COMPUTING THE TUTTE POLYNOMIAL OF LATTICE PATH
MATROIDS USING DETERMINANTAL CIRCUITS

JASON MORTON AND JACOB TURNER

Abstract. We give an $O(n^4)$ algorithm computing the Tutte polynomial of a lattice path matroid, where $n$ is the size of the ground set of the matroid. The best existing algorithm was $O(n^5)$. Furthermore, this can be improved to $O(n^2)$ if we evaluate the Tutte polynomial on a given input, fixing the values of the variables. Conceptually, our algorithm embeds the computation in a determinant using a recently demonstrated equivalence of categories useful for counting problems.

1. Introduction

Lattice path matroids were introduced in [1] as a particularly well-behaved and yet very interesting class of matroids. In the same paper, the authors prove that computation of the Tutte polynomials of lattice path matroids is polynomial time. In [2], they go on to prove that the time complexity of computing the Tutte polynomial is $O(n^5)$, where $n$ is the size of the ground set of the matroid.

We give an algorithm that improves the complexity to $O(n^4)$ using determinantal circuits [3]. Then we show that evaluating the Tutte polynomial on a specific input (fixed values of $x$ and $y$) can be done in time $O(n^2)$. The paper is organized as follows: first we discuss weighted lattice paths and their relation to determinantal circuits. Then we recall the definitions of lattice path matroids and the relevant theorems from [1, 2]. Lastly, we give an explicit algorithm for computing the Tutte polynomial and analyze its time complexity.

\begin{figure}[h]
\centering
\begin{subfigure}{0.3\textwidth}
\centering
\begin{tikzpicture}
\node at (0,0) {$(0,0)$};
\node at (1,1) {$(4,5)$};
\draw (0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle;
\end{tikzpicture}
\caption{Lattice bounded by two monotone paths $P$ (lower) and $Q$ (upper).}
\end{subfigure}
\begin{subfigure}{0.3\textwidth}
\centering
\begin{tikzpicture}
\node at (0,0) {$w_1$};
\node at (1,1) {$w_2$};
\node at (1,0) {$w_4$};
\node at (0,1) {$w_5$};
\node at (1,2) {$w_7$};
\node at (0,2) {$w_6$};
\draw (0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle;
\end{tikzpicture}
\caption{Weighted lattice.}
\end{subfigure}
\caption{Lattices}
\end{figure}
2. Weighted Lattice Paths

Let us consider $\mathbb{Z}^2$ as an infinite graph where two points are connected if they differ by $(\pm 1, 0)$ or $(0, \pm 1)$. Suppose we are given two monotone paths on $\mathbb{Z}^2$, $P$ and $Q$, that both start at $(0,0)$ and end at $(m,r)$. Furthermore, suppose that $P$ is never above $Q$ in the sense that there are no points $(p_1,p_2) \in P$, $(q_1,q_2) \in Q$ such that $p_1 - q_1 < 0$ and $p_2 - q_2 > 0$. We are interested in subgraphs of $\mathbb{Z}^2$ bounded by such pairs of paths. From here on out, “lattice” means of subgraph of this form. An example is given by Figure 1(a).

Let $E$ be the set of edges of a lattice $G$. Suppose for each $e \in E$, we assign it a weight, $w(e)$. We call this a weighted lattice. Given a monotone path $C \subseteq G$, we define the weight of $C$ to be the product of its edge weights

$$w(C) = \prod_{e \in C} w(e).$$

**Definition 2.1.** Let $G$ be a lattice bounded by two paths with common endpoint $(m,r)$. A full path in $G$ is monotone path from $(0,0)$ to $(m,r)$.

**Definition 2.2.** Let $\mathcal{F}$ be the set of full paths of a weighted lattice $G$. The value of $G$ is defined to be

$$\sum_{C \in \mathcal{F}} w(C).$$

In Figure 1(b), there are three full paths of the weighted lattice. The value of this lattice is $w_1w_2w_5 + w_3w_4w_5 + w_3w_6w_7$.

One way to assign matrices to every vertex of a weighted lattice to encode the weights of each edge. Let $G = (V,E)$ be a weighted lattice. For $v \in V$, we define the incoming edges of $v$ to be those edges below or to the left of $v$ incident to $v$. The other edges incident to $v$ are the outgoing edges of $v$.

The matrix we associate to $v$ has rows equal to the number of incoming edges and columns equal to the number of outgoing edges. We order the incoming edges of $v$ counter-clockwise starting with the incoming edge closest to the negative $x$-axis. We order the outgoing edges of $v$ clockwise starting with the outgoing edge closest to the positive $y$-axis. This order defines how to associate the edges of $v$ with rows and columns of the matrix. We fill each column with the weight of the outgoing edge of $v$ it corresponds to. For example:

$$
\begin{pmatrix}
  e_3 \\
  e_1 \\
  e_3 \\
  e_2 \\
\end{pmatrix}
\quad \begin{pmatrix}
  w(e_3) & w(e_4) \\
  w(e_3) & w(e_4) \\
\end{pmatrix}
$$

Note that every edge in a lattice is the outgoing edge of precisely one vertex, so given the matrices associated to the vertices, the weight on the edges can be recovered. For a vertex $v$, we denote the matrix associated with it by $M_v$. 

However, for the matrices associated with \((0, 0)\) and \((m, r)\), we do something slightly different. As defined, \(M_{(0,0)}\) would be have zero rows and \(M_{(m,r)}\) zero columns. We define \(M_{(0,0)}\) to have one row with the weights of the outgoing edges in the appropriate columns. We define \(M_{(m,r)}\) to have one column, with all entries 1.

### 2.1. Determinantal Circuits

We can turn a weighted lattice into a determinantal circuit in such a way that the value of the determinantal circuit is the value of the weighted lattice.

A determinantal circuit can be given as a series of matrices \(\{S_i\}_{i=1}^s\), which are called stacks. If \(M\) is an \(n \times m\) matrix, we define the function

\[
s\text{Det}(M) = \sum_{I \subseteq [n], J \subseteq [m]} M_{I,J} \langle I | J \rangle
\]

where \(M_{I,J}\) is minor of \(M\) including rows \(I\) and columns \(J\) and where \(|I\rangle = \bigotimes_{i \in [n]} v_i \chi(i,I)\), and \(|J\rangle = \bigotimes_{i \in [m]} v_i^* \chi(i,J)\). The indicator function \(\chi(i,I) = 1\) if \(i \in I\) and 0 otherwise. This function is applied to each stack and is functorial. That is, \(s\text{Det}\) defines a monoidal functor between a category whose morphisms are \(n \times m\) matrices to a category whose morphisms are \(2^n \times 2^m\) matrices that respects matrix multiplication, see [3] for details. The value of the determinantal circuit is the trace of the composition of the \(s\text{Det}(S_i)\):

\[
\text{Tr} \left( \prod_{i=1}^s s\text{Det} S_i \right).
\]

While the \(s\text{Det}\) function takes a matrix to an exponentially larger matrix, the following proposition still allows us to compute the value of determinantal circuit efficiently.

**Proposition 2.3 ([3]).**

\[
\text{Tr} \left( \prod_{i=1}^s s\text{Det} S_i \right) = \det \left( I + \prod_{i=1}^s S_i \right).
\]

Take a weighted lattice with the matrices described above assigned to each vertex. Figure 2 shows an example of how to determine the stacks of the determinantal circuit from the lattice. For each of the diagonal arrows, take the matrix direct sum...
of the matrices $M_v$ along the direction of the arrow, i.e., $M_v$ will be block diagonal. If $G$ is a weighted lattice, let $D_G$ be its associated determinantal circuit.

Let $G$ be a weighted lattice bounded by two paths $P$ and $Q$ from $(0,0)$ to $(m,r)$, and $\mathcal{F}$ the set of full paths of $G$. If $C \in \mathcal{F}$, we describe it as a series of triples $(v_i, e_i, e_{i+1})$ where $e_i$ is the $i$th edge and $v_i$ is the common vertex of $e_i$ and $e_{i+1}$, $i$ ranging from 1 to $n = m + r$. The following is a special case of Proposition 4.3 in [3].

**Proposition 2.4.** The value of $D_G$ is given by the expression

$$\sum_{C \in \mathcal{F}} \left( \det(M_{v_1,1,e_1}) \det(M_{v_n,e_n,1}) \prod_{(v_i,e_i,e_{i+1}) \in P} \det(M_{v_i,e_i,e_{i+1}}) \right)$$

where $M_{v_i,e_i,e_{i+1}}$ is the minor of $M_v$ specified by the edges $e, e'$. 

**Corollary 2.5.** The value of $D_G - 1$ is equal to the value of $G$.

**Proof.** By the way we constructed $M_{v_1,1}$, $M_{v_i,e_i,e_{i+1}}$ is a $1 \times 1$ minor with entry $w(e_{i+1})$. Furthermore $M_{v_1,1,e_1}$ has single entry $w(e_1)$ and $M_{v_n,e_n,1}$ has single entry 1. Thus

$$\sum_{C \in \mathcal{F}} \left( \det(M_{v_1,1,e_1}) \det(M_{v_n,e_n,1}) \prod_{(v_i,e_i,e_{i+1}) \in P} \det(M_{v_i,e_i,e_{i+1}}) \right)$$

$$= \sum_{C \in \mathcal{F}} \prod_{i=1}^n w(e_i) = \sum_{C \in \mathcal{F}} w(C).$$

However, the case of the empty path is counted in this value, so if we subtract one from the value of $D_G$, we get the value of $G$. \qed

### 3. Lattice Path Matroids

Recall that a *matroid* is a pair $(G,I)$ where $G$, the ground set, is a finite set and $I$, the independent sets, are a collection of subsets of $G$ such that (i) $\emptyset \in I$, (ii) if $A \in I$ and $A' \subseteq A$, then $A' \in I$, and (iii) if $A, B \in I$ and $|A| > |B|$, then $\exists a \in A \setminus B$ such that $B \cup \{a\} \in I$. Lattice path matroids are defined with respect to a lattice bounded by two monotone paths $P$ and $Q$ from $(0,0)$ to $(m,r)$ as described before. Since $P$ and $Q$ are both monotone, they can be described by a string of $n = m + r$ 0’s and 1’s where '1' corresponds to moving up one step and '0' corresponds to moving east one step.

**Definition 3.1 ([1]).** Let $P = p_1 \cdots p_n$ and $Q = q_1 \cdots q_n$ be two lattice paths from $(0,0)$ to $(m,r)$ with $P$ never going above $Q$. Let $u_1 < \cdots < u_r$ be the set of indices such that $p_{u_i} = 1$. Let $l_1 < \cdots < l_r$ be similarly for $Q$. Then let $N_i$ be the
interval \([i, u_i]\) of integers. Define \(M[P, Q]\) to be the matroid with ground set \([n]\) and presentation \((N_i : i \in [r])\). A lattice path matroid is any matroid isomorphic to some \(M[P, Q]\).

Lattice path matroids are a very nice example of matroids and of interest is the Tutte polynomial of such matroids. It turns out that this can be given as the value of a particular weighting of the lattice defining the matroid. The following is a slight restatement of the original theorem:

**Theorem 3.2** ([1]). The Tutte polynomial of a lattice path matroid \(M[P, Q]\) is the value of the lattice \(G\) defined by \(P\) and \(Q\) with the north steps of \(Q\) having weight \(x\), the east steps of \(P\) having weight \(y\), and all other lattice weights equal to 1.

An example is shown in Figure 3. Beside the edges in bold are the corresponding weights. The weight of each non-bold edge is simply 1.

4. **Algorithm for Computing the Tutte Polynomial**

We now give the algorithm for computing the Tutte polynomial in pseudocode. We assume that we are given as input two monotone paths \(P\) and \(Q\) from \((0, 0)\) to \((m, r)\) such that \(P\) never goes above \(Q\). These paths are given to us as a list of 1’s and 0’s where a ’1’ denotes a step north and a ’0’ denotes a step east.

**Distance**\((P,Q,i)\):
- Find the number of matrices in the \(i^{th}\) stack. This is how many more north steps \(Q\) has made than \(P\) by the \(i^{th}\) stack.

**M\((v)\)**:
- For input vertex \(v\), return \(M_v\).

**Tutte**\((P,Q)\):
- Let \(T\) be the length-one row vector \((1)\)
  - for each stack \(i\):
    - Let \(A\) be the empty list
      - for each node, \(v\), in the stack (there are Distance\((P,Q,i)\) nodes):
        - Append \(M(v)\) to \(A\)
        - Update the vector \(T = (T_1 A[1], \ldots, T_i A[i], \ldots)\)
  - After the last stack, \(T\) is 1 by 1, \(T = (t)\). Return \(t\)

Let \(n = m + r\) be the length of \(P\) and \(Q\). The function \(Tutte\) iterates over the stacks. The function \(Distance\) calculates the number of lattice points in a stack. For each vertex in the stack, the matrix \(M(v)\) associated to the vertex is found by a constant-time lookup.

After the matrices are calculated, we could take their direct sum and multiply it by \(T\). For example, the matrices associated to the nine stacks in Figure 3, each of which is a direct sum of \(M(v)\), are multiplied to obtain

\[
\begin{pmatrix}
  x & y \\
  x & y
\end{pmatrix}
\begin{pmatrix}
  x & 0 & 1 \\
  1 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
  1 & 0 & 0 \\
  1 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
  x & y & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  1 & 0 & 0 \\
  1 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
  0 & 1 & 0 \\
  0 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
  0 & 1 & 0 \\
  0 & 1 & 0
\end{pmatrix},
\]

which equals the Tutte polynomial \((x^2 + xy + y^2 + x + y)(x + y + y^2)\).
We can be more careful, however, to improve the running time since each matrix \( S_i \) is block-diagonal with block size at most 2.

\( T \) starts out as a (row) vector and each iteration of the main loop updates \( T \) to a new vector. Since at each stage we are multiplying \( T \) by a block diagonal matrix, we can partition \( T \) into \( T_1, \ldots, T_i, \ldots, q \) such that multiplying \( T \) by \( \bigoplus M_{v_i} \) is the same as calculating \( (T_1 M_{v_1}, \ldots, T_i M_{v_i}, \ldots) \). Eventually, \( T \) becomes a 1 \( \times \) 1 matrix and the algorithm then returns its sole entry.

Inside of the main for loop of the function \texttt{tutte}, there are three main contributors to the time complexity: the function \texttt{Distance}, finding the matrix associated to a vertex, and the matrix multiplication.

If we let \( P[i] \) be the \( i^{th} \) bit of \( P \) and \( Q[i] \) likewise. Then \texttt{Distance} finds \( \sum_{k \leq i} Q[k] - P[k] \). This runs in time \( O(n) \).

For the function \texttt{M}(\( v \)), there are a fixed finite of number of cases that need to be checked depending on which entering and exiting edges it has and whether or not \( v \) lies on \( P \) or \( Q \). This runs in time \( O(1) \).

For any stack, there are at most \( n \) vertices in the stack, and each \( M_{v_i} \) has at most two rows and at most two columns. If we specify values for \( x \) and \( y \), calculating \( (T_1 M_{v_1}, \ldots, T_i M_{v_i}, \ldots) \) takes at most \( 16n \) computations. Thus it is \( O(n) \).

Both the function \texttt{distance} and updating \( T \) runs in time \( O(n) \) inside the main loop. Since there are \( n \) iterations of the main for loop, the overall time complexity is \( O(n^2) \).

Should we wish to compute the polynomial itself rather than its value, we need to account for multiplying intermediate polynomials by \( 1, x, \) or \( y \), and adding them. The polynomials in the intermediate vector \( T \) after stack \( i \) are of degree at most \( i \) so have at most \( \binom{i}{2} \) terms. The last application of an \( S_i \) can involve adding quadratically large polynomials and so the additional cost is at most \( O(n^2) \).

Then updating the vector can be done in time \( O(n^3) \) and this lies in a for loop that iterates at most \( n \) times. So the overall algorithm is \( O(n^4) \). This is indeed an improvement over \cite{BoninGimenez}.

\begin{thebibliography}{9}
\bibitem{BoninMierNoy} J. Bonin, A. De Mier, and M. Noy. Lattice path matroids: enumerative aspects and Tutte polynomials. \textit{Journal of Combinatorial Theory, Series A}, 104(1):63–94, 2003.
\bibitem{BoninGimenez} J.E. Bonin and O. Gimenez. Multi-path matroids. \textit{Combinatorics, Probability and Computing}, 16(02):193–217, 2007.
\bibitem{MortonTurner} Jason Morton and Jacob Turner. Generalized counting constraint satisfaction problems with determinantal circuits. \textit{Arxiv preprint arXiv:1302.1892}, 2013.
\end{thebibliography}