Phys. Rev. X*, 5(1), 2015. doi:10.1103/physrevx.5.011033.

33 A. R. Rao, R. Jana, and S. Bandyopadhyay. A markov chain monte carlo method for generating random (0, 1)-matrices with given marginals. *Sankhyā: The Indian J. Statistics, Series A*, 1996.

34 J. M. Smith, D. S. Halgin, V. Kidwell-Lopez, G. J. Labianca, D. J. Brass, and S. P. Borgatti. Power in politically charged networks. *Soc. Networks*, 36:162–176, 2014. doi:10.1016/J.SOCNET.2013.04.007.

35 I. Stanton and A. Pinar. Sampling graphs with a prescribed joint degree distribution using markov chains. In *ALENEX*, pages 151–163. SIAM, 2011. doi:10.1137/1.9781611972917.15.

36 G. Strona, D. Nappo, F. Boccacci, S. Fattorini, and J. Miguel-Ayanz. A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. *Nature Commun.*, 2014. doi:10.1038/ncomms5114.

37 R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

38 W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. of the IEEE*, 55(11):1801–1809, 1967.

39 R. van der Hofstad. *Random Graphs and Complex Networks*, volume 43 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, 2016. doi:10.1017/9781316779422.

40 F. Viger and M. Latapy. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. *J. Complex Networks*, 4(1):15–37, 2016. doi:10.1093/COMNET/CNV013.

A Additional experimental results

Table 3 Number of possible $|\mathcal{W}^n|$ and consistent $|\mathbb{S}_{C_n,\mathcal{W}}|$ weights on the n-cycle.

n	$\lvert\mathcal{W}^n\rvert$	$\lvert\mathbb{S}_{C_n,\mathcal{W}}\rvert$	Acc. rate
8	6561	3834	0.584
12	531 441	302 615	0.569
16	43 046 721	24 121 674	0.560

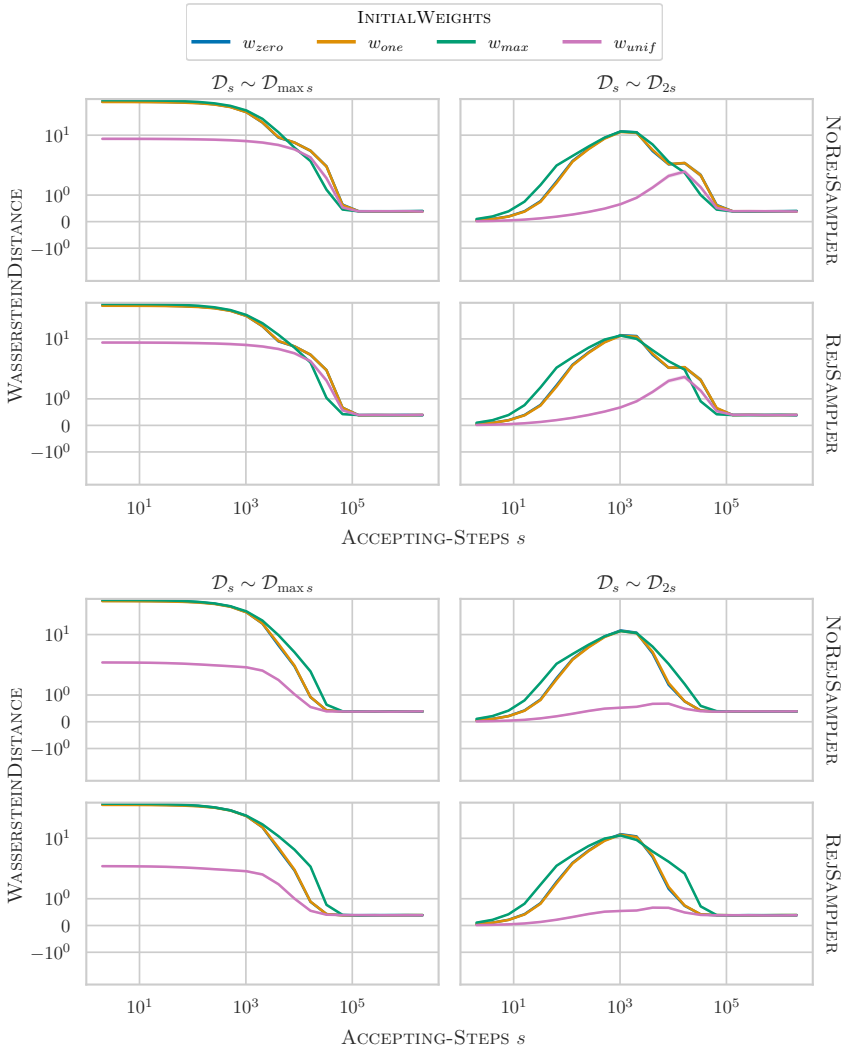

Figure 7 Wasserstein-Distances for weight-distributions of $\mathcal{GNP}(n = 100, \overline{d} = 20)$ (up) and $\mathcal{RHG}(n = 100, \overline{d} = 20)$ (down) graphs. Weights are categorized by sampler and initial weight function; meta-distributions are then constructed for each edge (index), evaluated, and then aggregated in these plots.