Integrating and Sampling Cuts in Bounded Treewidth Graphs

Ivona Bezáková[1*], Erin W. Chambers[2†], and Kyle Fox[3‡]

Keywords: treewidth, minimum cuts, algorithms

AMS Subject Classifications: 05C83, 05C85, 97P20

Abstract

In this paper, we consider the problem of evaluating (s,t)-cuts in a bounded treewidth graph. In particular, we show how to compute the partition function for weighted cuts of the graph, i.e., the total weight of all (s,t)-cuts where the weight of a single cut is the product of its edge weights. This method can also easily be adapted to work with additive weights for the cost of a cut. We also present a method for sampling a cut proportional to its weight in linear time. Computing the partition function is #P-hard for general graphs, and our sampling algorithm is simple enough to prove useful is several application areas. Finally, we discuss an alternative method for sampling cuts that uses Markov chains and show that, in the worst case, its mix-

Department of Computer Science, Rochester Institute of Technology
Rochester, NY 14623, USA
ib@cs.rit.edu
· Department of Computer Science and Mathematics, Saint Louis University
St. Louis, MO 63103, USA
echambe5@slu.edu
· Department of Computer Science, Duke University
Durham, NC 27708, USA
kylefox@cs.duke.ed

* Work by this author was partially supported by the National Science Foundation under Award No. CCF-1319987. Portions of this research took place while the author visited the Simons Institute for the Theory of Computing.

† Work by this author was partially supported by the National Science Foundation under Grants No. CCF 1054779 and IIS-1319573.

‡ Work by this author was partially supported by the Stutzke Dissertation Completion Fellowship from the University of Illinois at Urbana-Champaign.

ing time is exponential in the size of the graph even when the graph has bounded treewidth.

1 Introduction

Given a directed graph $G = (V, E)$ of bounded treewidth with edge weights $w(\cdot)$ and two designated vertices s and t, we describe a linear time algorithm for computing the partition function for weighted cuts in G. We assume a model of computation where pairs of values can be added, multiplied, and compared in constant time; requiring time proportional to the bit-complexity of numbers only increases our running times by a near-linear factor. After running our algorithm, it becomes possible to sample cuts proportionally to the product of edge weights in linear time. We emphasize that the probability of a cut being chosen in an individual sample is *exactly* as described above and not approximate.

For an (s,t)-cut $C \subseteq V$, $s \in C$, $t \notin C$, its weight $w(C)$ is the *product* of weights for edges crossing the cut, that is:

$$w(C) = \prod_{(x,y):x\in C, y\notin C, (x,y)\in E} w(x,y).$$

Weights corresponding to products are used in many applications. For example, in physics and biology, the weights often represent probabilities or energies of individual events. In practice, the actual probabilities are unknown and are replaced by energies that are proportional to the probabilities. However, for certain applications including the maximum likelihood principle, one needs a close estimate on the probability of an event, not just its energy. To compute this probability, one needs to scale the energy of the event by the sum of the energies of all events. This scaling quantity is known as the *partition function*. Our algorithms for computing the partition function and sampling cuts appear in Sects. 3 and 4 respectively.

In Sect. 5, we extend the above results to work with multiple sources and sinks. The extension is surprisingly simple; however, this is the first result we are aware of on evaluating cuts in the multiple sources and sinks setting. In Sect. 6, we describe how to modify the above techniques to count and sample (s,t)-cuts that have minimum summed edge weight. While the minimum weight (s,t)-cut counting procedure is nearly the same as that given for computing the partition function, we feel it is different enough to be of independent interest.

As further motivation for our approach, we conclude this report with a discussion on Markov chain methods for (approximately) sampling (s,t)-cuts by multiplicative weight (Sect. 7). Markov chains have been used successfully for a variety of difficult counting problems, either to approximately count the number of solutions or to provide an approximately uniformly random solution [13]. A common and natural approach is to use a Glauber dynamics type chain that in each transition modifies the current state by a constant number of sites. For cuts, this means adding or removing

a constant number of vertices to/from the current cut. We show that this approach may need exponential time to converge to an approximately uniform distribution on cuts, even in graphs of treewidth 2 or less. Therefore, for essentially any graph class Glauber dynamics Markov chains will not yield polynomial-time approximations of the partition function for weighted (s,t)-cuts. In contrast, our algorithms handle an interesting and widely studied class of graphs in (fast) polynomial time, and provide motivation for considering extensions to more general classes of graphs such as minor free graph families.

1.1 Related work

For (s,t)-cuts the partition function Z is defined as the sum of the weights of all (s,t)-cuts:

$$Z := \sum_{C:C\subseteq V,s\in C,t\notin C} w(C).$$

Notice that for a cut C, its actual probability when sampling proportionally to its weight is $w(C)/Z$. Computing the partition function is #P-hard. This fact can be shown via a reduction from the problem of counting minimum cardinality (s,t)-cuts, which is #P-complete [17]. Assigning weight $1/2^n$ to all edges in the graph, where n is the number of vertices, means the minimum cardinality cuts will dominate the partition function.

This problem of counting minimum weight (s,t)-cuts in general graphs is #P-complete [17] even for unit weights, and can be reduced to the problem of counting maximal antichains in a poset [2]. Ball and Provan first considered the problem of counting minimum cuts and gave a polynomial time algorithm to compute the number of minimum cardinality (s,t)-cuts in an (s,t)-planar graph (where the source and sink are on the same face) [2]. Later, Bezáková and Friedlander generalized the algorithm for arbitrary locations of s and t in a planar graph [3] while also allowing arbitrary edge capacities. Chambers, Fox, and Nayyeri further generalized the algorithm for directed graphs embedded on orientable surfaces of bounded genus [7].

The problem of counting minimum (cardinality) cuts was originally motivated by questions in network reliability [2, 8, 14, 16]. In particular, the problem is closely related to the probabilistic connectness of stochastic graphs, where edges may fail with known probabilities [2]. More recently, counting minimum cuts has been studied for its applications to problems in computer vision. In these applications, the pixels of an image are interpreted as vertices in a graph with edges between the vertices describing the similarity between pixels. Minimum cuts provide a high quality way to segment the pixels of the image [6]. Counting minimum cuts is closely related to sampling these cuts, allowing for a varied selection of high quality segmentations.

1.2 Courcelle's theorem for bounded treewidth graphs

Courcelle showed in 1990 that any graph property describable in counting monadic second order logic can be decided in linear time if the input graph has bounded treewidth [9]. There are often practicality concerns with using this particular meta-theorem, however, since many direct applications of it lead to hidden constants that are doubly or triply exponential in the treewidth [11].

Courcelle's theorem has been extended in a variety of ways. One extension particularly relevant to our work is as follows. Weight the edges of a graph G and fix a monadic second order logic formula ϕ with one free set variable. For any set of edges A, the weight of A can be defined as either the sum or product of its edge weights. If G has bounded treewidth, it is possible to sum the weight of all sets A that satisfy ϕ in linear time [10]. In particular, this result implies we can compute the partition function for weighted cuts of a graph G in linear time in bounded treewidth graphs, one of our main results.

The main advantage of our partition function algorithm over the meta-theorems mentioned above is that our algorithm is very simple, and the dependence on the treewidth of G is only singly exponential. As stated earlier, standard applications of Courcelle's theorem often have doubly or triply exponential dependence on treewidth.

Perhaps of greater interest, our algorithm also provides a simple way to randomly sample cuts, one of the key motivations behind the study of partition functions; to the best of our knowledge, no such sampling is known under the general framework of Courcelle's theorem.

2 Tree Decompositions and Treewidth

A *tree decomposition* $\mathscr{T}$ of a graph $G = (V,E)$ is a pair $(T, \mathscr{X})$ where T is a tree and $\mathscr{X}$ is a family of subsets (or *bags*) of V such that:

- Each node u of T has a corresponding subset $X_u \in \mathscr{X}$ and $\cup_{X\in\mathscr{X}} X = V$;
- For every edge $uv \in E$ there is a bag $X \in \mathscr{X}$ such that $u, v \in X$.
- For any three nodes $u, v, w \in T$ such that v is on the u-to-w path in T, $X_u \cap X_w \subseteq X_v$.

In this paper, we will refer to the *bags or nodes* of T and the *vertices* of G to avoid confusion.

The *width* of a tree decomposition $(T, \mathscr{X})$ is $\max_{X\in\mathscr{X}} |X| - 1$. The *treewidth* of a graph is the minimum possible width of a tree decomposition of the graph. Any graph of treewidth k has a tree decomposition with at most $n - k + 1$ nodes [5].

Tree decompositions were originally introduced by Halin [12] and were rediscovered (and popularized) by Robertson and Seymour [18]. While it is NP-complete to decide if any graph has treewidth at most k [1], a tree decomposition can be constructed in linear time if k is a constant (the dependance on k is exponential) [4].

We only consider tree decompositions $\mathscr{T} = (T, \mathscr{X})$ with $O(n)$ nodes where T is rooted at some node r and every node of T has at most two children. We can modify any tree decomposition of width k and $O(n)$ nodes to meet the last assumption in $O(kn)$ time without increasing the width of the decomposition by replacing any node v with d children by $d-1$ nodes, each with 2 children and the same bag as v.

3 Computing the Partition Function

In this section we show how to compute the partition function for weighted cuts of G. Let $G = (V, E, w)$ be a positively edge-weighted graph (directed or not) with treewidth k. Let $s, t \in V$ be the source and the sink, respectively, $s \neq t$. For an (s,t)-cut $C \subseteq V$, $s \in C$, $t \notin C$, its weight $w(C)$ is the *product* of weights for edges crossing the cut, that is:

$$w(C) = \prod_{(x,y):x\in C, y\notin C, (x,y)\in E} w(x,y).$$

We wish to compute the total weight, that is, the sum of weights over all (s,t)-cuts.

Let $\mathscr{T}$ be a tree decomposition of G with width k. For every edge $e = (x,y) \in E$, choose exactly one bag X_u with $x, y \in X_u$ as its *designated bag*. For a bag X_u, we refer to the set E_u of edges $e = (x,y) \in E$ such that $x, y \in X_u$ and X_u is the designated bag for e, as the *designated edges* for X_u.

We are now ready to present the algorithm. The idea is to compute, for each bag and for each of its subsets, the total weight of the cuts that are consistent with the subset, where the weight takes into account only the edges that are designated to this bag or to one of its descendents. Algorithm 1 contains pseudocode for the algorithm. The correctness of the algorithm is chiefly explained by Lemma 1 below.

We first analyze the running time of the algorithm. It does a single pass through the tree T, where in each node it goes through $2^{O(k)}$ operations. There are $O(n)$ nodes total in T, so we get an $2^{O(k)}n$ running time. More precisely, each node actually has $O(2^{3k}k)$ operations since there are 2^{k+1} choices for C, 2^{2k+2} choices for C_1 and C_2, and it takes $O(k)$ time to verify that for each pair C_1 and C_2, we have $C_1 \cap X_v = C \cap X_{v_1}$ and $C_2 \cap X_v = C \cap X_{v_2}$. We get a total running time of $O(2^{3k}kn)$.

Next we prove the correctness of our algorithm.

Lemma 1. *Let v be a node of T and let $P(v)$ be the set of all descendants of v in T (including v). Let $\tilde{V}_v = \cup_{u\in P(v)} X_u$ be the union of all the bags corresponding to $P(v)$, and let $\tilde{E}_v = \cup_{u\in P(v)} E_u$ be the union of all the edges that are designated for those bags. At the time when a node v is marked as done, the following holds for each $C \subseteq X_v$:*

$$weightDP_v[C] = \sum_{\tilde{C}\subseteq\tilde{V}_v, s\notin(\tilde{V}_v\setminus\tilde{C}), t\notin\tilde{C}, \tilde{C}\cap X_v = C} \left(\prod_{x,y:x\in\tilde{C}, y\notin\tilde{C}, (x,y)\in\tilde{E}_v} w(x,y) \right).$$

Algorithm 1 Computing the partition function for weighted (s,t)-cuts in a graph G with treewidth k

Require: A graph G, a corresponding tree decomposition $\mathscr{T} = (T, \mathscr{X})$ of width k with T rooted at node r, and distinct vertices s,t

1: Mark all nodes of T as not done.
2: **for** every leaf node u of T **do**
3: **for** every subset $C \subseteq X_u$ such that $s \notin (X_u \setminus C)$ and $t \notin C$ **do**
4: Let $\text{weightDP}_u[C]$ be the product of all the designated edges for X_u that are cut by C: $\text{weightDP}_u[C] := \prod_{(x,y)\in E_u, x\in C, y\notin C} w(x,y)$.
5: **for** all other subsets $C \subseteq X_u$ **do**
6: Let $\text{weightDP}_u[C] := 0$.
7: Mark u as done.
8: **for** every non-leaf node v of T such that all its children are done **do**
9: Let v_1 be v's child and v_2 be the other child (if it exists).
10: **for** every subset $C \subseteq X_v$ such that $s \notin (X_v \setminus C)$ and $t \notin C$ **do**
11: Let $\text{weightDP}_v[C] := 0$.
12: **for** every subset $C_1 \subseteq X_{v_1}$ such that $C_1 \cap X_v = C \cap X_{v_1}$ and every subset $C_2 \subseteq X_{v_2}$ such that $C_2 \cap X_v = C \cap X_{v_2}$ (if applicable) **do**
13: Add $\text{weightDP}_{v_1}[C_1] \cdot \text{weightDP}_{v_2}[C_2]$ to $\text{weightDP}_v[C]$. (If there is no v_2, take $\text{weightDP}_{v_2}[C_2] = 1$.)
14: Set $\text{weightDP}_v[C] := \text{weightDP}_v[C] \cdot \prod_{(x,y)\in E_v, x\in C, y\notin C} w(x,y)$.
15: **for** all other subsets $C \subseteq X_v$ **do**
16: Let $\text{weightDP}_v[C] := 0$.
17: Mark v as done.
18: **return** $\sum_{C\subseteq X_r} \text{weightDP}_r[C]$.

Proof. We will prove the lemma by induction on the number of descendants of v. For the base case, v is a leaf node of T. The summation goes through a single $\tilde{C}$ as $\tilde{v} = X_v$ and $\tilde{C} \cap X_v = C$ implies that $\tilde{C} = C$. The claim follows directly from steps 4 and 6 of the algorithm.

For the inductive case, suppose that v is not a leaf and that the claim holds for all nodes with fewer descendants; in particular, the claim holds for v's children. Let $\tilde{C} \subseteq \tilde{V}_v$ be such that $s \notin (\tilde{V}_v \setminus \tilde{C})$, $t \notin \tilde{C}$, and $\tilde{C} \cap X_v = C$. We will show that the algorithm includes the weight $\prod_{x,y:x\in\tilde{C}, y\notin\tilde{C}, (x,y)\in\tilde{E}_v} w(x,y)$ in the computation of $\text{weightDP}_v[C]$.

Recall that v_1 and v_2 (if it exists) are v's children in T. Let $\tilde{C}_i = \tilde{C} \cap \tilde{V}_{v_i}$ be the restriction of the cut $\tilde{C}$ to the descendants of v_i and let $C_i = \tilde{C} \cap X_{v_i}$ be its restriction to the bag X_{v_i}. Notice that $C_i \cap X_v = C \cap X_{v_i}$. Let $\pi(C) = \prod_{(x,y)\in E_v, x\in C, y\notin C} w(x,y)$. The sum of weights for all $\tilde{C}$ as described above is

$$\sum_{\tilde{C}} \left[\pi(C) \cdot \prod_{(x,y)\in\tilde{E}_{v_1}, x\in\tilde{C}_1, y\notin\tilde{C}_1} w(x,y) \cdot \prod_{(x,y)\in\tilde{E}_{v_2}, x\in\tilde{C}_2, y\notin\tilde{C}_2} w(x,y) \right]$$
$$= \pi(C) \sum_{\tilde{C}} \left[\prod_{(x,y)\in\tilde{E}_{v_1}, x\in\tilde{C}_1, y\notin\tilde{C}_1} w(x,y) \cdot \prod_{(x,y)\in\tilde{E}_{v_2}, x\in\tilde{C}_2, y\notin\tilde{C}_2} w(x,y) \right].$$

By the definition of tree decompositions, any pair of choices for $\tilde{C_1}$ and $\tilde{C_2}$ as allowed above will include or exclude the same members of $\tilde{V}$. By this fact and distribution, we see the expression above is equal to

$$\pi(C) \cdot \text{weightDP}_v[C_1] \cdot \text{weightDP}_v[C_2].$$

The lemma immediately yields the main theorem of this section.

Theorem 1. *Algorithm 1 correctly computes the partition function for all (s,t)-cuts. The running time is $O(2^{3k}kn)$.*

Proof. Apply the lemma to $v = r$. Then we get $\tilde{V}_r = V$, $\tilde{E}_r = E$. An (s,t)-cut $\tilde{C}$ will be accounted for by $\text{weightDP}_r[C]$ for $C = \tilde{C} \cap X_r$. Summing across all C's we account for all (s,t)-cuts.

4 Sampling Cuts

One interesting application of our dynamic programming formulation is that it can be easily modified to aid in repeatedly sampling cuts. Recall that in many applications, these weights are probabilities of some individual event's occurrence, and may be estimates or energies that correspond to probabilities but lack the scaling factor. Therefore, sampling provides a method via which events can be repeatedly selected from the given probability distribution.

Algorithm 1 given above builds a dynamic programming table that can be used to randomly sample cuts proportionally to their weight. For a node v, the subset of vertices $\tilde{V}_v$ as described in Lemma 1, and a subset of vertices $C \subseteq X_v$, we need a way to sample a subset of vertices $\tilde{C} \subseteq \tilde{V}_v$ such that $s \notin (\tilde{V}_v \setminus \tilde{C})$, $t \notin \tilde{C}$, and $\tilde{C} \cap X_v = C$ proportionally to $\prod_{x,y:x\in\tilde{C},y\notin\tilde{C},(x,y)\in\tilde{E}_v} w(x,y)$. Let v_1 and v_2 be the children of v in T (assuming v_2 exists). In order to sample this subset, our algorithm randomly selects two subsets of vertices $C_1 \subseteq X_{v_1}$ and $C_2 \subseteq X_{v_2}$ such that $C_1 \cap X_v = C \cap X_{v_1}$ and $C_2 \cap X_v = C \cap X_{v_2}$. It does so proportionally to $\text{weightDP}_{v_1}[C_1] \cdot \text{weightDP}_{v_2}[C_2]$. Similar to before, if v_2 does not exist, then our algorithm only selects C_1 and it assumes $\text{weightDP}_{v_2}[C_2] = 1$ when giving weights to the subsets. Once C_1 and C_2 are selected in $O(k)$ time, it recursively selects subsets from $\tilde{V}_{v_1}$ and $\tilde{V}_{v_2}$ using the same procedure.

In order to select an (s,t)-cut, it selects a set $C_r \subseteq X_r$ from the root bag proportionally to $\text{weightDP}_r[C_r]$ in $O(k)$ time. It then uses the above procedure to select the whole cut $\tilde{C}$. A random sample is performed once per tree node, so the entire procedure takes $O(kn)$ time.

Theorem 2. *There exists an algorithm to sample (s,t)-cuts proportionally to their weight in $O(kn)$ time per sample after running Algorithm 1 once.*

5 Multiple Source-Sink Cuts

We now describe an extension to our previous algorithms to handle computing the partition function when there are multiple sources and multiple sinks in the input graph G. In essence, this is a simple modification to our previous algorithms, but as far as we are aware it is the first result on evaluating cuts in the multiple sources and sinks setting.

The extension works as follows. Let S be the set of source vertices and T be the set of sink vertices. Our algorithms modify the graph G by adding two vertices s^* and t^*. They then add an edge from s^* to every member of S and an edge from every member of T to t^*. Vertex s^* is set as the only source and vertex t^* is set as the only sink.

These edges are given weight 0 so that any (s^*,t^*)-cut that divides S or T will have weight 0. We can add s^* and t^* to every bag in any tree decomposition of G, increasing the width of the decomposition by 2 while still maintaining it as a valid tree decomposition after modifying G. The partition function and number of minimum weight (S_1,S_2)-cuts can still be computed in $O(2^{3k}kn)$ time.

6 Minimum (s,t)-cuts

In this section, we describe how to count and sample (s,t)-cuts that have minimum summed edge weight. Our algorithm is very similar to the one used for computing the partition function. The pseudocode appears in Algorithm 2. The key idea behind our algorithm for counting minimum cuts is that our dynamic programming procedure takes a subset of vertices for a bag and returns *two* values, the *weight* of any minimum (s,t)-cut consistent with that subset, and the *number* of these minimum weight cuts. When computing the two values for a node v's bag, it enumerates all consistent subsets for the children of v. The children's subsets only contribute to v's number variable if the sum of their weight variables is minimum. The proof of correctness is nearly the same as that given earlier for computing the partition function.

Theorem 3. *Algorithm 2 correctly computes the number of minimum weight (s,t)-cuts. The running time is $O(2^{3k}kn)$.*

Similar to before, our algorithm for counting minimum (s,t)-cuts builds a dynamic programming table that can be used to sample minimum weight (s,t)-cuts uniformly at random. The procedure is the same as the one given for sampling cuts proportionally to multiplicative weight, except the sampling algorithm will pick subsets of vertices C_1 and C_2 for each node v's children proportionally to the product of C_1 and C_2's number variables. The algorithm only considers subsets C_1 and C_2 where the sum of their weight variables are minimum.

Theorem 4. *There exists an algorithm to sample minimum weight (s,t)-cuts uniformly at random in $O(kn)$ time per sample after running Algorithm 2 once.*

Algorithm 2 Counting the minimum weight (s,t)-cuts in a graph G with treewidth k

Require: A graph G, a corresponding tree decomposition $\mathscr{T} = (T, \mathscr{X})$ of width k with T rooted at node r, and distinct vertices s,t

Mark all nodes of T as not done.
for every leaf node u of T **do**
  for every subset $C \subseteq X_u$ such that $s \notin (X_u \setminus C)$ and $t \notin C$ **do**
    Let $\text{countDP}_u[C]$ be the total weight and number of cuts designated by C: $\text{countDP}_u[C] := \left(\sum_{(x,y)\in E_u, x\in C, y\notin C} w(x,y), 1\right)$.
  for all other subsets $C \subseteq X_u$ **do**
    Let $\text{countDP}_u[C] := (\infty, 0)$.
  Mark u as done.
for every non-leaf node v of T such that all its children are done **do**
  Let v_1 be v's child and v_2 be the other child (if it exists).
  for every subset $C \subseteq X_v$ such that $s \notin (X_v \setminus C)$ and $t \notin C$ **do**
    Let $(\text{minWeight}, \text{cutCount}) := (\infty, 0)$.
    for every subset $C_1 \subseteq X_{v_1}$ such that $C_1 \cap X_v = C \cap X_{v_1}$ and every subset $C_2 \subseteq X_{v_2}$ such that $C_2 \cap X_v = C \cap X_{v_2}$ (if applicable) **do**
      $(\text{subWeight}_i, \text{subCount}_i) := \text{countDP}_{v_i}[C_i]$ for $i \in \{1,2\}$.
      (If there is no v_2, take $\text{countDP}_{v_2}[C_2] = (0,1)$.)
      if $\text{subWeight}_1 + \text{subWeight}_2 = \text{minWeight}$ **then**
        Set $(\text{minWeight}, \text{cutCount}) := (\text{minWeight}, \text{cutCount} + \text{subCount}_1 \cdot \text{subCount}_2)$.
      if $\text{subWeight}_1 + \text{subWeight}_2 < \text{minWeight}$ **then**
        Set $(\text{minWeight}, \text{cutCount}) := (\text{subWeight}_1 + \text{subWeight}_2, \text{subCount}_1 \cdot \text{subCount}_2)$.
    Set $\text{countDP}_v[C] := (\text{minWeight} + \sum_{(x,y)\in E_u, x\in C, y\notin C} w(x,y), \text{cutCount})$.
  for all other subsets $C \subseteq X_v$ **do**
    Let $\text{countDP}_u[C] := (\infty, 0)$.
  Mark v as done.
Let $(\text{minWeight}, \text{cutCount}) := (\infty, 0)$.
for every subset $C \subseteq X_r$ **do**
  $(\text{subWeight}, \text{subCount}) := \text{countDP}_r[C]$.
  if $\text{subWeight} = \text{minWeight}$ **then**
    Set $(\text{minWeight}, \text{cutCount}) := (\text{minWeight}, \text{cutCount} + \text{subCount})$.
  if $\text{subWeight} < \text{minWeight}$ **then**
    Set $(\text{minWeight}, \text{cutCount}) := (\text{subWeight}, \text{subCount})$.
return cutCount.

7 Markov Chain Techniques: Slow Mixing

In this section we discuss using Markov chains to generate a random (s,t)-cut approximately proportional to its multiplicative weight. In particular, we provide a simple undirected graph with bounded treewidth for which Markov chains that modify only a constant portion of the cut need exponential time to get close to the stationary distribution. We begin with a refresher on Markov chains before getting into our results.

7.1 Markov chain preliminaries

A *Markov chain* is a pair (Ω, P), where Ω is a set of *states* and P is a (right) stochastic matrix of size $|\Omega| \times |\Omega|$ that specifies the probabilities $P(x,y)$ of *transitioning* from state x to state y. A distribution π on states Ω is *stationary* if $\pi(y) = \sum_{x\in\Omega} \pi(x)P(x,y)$ for all $x,y \in \Omega$; in other words, if starting from a state chosen according to the distribution π, after one step of the Markov chain the states are distributed according to π. Notice that $P^t(x,y)$ is the probability of transitioning from x to y in t steps. A Markov chain is irreducible if for every x,y there is a t such that $P^t(x,y) > 0$; it is aperiodic if $\gcd\{t : P^t(x,y) > 0\} = 1$ for every x,y. An irreducible and aperiodic Markov chain has a unique stationary distribution; moreover, if the transition matrix is symmetric (that is $P(x,y) = P(y,x)$ for every x,y), then the stationary distribution is uniform (that is $\pi(x) = 1/|\Omega|$). The Metropolis-Hastings technique can be used to modify the transition probabilities of a symmetric Markov chain to achieve a desired stationary distribution σ. In particular, for an irreducible, aperiodic, and symmetric Markov chain $M = (\Omega, P)$, we can construct a Markov chain $M_\sigma = (\Omega, P_\sigma)$ such that $P_\sigma(x,y) = P(x,y)\min\{\sigma(y)/\sigma(x), 1\}$ for $x \neq y$.

The *mixing time* $\tau(\varepsilon) := \max_{x\in\Omega} \min\{t : d_{tv}(P^t(x,.), \pi) < \varepsilon\}$ is the time needed to get ε-close to stationarity when starting from an arbitrary state x. The *total variation distance* $d_{tv}(\mu,\pi) := \sum_{x\in\Omega}(\mu(x) - \pi(x))/2$ measures the closeness of two distributions μ and π. For any $A \subset \Omega$, let $\pi(A) := \sum_{x\in A} \pi(x)$. A quantity known as *conductance*

$$\Phi := \min_{A\subset\Omega, \pi(A)\leq 1/2} \frac{\sum_{x\in A, y\in\Omega-A} \pi(x)P(x,y)}{\pi(A)} \tag{1}$$

can be used to bound the mixing time of an ergodic Markov chain (from above and below). In particular, for a Markov chain with $P(u,u) \geq 1/2$, for every $u \in \Omega$,

$$\frac{1}{2}\left(\frac{1}{2\Phi} - 1\right)\log\left(\frac{1}{2\varepsilon}\right) \leq \tau(\varepsilon) \leq \frac{2}{\Phi^2}\log\left(\frac{1}{\pi_{\min}\varepsilon}\right), \tag{2}$$

where $\pi_{\min} = \min_{x\in\Omega} \pi(x)$ [15, 19]. The requirement on $P(u,u) \geq 1/2$ is technical, typically used to guarantee that a chain is aperiodic. For every Markov chain $M = (\Omega, P)$ there exists a so-called *lazy* Markov chain $M_{\text{lazy}} = (\Omega, P_{\text{lazy}})$ that with probability $1/2$ stays in the current state, otherwise it follows transitions of M; formally, $P_{\text{lazy}} = 1/2(I + P)$ where I is the identity matrix. The stationary distribution of M_{lazy} is the same as that of M. Intuitively, a lazy Markov chain takes about twice as long to mix compared to the original chain.

7.2 Glauber dynamics Markov chains for cuts

We discuss Markov chains for all (s,t)-cuts sampled proportionally to their multiplicative weight, as well as Markov chains for sampling just minimum (s,t)-cuts.

We mentioned that our earlier dynamic programming results for bounded treewidth graphs can be easily modified to use additive weights and/or to be restricted to only minimum cuts. However, for Markov chain based sampling the situation is different and we present slow mixing examples for both scenarios.

Let Ω be the set of all minimum (s,t)-cuts. Consider a Glauber dynamics Markov chain that tries to modify a single site in each transition. Then, the transition from a current state C is as follows:

1. choose a random vertex $v \in V - \{s,t\}$,
2. let $C' := C \oplus \{v\}$, the symmetric difference of C and $\{v\}$,
3. if $C' \in \Omega$, then C' is the next state; otherwise, the chain stays in C.

As this chain is symmetric, its stationary distribution is uniform. The chain moves from C to C' with probability $1/(n-2)$ if the chain is not lazy and with probability $1/(2(n-2))$ for its lazy version. More general Glauber dynamics chains attempt to modify more sites in one transition. For c modified sites the transition probabilities are $\Theta(1/n^c)$ and the chain is increasingly more likely to reject a move in step 3 due to $C' \notin \Omega$. As such, Markov chains that modify the current state *locally*, in other words, by changing only a constant-size part of the state, are generally preferred.

For weighted cuts, let Ω be the set of all (s,t)-cuts. The desired stationary distribution is $\pi_w(C) = w(C)/Z$, where $Z = \sum_{x\in\Omega} w(x)$ is the normalization factor, i.e, the partition function. The Metropolis-Hastings variant of the Glauber dynamics chain redefines step 3 as follows: if $C' \in \Omega$, then with probability $\min\{\pi(C')/\pi(C), 1\}$ state C' becomes the next state; otherwise, the chain stays in C. Notice that we do not need to know the (generally difficult to compute) normalization factor Z, since $\pi_w(C')/\pi_w(C) = w(C')/w(C)$.

7.3 Slow mixing for all weighted cuts

We present a simple family of graphs for which the lazy Metropolis-Hastings variant of the above Markov chain needs exponential time to mix.

Consider the following undirected weighted graph $G = (V,E,w)$, where $V = \{u_1,u_2,\ldots,u_n\}$ with edges $E = \{(u_i,u_{i+1}) \mid i \in [n-1]\}$ of weights $w(u_1,u_2) = w(u_{n-1},u_n) = 1$ and $w(u_i,u_{i+1}) = 1/2^n$ for $2 \le i \le n-2$. Let $s = u_1$ and $t = u_n$. Graph G is a path and therefore has treewidth 1.

In this case $\Omega := \{\{s\} \cup S \mid S \subseteq V - \{s,t\}\}$. Notice that there are only two (s,t)-cuts with weight 1, namely $\{u_1\}$ and $\{u_1,\ldots,u_{n-1}\}$, and that the (multiplicative) weight of any other (s,t)-cut is at most $1/2^n$. Therefore, $Z \le 1+1+(2^{n-2}-2)/2^n < 3$.

Let $A = \{\{s\}\}$. Then, $1/3 < \pi_w(A) = w(A)/Z = 1/Z < 1/2$. The probability of moving from cut $\{s\}$ to another cut $\{s,u_i\}$ is at most $1/(2(n-2)2^n)$ since we choose u_i with probability $1/(2(n-2))$ and accept the move with probability $w(\{s,u_i\})/w(\{s\}) \le 1/2^n$ (more precisely, the acceptance probability is $1/2^n$ if $i = 2$ and $(1/2^n)^2$ otherwise).

We claim that the conductance out of A is exponentially small, see (1):

$$\Phi \leq \frac{\sum_{x\in A, y\in\Omega-A}\pi_w(x)P_w(x,y)}{\pi_w(A)} = \frac{\sum_{y\in\Omega-A}\pi_w(\{s\})P_w(\{s\},y)}{\pi_w(\{s\})} =$$
$$= \sum_{y\in\Omega-A} P_w(\{s\},y) \leq (n-2)\frac{1}{2(n-2)2^n} = \frac{1}{2^{n+1}}.$$

Therefore, we can bound the mixing time, see (2):

$$\tau(\varepsilon) \geq \frac{1}{2}(\frac{1}{2\Phi}-1)\log(\frac{1}{2\varepsilon}) \geq \frac{1}{2}(2^n-1)\log(\frac{1}{2\varepsilon}).$$

The mixing time is exponential in n. If we instead want to bound the mixing time in terms of input size, then note there are $n-1$ edge weights, each of up to n bits. Therefore, the size of the input is $\Theta(n^2)$ and the mixing time is still super polynomial, as it is exponential in the square root of the size of the input.

7.4 Slow mixing for minimum (s,t)-cuts

We conclude this paper with a family of graphs for which the Markov chain for minimum (s,t)-cuts needs exponential time to mix. For simplicity we assume additive weights, as is standard for minimum (s,t)-cuts due to their correspondence to maximum *s-t* flows. We note that the same example with edge weights $1/2$ yields slow mixing arguments in case of multiplicative weights.

For any integer $\ell \geq 1$, consider the following undirected unweighted graph $G=(V,E)$, where $V = \{s,t,u,a_1,a_2,\ldots,a_\ell,b_1,b_2,\ldots,b_\ell\}$ and $E = \{(s,a_i),(a_i,u),(u,b_i),(b_i,t) \mid i\in[\ell]\}$. Graph G is series-parallel and therefore has treewidth at most 2. It has $n=2\ell+3$ vertices. Any (s,t)-cut with value ℓ is minimum (since the value of the maximum *s-t* flow is ℓ). Therefore, $A := \{\{s\}\cup C_a \mid C_a \subseteq \{a_1,\ldots,a_\ell\}\}$ and $B := \{\{s,a_1,\ldots,a_\ell,u\}\cup C_b \mid C_b \subseteq \{b_1,\ldots,b_\ell\}\}$ are sets of minimum (s,t)-cuts. We claim that there are no other minimum (s,t)-cuts.

Lemma 2. *If C is a minimum (s,t)-cut, then $C\in A$ or $C\in B$.*

Proof. Suppose that $u\notin C$. Then, for every i, either (s,a_i) or (a_i,u) is cut; for total cut cost ℓ. If there is a $b_j\in C$, then (b_j,t) increases the cut cost beyond ℓ, a contradiction. In this case, $C\in A$.

Suppose that $u\in C$. Then, for every j, either (u,b_j) or (b_j,t) is cut; for total cut cost ℓ. If there is an $a_i\notin C$, then (s,a_i) increases the cut cost beyond ℓ, a contradiction. In this case, $C\in B$.

Therefore, $\Omega = A\cup B$. Notice that to move from A to B one has to pass through the state $y' := \{s,a_1,\ldots,a_\ell,u\}$; however, there is a single state $x' := \{s,a_1,\ldots,a_\ell\}$ in A that can move to y'. Since $|A|=|B|=2^\ell$, we have $\pi(A)=1/2$ and $\pi(x)=1/2^{\ell+1}$ for any $x\in\Omega$. Therefore, we can bound the conductance of the lazy chain as follows:

$$\Phi \leq \frac{\sum_{x\in A, y\in \Omega - A} \pi(x)P(x,y)}{\pi(A)} = \frac{1/(2^{\ell+1})1/(2(n-2))}{1/2} = \frac{1}{2^{(n-1)/2}(n-2)}.$$

Then, the mixing time is bounded by:

$$\tau(\varepsilon) > \frac{1}{2}(\frac{1}{2\Phi} - 1)\log(\frac{1}{2\varepsilon}) \geq (2^{(n-5)/2}(n-2) - 1)\log(\frac{1}{2\varepsilon}).$$

Thus, we need exponential mixing time in n to get ε-close to the uniform distribution even if ε is a constant. The computation can be adapted to show exponential mixing time for Glauber dynamics Markov chains that change c vertices at a time for any constant c.

8 Conclusions

In this paper, we presented a simple dynamic programming algorithm to compute the partition function for weighted cuts of a bounded treewidth graph. This algorithm easily extends to multiple source multiple sink cuts as well. We also provided an algorithm to sample cuts under our framework in the same amount of time, and demonstrated that Markov chain techniques to generate cuts require exponential time to converge in our setting.

We remark that in many computer vision applications the graph is a grid graph with two extra vertices, the source and the sink, that are each connected to a set of grid vertices. This situation arises, for example, for the Random Markov Field model. When using maximum likelihood to determine the best parameters for the model, one needs to compute the partition function across all weighted cuts. Unfortunately, this graph does not have bounded treewidth. We leave the study of evaluating cut problems for planar graphs with two apices, the source and the sink, for future work.

Acknowledgements

Portions of this research took place during the authors' participation in Dagstuhl Seminar 13421: Algorithms for Optimization Problems in Planar Graphs.

References

1. Arnborg, S., Corneil, D., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM Journal on Algebraic Discrete Methods **8**(2), 277–284 (1987). DOI 10.1137/0608024. URL http://epubs.siam.org/doi/abs/10.1137/0608024
2. Ball, M.O., Provan, S.J.: Calculating bounds on reachability and connectedness in stochastic networks. Networks **13**, 253–278 (1983)

3. Bezáková, I., Friedlander, A.J.: Counting and sampling minimum (s,t)-cuts in weighted planar graphs in polynomial time. Theoret. Comput. Sci. **417**, 2–11 (2012)
4. Bodlaender, H.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing **25**(6), 1305–1317 (1996). DOI 10.1137/S0097539793251219. URL http://epubs.siam.org/doi/abs/10.1137/S0097539793251219
5. Bodlaender, H.L.: Dynamic programming on graphs with bounded treewidth. In: T. Lepistö, A. Salomaa (eds.) Automata, Languages and Programming, *Lecture Notes in Computer Science*, vol. 317, pp. 105–118. Springer Berlin Heidelberg (1988)
6. Boykov, Y., Veksler, O.: Graph cuts in vision and graphics: Theories and applications. In: N. Paragios, Y. Chen, O. Faugeras (eds.) Handbook of Mathematical Models in Computer Vision, pp. 79–96. Springer US (2006)
7. Chambers, E.W., Fox, K., Nayyeri, A.: Counting and sampling minimum cuts in genus g graphs. In: Proceedings of the Twenty-ninth Annual Symposium on Computational Geometry, SoCG '13, pp. 249–258. ACM, New York, NY, USA (2013). DOI 10.1145/2462356.2462366. URL http://doi.acm.org/10.1145/2462356.2462366
8. Colbourn, C.J.: Combinatorial aspects of network reliability. Annals Operations Research **33**, 1–15 (1991)
9. Courcelle, B.: The monadic second-order logic of graphs i. recognizable sets of finite graphs. Information and Computation pp. 12–75 (1990)
10. Courcelle, B., Makowsky, J., Rotics, U.: On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. Discrete Applied Mathematics **108**(1–2), 23 – 52 (2001). DOI http://dx.doi.org/10.1016/S0166-218X(00)00221-3. URL http://www.sciencedirect.com/science/article/pii/S0166218X00002213. Workshop on Graph Theoretic Concepts in Computer Science
11. Grohe, M.: Algorithmic meta theorems. In: H. Broersma, T. Erlebach, T. Friedetzky, D. Paulusma (eds.) Graph-Theoretic Concepts in Computer Science, *Lecture Notes in Computer Science*, vol. 5344, pp. 30–30. Springer Berlin Heidelberg (2008)
12. Halin, R.: S-functions for graphs. Journal of Geometry **8**(1-2), 171–186 (1976). DOI 10.1007/BF01917434. URL http://dx.doi.org/10.1007/BF01917434
13. Jerrum, M.: Counting, Sampling and Integrating: Algorithms and Complexity. Lectures in Mathematics. ETH Zürich. SPRINGER VERLAG NY (2003). URL http://books.google.com/books?id=aLINQMsDQQ0C
14. Karger, D.R.: A randomized fully polynomial time approximation scheme for the all terminal network reliability problem. In: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing, STOC '95, pp. 11–17. ACM, New York, NY, USA (1995)
15. Lawler, G., Sokal, A.: Bounds on the l2 spectrum for markov chains and markov processes: a generalization of cheeger's inequality. Transactions of the American Mathematical Society **309**, 557–580 (1988)
16. Nagamoch, H., Sun, Z., Ibaraki, T.: Counting the number of minimum cuts in undirected multigraphs. IEEE Transactions on Reliability **40**, 610–614 (1991)
17. Provan, S.J., Ball, M.O.: The complexity of counting cuts and of computing the probability that a graph is connected. SIAM Journal of Computing **12**, 777–788 (1983)
18. Robertson, N., Seymour, P.: Graph minors. iii. planar tree-width. Journal of Combinatorial Theory, Series B **36**(1), 49 – 64 (1984). DOI http://dx.doi.org/10.1016/0095-8956(84)90013-3. URL http://www.sciencedirect.com/science/article/pii/0095895684900133
19. Sinclair, A., Jerrum, M.: Approximate counting, uniform generation and rapidly mixing markov chains. Inf. Comput. **82**(1), 93–133 (1989)