Weighted Tiling Systems for Graphs: Evaluation Complexity

C. Aiswarya
Chennai Mathematical Institute, India
IRL ReLaX, CNRS, France
aiswarya@cmi.ac.in

Paul Gastin
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
paul.gastin@ens-paris-saclay.fr

Abstract

We consider weighted tiling systems to represent functions from graphs to a commutative semiring such as the Natural semiring or the Tropical semiring. The system labels the nodes of a graph by its states, and checks if the neighbourhood of every node belongs to a set of permissible tiles, and assigns a weight accordingly. The weight of a labeling is the semiring-product of the weights assigned to the nodes, and the weight of the graph is the semiring-sum of the weights of labelings. We show that we can model interesting algorithmic questions using this formalism - like computing the clique number of a graph or computing the permanent of a matrix. The evaluation problem is, given a weighted tiling system and a graph, to compute the weight of the graph. We study the complexity of the evaluation problem and give tight upper and lower bounds for several commutative semirings. Further we provide an efficient evaluation algorithm if the input graph is of bounded tree-width.

2012 ACM Subject Classification Theory of computation → Quantitative automata

Keywords and phrases Weighted graph tiling, tiling automata, Evaluation, Complexity, Tree-width

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2020.34

Related Version An extended version of the paper is available at `https://arxiv.org/abs/2009.14542`

Funding Supported by IRL ReLaX.
C. Aiswarya: Supported by DST Inspire.

1 Introduction

Weighted automata have been classically studied over words, as they naturally extend automata from representing languages to representing functions from words to a semiring.

We are interested in finite state formalisms for representing functions from *graphs* to a semiring. Many natural algorithmic questions on graphs are about computing a function, such as the clique number, weight of the shortest path etc. It is interesting to see if one can design weighted automata to model such problems. Further can one design efficient algorithms for problems modeled by such weighted automata?

We study weighted tiling systems (WTS), a variant of the weighted graph automata of Droste and Dück [10], motivated by the graph acceptors of Thomas [24]. This subsumes many quantitative models that have been studied on words, trees [13, 14], nested words [22], pictures [17], Mazurkiewicz traces [11, 23, 5], etc. The reader is referred to the handbook [12] for more details and references. Many of these works are mainly interested in expressivity questions, and show that the model has good expressive power. The model is also easy to understand as it is formulated in terms of tiling/colouring respecting local constraints. We reiterate the expressivity by modeling computational problems on graphs using this model.

Our focus is on the computational complexity of the evaluation problem. It is closer in spirit to [18] which provides an efficient evaluation algorithm for weighted pebble automata on words.

We show that many algorithmic questions, like computing the clique number, computing the permanent of a matrix, or counting-variants of SAT, can be naturally modeled using this formalism. We investigate the computational complexity of the evaluation problem and obtain tight upper- and lower-bounds for various semirings.

To give more details, a WTS has a finite number of states and a run labels the vertices of a graph with states. The tiles (analogous to transitions) observe the neighbourhood of a vertex under the labeling, and assign a weight accordingly. The weight of the run is the semiring-product of the weights thus assigned, and the weight assigned to a graph is the semiring-sum of the weights of the runs. We only consider commutative semirings and hence the order in which the product is taken does not matter.

The evaluation problem is to compute the weight of an input graph in an input WTS. We study the computational complexity of this problem for various semirings. Over Natural semiring and non-negative rationals, the problem is shown to be #P-complete. Over integers and rationals the problem is GapP-complete. Over tropical semirings – $(\mathbb{N}, \max, +), (\mathbb{Z}, \max, +), (\mathbb{N}, \min, +), (\mathbb{Z}, \min, +)$ – the problem is $\mathrm{FP}^{\mathrm{NP}[log]}$ complete.

We further consider the evaluation problem for graphs of bounded tree-width and show that they are computable in time polynomial in the WTS and linear in the graph. Bounded tree-width captures a variety of formal models of concurrent and infinite state systems such as Mazurkiewicz traces, nested words, and decidable under-approximations of message passing automata or multi-pushdown automata [21, 1, 2, 9, 4].

Even though our focus is evaluation, and not expressiveness of the model, we get a deep insight into the modeling power of this formalism through the upper and lower complexity bounds. For instance, we cannot polynomially encode the traveling salesman problem (lower bound $\mathsf{FP}^{\mathrm{NP}}$) in our formalism over tropical semiring (upper bound $\mathsf{FP}^{\mathrm{NP}[\log]}$) unless the polynomial hierarchy collapses [19].

2 Model

First we will fix the notations for semirings, graphs and then introduce the WTS formally.

Preliminaries. Let $\mathbb{N}$ denote the set of natural numbers including 0, $\mathbb{Z}$ the integers, and $\mathbb{Q}$ the rationals.

Let $A = \{a_1, \ldots a_n\}$ and B be two sets. We sometimes write a function $f : A \to B$ explicitly by listing the image of each element: $f = [a_1 \mapsto f(a_1), \ldots, a_n \mapsto f(a_n)]$. The set of all functions from A to B is denoted B^A. If A is $\emptyset$ then the only relation (and hence function) from A to B is $\emptyset$. We denote this trivial empty function by $f_\emptyset$.

Let M be a non-deterministic Turing machine. The number of accepting runs of M on an input x is denoted $\#M(x)$, and the number of rejecting runs of M on x is denoted $\#\overline{M}(x)$.

A semiring is an algebraic structure $\mathbb{S} = (S, \oplus, \otimes, 0_{\mathbb{S}}, 1_{\mathbb{S}})$ where S is a set, $\oplus$ and $\otimes$ are two binary operations on S, $(S, \oplus, 0_{\mathbb{S}})$ is a commutative monoid, $(S, \otimes, 1_{\mathbb{S}})$ is a monoid, $\otimes$ distributes over $\oplus$, $0_{\mathbb{S}}$ is an annihilator for $\otimes$. A semiring is *commutative* if $\otimes$ is commutative.

Examples are $\mathsf{Boolean} = (\{0,1\}, \vee, \wedge, 0, 1)$, $\mathsf{Natural} = (\mathbb{N}, +, \times, 0, 1)$, $\mathsf{Integer} = (\mathbb{Z}, +, \times, 0, 1)$, $\mathsf{Rational} = (\mathbb{Q}, +, \times, 0, 1)$ and $\mathsf{Rational}^+ = (\mathbb{Q}_{\geq 0}, +, \times, 0, 1)$. Further examples are tropical semirings: $\mathsf{max\text{-}plus\text{-}}\mathbb{N} = (\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$, $\mathsf{max\text{-}plus\text{-}}\mathbb{Z} = (\mathbb{Z} \cup \{-\infty\}, \max, +, -\infty, 0)$, $\mathsf{min\text{-}plus\text{-}}\mathbb{N} = (\mathbb{N} \cup \{+\infty\}, \min, +, +\infty, 0)$ and $\mathsf{min\text{-}plus\text{-}}\mathbb{Z} = (\mathbb{Z} \cup \{+\infty\}, \min, +, +\infty, 0)$. We will consider only these semirings in this paper. Note that all these semirings are commutative.

Graphs. We consider graphs with different sorts of edges. For example, a grid will have horizontal successor edges, and vertical successor edges. A binary tree will have left-child relations and right-child relations. Message sequence charts will have process-successor relations and message send-receive relations. These graphs have bounded degree, and for each sort of edge, a vertex will have at most one outgoing/incoming edge of that sort[1]. Our definition of graphs below allows to capture such graph classes.

Let Γ be a finite set of edge names, and let Σ be a finite set of node labels. A (Γ, Σ)-graph $G = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda)$ has a finite set of vertices V, an edge relation $E_\gamma \subseteq V \times V$ for every $\gamma \in \Gamma$, and a mapping $\lambda : V \to \Sigma$ assigning a label from Σ to each vertex $v \in V$. The graphs we consider will have at most one outgoing edge and at most one incoming edge for every edge name. That is, for each $\gamma \in \Gamma$, for all $v \in V$, $|\{u \mid (v,u) \in E_\gamma\}| \le 1$ and $|\{u \mid (u,v) \in E_\gamma\}| \le 1$.

The type of a vertex is determined by the set of names of incoming edges and the set of names of outgoing edges. For example, the root of a tree has no incoming left-child or right-child edges and leaves of a tree have no outgoing left- or right-child. A type $\tau = (\Gamma_{\text{in}}, \Gamma_{\text{out}})$ indicates that the set of incoming (resp. outgoing) edge names is Γ_{in} (resp. Γ_{out}). Let $\mathsf{Types} = 2^\Gamma \times 2^\Gamma$ be the set of all types. We define $\mathsf{type}: V \to \mathsf{Types}$ and use $\mathsf{type}(v)$ to denote the type of vertex v.

▶ Remark 1. Even though we consider only bounded degree graphs, we are able to model graph functions on arbitrary graphs (even edge weighted) as illustrated in the examples below. Basically an arbitrary graph is input via its adjacency matrix, which is naturally a grid, a special case of the graphs that we can handle. We can even model problems on arbitrary graphs with edge weights.

A weighted Tiling System. is a finite state mechanism for defining functions from a class of graphs to a weight domain. It has a finite set of states and a set of permissible tiles for each type of vertices. Formally, a *weighted tiling system* (WTS) over (Γ, Σ)-graphs and a semiring $\mathbb{S} = (S, \oplus, \otimes, 0_\mathbb{S}, 1_\mathbb{S})$ is a tuple $\mathcal{T} = (Q, \Delta, \mathsf{wgt})$ where

- Q is the finite set of states,
- $\Delta = \bigcup_{\tau\in\mathsf{Types}} \Delta_\tau$ – for a type $\tau = (\Gamma_{\text{in}}, \Gamma_{\text{out}}) \in \mathsf{Types}$, the set $\Delta_\tau \subseteq Q^{\Gamma_{\text{in}}} \times Q \times \Sigma \times Q^{\Gamma_{\text{out}}}$ gives the set of permissible tiles of type τ,
- $\mathsf{wgt}: \Delta \to S$, assigns a weight for each tile.

A run ρ of $\mathcal{T}$ on a graph $G = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda)$ is a labeling of the vertices by states that conforms to Δ. Given a labeling $\rho: V \to Q$, for a vertex $v \in V$ with $\mathsf{type}(v) = (\Gamma_{\text{in}}, \Gamma_{\text{out}})$ we define the tile of v wrt. ρ to be $\mathsf{tile}_\rho(v) = (f_{\text{in}}, \rho(v), \lambda(v), f_{\text{out}})$ where $f_{\text{in}}: \Gamma_{\text{in}} \to Q$ is given by $\gamma \mapsto \rho(u)$ if $(u,v) \in E_\gamma$ and $f_{\text{out}}: \Gamma_{\text{out}} \to Q$ is given by $\gamma \mapsto \rho(u)$ if $(v,u) \in E_\gamma$. A labeling $\rho: V \to Q$ is a *run* if for each $v \in V$, $\mathsf{tile}_\rho(v) \in \Delta_{\mathsf{type}(v)}$.

The *weight of a run* ρ, denoted $\mathsf{wgt}(\rho)$, is the product of the weights of the tiles in ρ. With commutative semirings, we do not need to specify an order for this product. The value $[\![\mathcal{T}]\!](G)$ computed by $\mathcal{T}$ for a graph G is the sum of the weights of the runs. That is,

$$[\![\mathcal{T}]\!](G) = \bigoplus_{\rho \mid \rho \text{ is a run of } \mathcal{T} \text{ on } G} \mathsf{wgt}(\rho) \qquad\qquad \mathsf{wgt}(\rho) = \bigotimes_{v\in V} \mathsf{wgt}(\mathsf{tile}_\rho(v)).$$

▶ Remark 2. The WTS is a variant of the weighted graph automata (WGA) of [10]. There are two main differences. First, WGA admits tiles of bigger radius and the tile size is a

[1] This choice is mainly for notational convenience, and is not really a restriction, provided we consider only bounded degree graphs. Another option would be to enumerate the neighbours in some order and address a neighbour as the ith incoming/outgoing neighbour.

parameter. This is not more powerful, as it can be realized with immediate neighborhood tiles like in WTS. Second, WGA allows occurrence constraints. We discuss this in more detail in Section 5.

We give some examples of WTS below, which will also serve as reductions proving complexity lower-bounds in Section 3.

▶ **Example 3** (A WTS to compute the clique number of a graph)**.** The *clique number* of a graph is the size of the largest clique in the graph.

The graphs on which we want to compute the clique number have unbounded degrees indeed. In our setting we consider only bounded degree graphs. Hence we need to encode any arbitrary graph as a bounded degree graph. One way to do that is to consider the adjacency matrix and represent this matrix using a grid graph.

For the particular case of clique number, our input is an undirected graph, so we will consider a lower-right triangular matrix in a lower-right triangular grid graph. For this we let $\Gamma = \{\rightarrow, \downarrow\}$ and $\Sigma = \{0, 1\}$. The labels of all diagonal vertices are 1. A graph is depicted in Figure 1 and its lower-right triangular adjacency matrix is depicted in Figure 2.

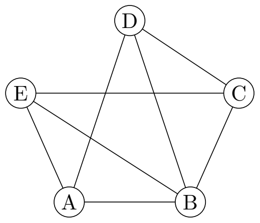

Figure 1 A graph

$$\begin{tikzcd}
 & & & & E & 1 \arrow[d] \\
 & & & D & 1 \arrow[r] \arrow[d] & 0 \arrow[d] \\
 & & C & 1 \arrow[r] \arrow[d] & 1 \arrow[r] \arrow[d] & 1 \arrow[d] \\
 & B & 1 \arrow[r] \arrow[d] & 1 \arrow[r] \arrow[d] & 1 \arrow[r] \arrow[d] & 1 \arrow[d] \\
A & 1 \arrow[r] & 1 \arrow[r] & 0 \arrow[r] & 1 \arrow[r] & 1
\end{tikzcd}$$

Figure 2 The lower-right triangular adjacency matrix of the graph of Figure 1 as a grid graph

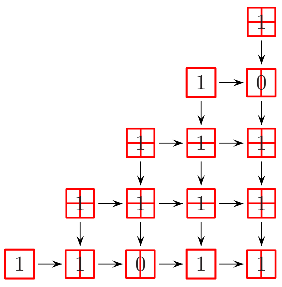

Figure 3 A run. Three tiles B, C and E gets weights 1, and hence the weight of this run is 3.

We will now construct a WTS over the tropical semiring max-plus-$\mathbb{N}$ that computes the clique number on a lower triangular grid graph. The run of the WTS will guess a subset of vertices of the original graph (corresponds to labeling some diagonal elements with state $\boxplus$) and checks that there is an edge between every pair of these (corresponds to checking the label is 1, if the row and column start in a $\boxplus$-labeled vertex). The weight of such a run will be the size of the subset, and the max over all the runs gives us the clique number as required.

Let $Q = \{\boxplus, \boxbar, \boxminus, \square\}$. A run will label a subset of diagonal vertices with $\boxplus$. A vertex is labeled with $\boxbar$ (resp. $\boxminus$, $\boxplus$) if its column (resp. row, both) starts in a vertex labeled $\boxplus$. In addition a vertex may get state $\boxplus$ only if its label is 1. All other vertices get state $\square$. A run on the graph in Figure 2 is depicted in Figure 3.

Tiles for diagonal vertices are given by $\Delta_{(\emptyset, \Gamma_{\text{out}})} = \{(f_\emptyset, \square, 1, f_{\text{out}}), (f_\emptyset, \boxplus, 1, f_{\text{out}})\}$. For an inside vertex we have (f_{out} being arbitrary in all tuples):

$$\begin{aligned}
\Delta_{(\{\rightarrow,\downarrow\},\Gamma_{\text{out}})} = {} & \{(f_{\text{in}}, \square, b, f_{\text{out}}) \mid b \in \{0,1\}, f_{\text{in}}(\rightarrow) \in \{\boxbar, \square\}, f_{\text{in}}(\downarrow) \in \{\boxminus, \square\}\} \\
& \cup \{(f_{\text{in}}, \boxplus, 1, f_{\text{out}}) \mid f_{\text{in}}(\rightarrow) \in \{\boxplus, \boxminus\}, f_{\text{in}}(\downarrow) \in \{\boxplus, \boxbar\}\} \\
& \cup \{(f_{\text{in}}, \boxminus, b, f_{\text{out}}) \mid b \in \{0,1\}, f_{\text{in}}(\rightarrow) \in \{\boxplus, \boxminus\}, f_{\text{in}}(\downarrow) \in \{\boxminus, \square\}\} \\
& \cup \{(f_{\text{in}}, \boxbar, b, f_{\text{out}}) \mid b \in \{0,1\}, f_{\text{in}}(\rightarrow) \in \{\boxbar, \square\}, f_{\text{in}}(\downarrow) \in \{\boxplus, \boxbar\}\}\,.
\end{aligned}$$

The weight of a tile of the form $(f_\varnothing, \boxplus, 1, f_{\text{out}})$ is 1. Notice that only the diagonal vertices labeled $\boxplus$ will get such a tile. The weight of all other tiles is 0. Thus the weight of a run is the number of diagonal vertices labeled $\boxplus$ - which corresponds to a subset of vertices inducing a clique. The maximum weight across different runs will compute the clique number as required. ◀

▶ **Example 4** (A WTS to compute the permanent of a (0,1)-matrix). We will model (0,1)-matrices as (0,1)-labelled grids. As in Example 3, we let $\Gamma = \{\rightarrow, \downarrow\}$ and $\Sigma = \{0, 1\}$. A 5×5 (0,1)-matrix as a grid graph is illustrated in Figure 4.

We will define a WTS $\mathcal{T}$ on such graphs over Natural such that $[\![\mathcal{T}]\!](G)$ is the permanent of the 0,1 matrix A represented by G. In each run exactly one vertex in each row and each column will be circled – representing one permutation σ of $\{1, \dots, n\}$ if G is an $n \times n$ grid. The weight of the tile on the circled vertex will be the vertex label (0 or 1) interpreted as an integer. Every other tile will have weight 1. Thus the weight of a run will be $\prod_i A(i, \sigma(i))$ where σ is the permutation represented by the run. Finally the value of a graph G representing an $n \times n$ $(0,1)$-matrix A will be $\sum_\sigma \prod_i A(i, \sigma(i))$ which is its permanent.

The WTS $\mathcal{T}$ has five states: $Q = \{◯, ⤷, ⤴, ⤶, ⤵\}$. We will define tiles so as to accept only the labeling reflecting the following:

- a vertex labeled ◯ means it is the circled vertex in its row and column,
- a vertex v labeled ⤷ means that the circled vertex in its column is upward of v, and the circled vertex in its row is to the right of v,
- similarly for other states ⤴, ⤶, ⤵.

The tiles are given formally below. The weight function wgt assigns weight 0 to any tile labeling a 0-labeled node with ◯. The weight of all other tiles is 1. A run of this WTS is illustrated in Figure 5.

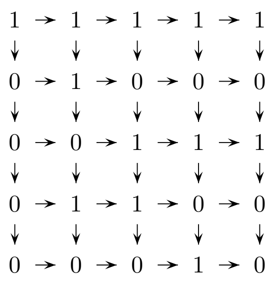
Figure 4 A 5×5 (0,1)-matrix as a grid graph

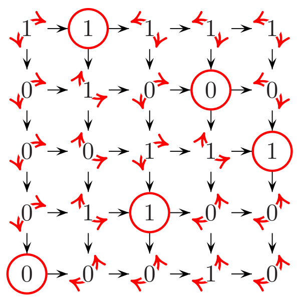
Figure 5 A run of the WTS $\mathcal{T}$ on the graph in Fig. 4. It has weight 0 as two tiles have wgt 0.

We now describe the tiles formally. For the top-left vertex we have

$$\begin{aligned}\Delta_{(\varnothing,\{\rightarrow,\downarrow\})} &= \{(f_\varnothing, ◯, b, f_{\text{out}}) \mid b \in \{0,1\}, f_{\text{out}}(\rightarrow) = ⤶, f_{\text{out}}(\downarrow) = ⤷\} \\ &\cup \{(f_\varnothing, ⤴, b, f_{\text{out}}) \mid b \in \{0,1\}, f_{\text{out}}(\rightarrow) \in \{⤴, ◯\}, f_{\text{out}}(\downarrow) \in \{⤴, ◯\}\}\end{aligned}$$

The tiles for other corner vertices are analogous. For the left border vertices we have $\Delta_{(\{\downarrow\},\{\rightarrow,\downarrow\})} =$

$$\begin{aligned}&\{(f_{\text{in}}, ◯, b, f_{\text{out}}) \mid b \in \{0,1\}, f_{\text{in}}(\downarrow) = ⤴, f_{\text{out}}(\rightarrow) \in \{⤶, ⤵\}, f_{\text{out}}(\downarrow) = ⤷\} \\ \cup\ &\{(f_{\text{in}}, ⤴, b, f_{\text{out}}) \mid b \in \{0,1\}, f_{\text{in}}(\downarrow) = ⤴, f_{\text{out}}(\rightarrow) \in \{⤷, ⤴, ◯\}, f_{\text{out}}(\downarrow) \in \{⤴, ◯\}\} \\ \cup\ &\{(f_{\text{in}}, ⤷, b, f_{\text{out}}) \mid b \in \{0,1\}, f_{\text{in}}(\downarrow) \in \{⤷, ◯\}, f_{\text{out}}(\rightarrow) \in \{⤷, ⤴, ◯\}, f_{\text{out}}(\downarrow) = ⤷\}\end{aligned}$$

The tiles for other border vertices are analogous. For an interior vertex, we have

$$\begin{aligned}
\Delta_{(\{\rightarrow,\downarrow\},\{\rightarrow,\downarrow\})} = {} & \{(f_{\text{in}},\bigcirc,b,f_{\text{out}}) \mid b\in\{0,1\}, f_{\text{in}}(\downarrow)\in\{↶,↷\}, f_{\text{in}}(\rightarrow)\in\{⤷,↷\},\\
& \qquad f_{\text{out}}(\rightarrow)\in\{⤶,↶\}, f_{\text{out}}(\downarrow)\in\{⤶,⤷\}\}\\
\cup {} & \{(f_{\text{in}},↷,b,f_{\text{out}}) \mid b\in\{0,1\}, f_{\text{in}}(\downarrow)\in\{↶,↷\}, f_{\text{in}}(\rightarrow)\in\{⤷,↷\},\\
& \qquad f_{\text{out}}(\rightarrow)\in\{⤷,\bigcirc,↷\}, f_{\text{out}}(\downarrow)\in\{↶,\bigcirc,↷\}\}\\
\cup {} & \{(f_{\text{in}},⤷,b,f_{\text{out}}) \mid b\in\{0,1\}, f_{\text{in}}(\downarrow)\in\{⤶,\bigcirc,⤷\}, f_{\text{in}}(\rightarrow)\in\{⤷,↷\},\\
& \qquad f_{\text{out}}(\rightarrow)\in\{⤷,\bigcirc,↷\}, f_{\text{out}}(\downarrow)\in\{⤶,⤷\}\}\\
\cup {} & \{(f_{\text{in}},⤶,b,f_{\text{out}}) \mid b\in\{0,1\}, f_{\text{in}}(\downarrow)\in\{⤶,\bigcirc,⤷\}, f_{\text{in}}(\rightarrow)\in\{⤶,\bigcirc,↶\},\\
& \qquad f_{\text{out}}(\rightarrow)\in\{⤶,↶\}, f_{\text{out}}(\downarrow)\in\{⤶,⤷\}\}\\
\cup {} & \{(f_{\text{in}},↶,b,f_{\text{out}}) \mid b\in\{0,1\}, f_{\text{in}}(\downarrow)\in\{↶,↷\}, f_{\text{in}}(\rightarrow)\in\{⤶,\bigcirc,↶\},\\
& \qquad f_{\text{out}}(\rightarrow)\in\{⤶,↶\}, f_{\text{out}}(\downarrow)\in\{↶,\bigcirc,↷\}\}
\end{aligned}$$

Finally, we describe the weight function wgt. The weight of a tile of the form $(f_{\text{in}},\bigcirc,0,f_{\text{out}})$ is 0. The weight of all other tiles is 1. ◀

▶ **Example 5** (Permanent of matrix with entries from $\mathbb{N}$). The purpose of this example is to illustrate that it is possible to encode natural numbers, which may appear as matrix entries or edge weights, also as bounded degree graphs with a fixed alphabet Σ.

A length k bit string $b_{k-1}\cdots b_1 b_0$ where $b_i \in \{0,1\}$ for all $0 \le i < k$, is represented by a path graph of length k. The vertices of this path graph are labelled with 1 or 0 to indicate the value of the bit, and the edges are labeled $<$. We describe a WTS on such path graphs whose computed weight is the binary number $\sum_i b_i 2^i$. The WTS guesses a prefix ending with label 1. All the nodes in the prefix take state q_0 and all nodes after the prefix may take the two states q_1 or q_2. The weight of all tiles is 1. The number of runs is $\sum_{i:b_i=1} 1^{k-i} \times 2^i = \sum_i b_i 2^i$.

As before, we will have an $n \times n$ grid graph to represent the matrix, but the vertices of the grid graph take a neutral label, say X. A path graph originates from every vertex of the grid graph indicating the entry of the matrix at that cell. Now, to compute the permanent, the path graphs starting from a circled vertex can start the WTS described in the previous paragraph. All other path graphs vertices can be labeled only by a special state q_4. The weights of all permissible tiles are 1. The weight computed by one permutation will indeed be the product of the entries. This crucially depends on the distributivity of the semiring. Thus, this WTS computes the permanent of an arbitrary matrix with entries in $\mathbb{N}$. ◀

Evaluation problem (Eval). is to compute $[\![\mathcal{T}]\!](G)$, given the following input:

$$\begin{aligned}
\mathcal{T} \quad & : \text{a WTS over } (\Gamma,\Sigma)\text{-graphs and a semiring } \mathbb{S}\text{, and}\\
G \quad & : \text{a } (\Gamma,\Sigma)\text{-graph.}
\end{aligned}$$

We study the complexity of this problem in Section 3, for various semirings. We provide an efficient algorithm for this problem in the case of bounded tree-width graphs in Section 4.

3 Evaluation complexity: Arbitrary graphs

Recall that we only consider the boolean semiring, the counting semirings over $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ or $\mathbb{Q}_{\ge 0}$ and the tropical semirings over $\mathbb{N}$ or $\mathbb{Z}$.

Given a WTS $\mathcal{T}$ and a graph G, we can compute $[\![\mathcal{T}]\!](G)$ in polynomial space as follows. Initialise the current aggregate to $0_{\mathbb{S}}$. Enumerate in lexicographic order through the different labelings of the vertices of G with states of $\mathcal{T}$. For each labeling, if it conforms to Δ, compute its weight and add to the current aggregate. Thus Eval belongs to FPSpace – the set of functions computable in polynomial space.

▶ **Theorem 6.** *Problem Eval is in FPSpace.*

However, for particular semirings the complexity is different as stated in the following subsections.

3.1 $(+,\times)$-semirings

▶ **Theorem 7.** *The evaluation problem is #P-complete over Natural, and non-negative Rational. It is GapP-complete over Integer and Rational.*

The upper bounds hold for arbitrary graphs, and the lower bounds hold for the special case of grids. The weights can be assumed to be given in binary.

A function f is in #P if there is an NP machine M such that $f(x) = \#M(x)$. That is, it denotes the set of function problems that correspond to counting the number of accepting paths in a non-deterministic polynomial time turing machine. Computing the permanent of a (0,1)- matrix is a #P-complete problem [25], and hence the #P-hardness claimed above follows from Example 4. We give an alternate hardness proof by a reduction from #-CNF-SAT.

A function $f(x)$ is in GapP if there is a non-deterministic polynomial time turing machine M such that $f(x) = \#M(x) - \#\overline{M}(x)$. GapP is also the closure of #P under subtraction.

Most of this subsection is devoted to the proof of Theorem 7. First we give the non-deterministic Turing machines realising the upper bounds for Natural and Integer. After that we give reductions from respective counting versions of SAT to prove the lower bounds. The case of Rational is finally considered.

The Turing Machine $\mathcal{M}$ such that $\#\mathcal{M}(\mathcal{T},G) = [\![\mathcal{T}]\!](G)$. We describe a non-determinstic polynomial time turing machine $\mathcal{M}$ that takes as input a WTS $\mathcal{T}$ over Natural with weights given in binary, and a graph G. The number of accepting runs $\#\mathcal{M}(\mathcal{T},G) = [\![\mathcal{T}]\!](G)$. We assume the states, weights etc. are given by some standard encoding.

The turing machine $\mathcal{M}$ non-deterministically guesses a labeling of the vertices of G by the states of $\mathcal{T}$. Then it computes the product w of the weights of the tiles in the guessed tiling and writes it in binary (MSB on the left) in a different tape. Computing the product can be done in time polynomial in $|G|$ and $\log(k)$ where $k = \max\{x \mid x \text{ is a weight of some tile of } \mathcal{T}\}$.

Afterwards it enters a phase which will have exactly w different accepting branches. Simply decrementing the value while it is positive, and non-deterministically accepting at any step will have w accepting branches, but the running time is exponential. We want the machine to run in polynomial time. Hence we implement this phase similar to Example 5. It runs in $\mathcal{O}(|w|)$ steps as we detail below.

$\mathcal{M}$ scans w from left to right starting in some state q. While in state q and the current cell is labeled 0 it moves right. If in state q and the current cell is labelled 1 it moves right and non determistically stays in state q or enters one of the two special states q_0 or q_1. When it is in state q_0 or q_1 and the current cell is labelled with 0 or 1, it will move right and non deterministically chose either q_0 or q_1. Finally, When in state q_0 or q_1 and the current cell is *blank* (i.e., the scan of w is over), then $\mathcal{M}$ accepts. Thus if the ith bit from the right of w is labeled 1, then $\mathcal{M}$ can have 2^i accepting runs if it moved from state q to q_0 or q_1 when reading this bit. Switching from state q can occur at any 1-labelled cell, and hence $\mathcal{M}$ will have w many accepting runs.

The machine $\mathcal{M}$ non deterministically picks a labeling at first, and hence the total number of accepting runs $\#\mathcal{M}(\mathcal{T},G) = [\![\mathcal{T}]\!](G)$. With this we prove the #P upper bound for Natural.

The Turing Machine $\mathcal{M}'$ such that $\#\mathcal{M}'(\mathcal{T},G) - \#\overline{\mathcal{M}'}(\mathcal{T},G) = [\![\mathcal{T}]\!](G)$ This is similar to the machine $\mathcal{M}$ above. There are two differences. The machine $\mathcal{M}'$ still guesses a labeling of vertices of G with states of $\mathcal{T}$ over Integer and computes the weight w. If w is positive, it proceeds exactly as $\mathcal{M}$ does to produce w accepting runs. If the weight w is negative, the machine $\mathcal{M}'$ proceeds analogously but with states q', q'_0 and q'_1 instead. If the machine is in state q'_0 or q'_1 with current cell *blank* then it rejects instead of accepting. The second difference is for blocked runs (e.g., if the guessed labeling of vertices of G by states of $\mathcal{T}$ is not a valid tiling, or if at the end the machine is still in state q or q' with current cell *blank*). In such a case, $\mathcal{M}'$ will non-deterministically proceed to either accept or reject. Thus the net difference between accepting runs and rejecting runs is kept intact and $\#\mathcal{M}'(\mathcal{T},G) - \#\overline{\mathcal{M}'}(\mathcal{T},G) = [\![\mathcal{T}]\!](G)$. This proves the GapP upper bound for Integer.

Encoding a CNF formula φ in a grid G_φ. Given a CNF formula φ with n variables and m clauses, we encode it in an $n \times m$ grid with node labels $\{p, n, \star\}$. If the node (i,j) is labeled by p (resp. n) it means that the ith variable appears in jth clause positively (resp. negatively). The node (i,j) is labeled $\star$ if the ith variable does not occur in the jth clause.

A WTS $\mathcal{T}^{\#}$ over Natural for counting $\#\varphi$. Recall that $\#\varphi$ is the number of satisfying assignments for the formula φ. We assume input to the WTS $\mathcal{T}^{\#}$ is given as G_φ – a $\{p, n, \star\}$-labeled grid encoding a CNF formula.

A state of $\mathcal{T}^{\#}$ is a pair from $\{q_{\mathsf{true}}, q_{\mathsf{false}}\} \times \{q'_{\mathsf{true}}, q'_{\mathsf{false}}\}$. The first part of a state indicates a truth assignment with q_{true} and q_{false}. The allowed tiles make sure that in this part the truth assignment remains the same along a row. The second part of a state indicates with q'_{true} and q'_{false} the partial evaluation of the formula. A p-labeled node which is assigned q_{true} from the first part, and an n-labeled node which is assigned q_{false} from the first part gets the value q'_{true} in the second part of the state (call this condition A for future reference). Further all the successor nodes in the column of the q'_{true} labeled node also gets the value q'_{true}, except for the nodes in the last row. For the nodes in the last row, it gets the value q'_{true} if the left neighbour is labeled q'_{true} (assume this is satisfied if the left neighbour does not exist), and a) if it satisfies condition A or b) if the node above is labeled q'_{true}. Otherwise the nodes get the value q'_{false}. The second part of a state labeling a node (n,j) in the last row indicates the evaluation of the prefix of the formula until the jth clause.

The tiles capture the description above. The weight of all tiles is 1, except for the tile labeling the last node (n,m). If it is labeled $(-, q'_{\mathsf{true}})$ then the weight is 1, otherwise it is 0. The value $[\![\mathcal{T}^{\#}]\!](G_\varphi) = \#\varphi$, the number of satisfying assignments.

This proves the #P lower bound for Natural. As alluded to earlier, the permanent computation (Example 4) gives an alternate lower bound proof.

A WTS $\mathcal{T}^{\mathsf{gap}}$ over Integer for counting $\#\varphi_1 - \#\varphi_2$. We will reduce the GapP-complete problem of computing $\#\varphi_1 - \#\varphi_2$, where φ_1 and φ_2 are input CNF formulas on the same set of n variables with m_1 and m_2 clauses respectively. We represent the input in an $n \times (m_1 + m_2)$ grid by putting G_{φ_1} and G_{φ_2} side by side. The node labels contain a special tag $i \in \{1,2\}$ to indicate that it comes from G_{φ_i}. The WTS $\mathcal{T}^{\mathsf{gap}}$ will ensure that rows are of the form 1^*2^* and columns are of the form 1^* or 2^*. In a run it evaluates either φ_1 or φ_2 similar to $\mathcal{T}^{\#}$. If it is evaluating φ_i all nodes with the tag $3-i$ gets a special state q_{skip}. The weight of all tiles is 1, except for the tile labeling the nodes (n, m_1) and (n, m_1+m_2). If the node (n, m_1) is labeled $(-, q'_{\mathsf{true}})$ or q_{skip} then the weight is 1, otherwise it is 0. If the node (n, m_1+m_2) is labeled $(-, q'_{\mathsf{true}})$ (resp. q_{skip}) then the weight is -1 (resp. 1), otherwise it is 0.

Rational. We will use counting reduction from Rational (resp. non-negative Rational) to the evaluation problem over Integer (resp. Natural) in order to prove the upper bounds. First we will transform an input $(\mathcal{T}, G)$ of the evaluation problem over Rational (resp. non-negative Rational) to an input $(\mathcal{T}', G,)$ over Integer (resp. Natural). In $\mathcal{T}'$ we will multiply the weight of a tile by ℓ - the lcm of the denominators appearing in the weights of any tile of $\mathcal{T}$. The multiplication can be performed in time polynomial. Now $\mathcal{T}'$ is a WTS over Integer (resp. Natural), and following the GapP procedure (resp. #P procedure) we compute $[\![\mathcal{T}']\!](G)$. Now, we transform the output back to the required output over Rational (resp. non-negative Rational) by dividing with $\ell^{|V_G|}$. That is, $\mathsf{Eval}(G, \mathcal{A}) = \frac{\mathsf{Eval}(G, \mathcal{A}')}{\ell^{|V_G|}}$.

Notice that we allow the weights to be given in binary. The lcm ℓ and $\ell^{|V_G|}$ can be computed in polynomial time. The counting reduction is hence polynomial. This proves the upper bounds.

The GapP-hardness (resp. #P-hardness) follows because Integer (resp. Natural) is a special case of Rational (resp. non-negative Rational).

3.2 Boolean semiring

Note that the evaluation problem Eval over Boolean is in fact the classical Membership problem (denoted Membership) and is indeed a decision problem. We can check in NP whether the value is 1 (witnessed by the NP machine $\mathcal{M}$, if the input is assumed to be over Boolean then $\times$ serve as $\wedge$). It is also NP-hard by a simple reduction from CNF SAT (witnessed by $\mathcal{T}^{\#}$ interpreted over Boolean).

▶ **Theorem 8.** *Membership is NP-complete.*

3.3 Tropical semirings

▶ **Theorem 9.** *We assume the weights are given in unary. The evaluation problem over any tropical semiring is FP$^{\mathrm{NP}[log]}$-complete.*

FP$^{\mathrm{NP}[log]}$ is the class of functions computable by a polynomial time turing machine with logarithmically many queries to NP.

Proof. We will prove the upper bound for max-plus-$\mathbb{Z}$. The case of max-plus-$\mathbb{N}$ is subsumed. The cases of min-plus-$\mathbb{N}$ and min-plus-$\mathbb{Z}$ are analogous.

Let k be the maximal constant and ℓ be the minimal constant (other than $+/-\infty$) appearing in the WTS $\mathcal{A}$. The maximum possible weight of a run is $n \times k$ and the minimum is $n \times \ell$ where n is the number of vertices in the input graph. We will do a binary search in the set $W = \{n \times \ell, \dots, -1, 0, 1, \dots, n \times k\}$ checking if $[\![\mathcal{A}]\!](G) \geq s$ to find the value of $[\![\mathcal{A}]\!](G)$. In each iteration of the binary search, we make an oracle call to the NP machine for $[\![\mathcal{A}]\!](G) \geq s$. The number of NP oracle queries is $\mathcal{O}(\log(n \times k))$ which is only logarithmic in the input size. Recall that the weights are encoded in unary.

Finding the clique number is an FP$^{\mathrm{NP}[log]}$-complete problem [19]. From Example 3, the lower bound follows. ◀

4 Efficient evaluation for bounded tree-width graphs

In this section, we show that the problem Eval can be solved efficiently when restricted to graphs of bounded tree-width (the bound is not part of the input). By efficient, we mean time polynomial wrt. the WTS $\mathcal{T}$ and linear wrt. the graph G (see Theorems 11 and 13 below). Bounded tree-width covers many graphs used to model behaviours of concurrent or infinite-state systems. For example, it is well-known that words and trees have tree-width 1, nested words used for pushdown systems have tree-width 2, Mazurkiewicz traces describing behaviours of concurrent asynchronous systems with rendez-vous, and most decidable under-approximations of Turing complete models such as multi-pushdown automata, message passing automata with unbounded FIFO channels, etc. [21, 9, 4]. We start by explaining our results for bounded path-width since this is technically simpler. Then we explain how this is extended to bounded tree-width.

4.1 Bounded path-width evaluation

A path decomposition. of a (Γ,Σ)-graph $G=(V,(E_\gamma)_{\gamma\in\Gamma},\lambda)$, is a sequence $V_1,\dots,V_n$ of nonempty subsets of vertices satisfying:

1. for all $v\in V$, we have $v\in V_i$ for some $1\le i\le n$,
2. for all $(u,v)\in\bigcup_{\gamma\in\Gamma}E_\gamma$, we have $u,v\in V_i$ for some $1\le i\le n$,
3. for all $1\le i\le j\le k\le n$, we have $V_i\cap V_k\subseteq V_j$.

The width of the path decomposition is $\max\{|V_i|-1\mid 1\le i\le n\}$. The path-width of a graph G is the least k such that G admits a path decomposition of width k.

Words have path-width 1, but trees, nested words, grids have unbounded path-width.

We present below an equivalent definition of path-width which will be convenient to solve the evaluation problem on graphs with bounded path-width. Let $[k]=\{0,1,\dots,k\}$. Graphs over (Γ,Σ) of path-width at most k can be described with words over the alphabet

$$\Omega_k=\{(i,a)\mid i\in[k],\ a\in\Sigma\}\cup\{\mathsf{Forget}_i\mid i\in[k]\}\cup\{\mathsf{Add}^\gamma_{i,j}\mid i,j\in[k],\ \gamma\in\Gamma\}$$

The semantics of a word $\tau\in\Omega_k^*$ is a colored graph $\{\!|\tau|\!\}=(G_\tau,\chi_\tau)$ where G_τ is a (Γ,Σ)-labeled graph and $\chi_\tau\colon[k]\to V$ is a partial injective function coloring some vertices of G_τ. We say that a color $i\in[k]$ is *active* in τ if it is in the domain of χ_τ. The semantics is defined by induction on the length of τ. The semantics of the empty word $\tau=\varepsilon$ is the empty graph. Assuming that $\{\!|\tau|\!\}=(V,(E_\gamma)_{\gamma\in\Gamma},\lambda,\chi)$, we define the effect of appending a new letter to τ: (i,a) adds a new a-labeled vertex with color i, provided i is not active in τ, Forget_i removes color i from the domain of the color map, and $\mathsf{Add}^\alpha_{i,j}$ adds an α-labeled edge between the vertices colored i and j (if such vertices exist, i.e., if i,j are active in τ).

We say that a word τ over Ω_k is *well-formed* if the following conditions are satisfied:

1. if $\tau'\cdot(i,a)$ is a prefix of τ then i is not active in τ',
2. if $\tau'\cdot\mathsf{Forget}_i$ is a prefix of τ then i is active in τ',
3. if $\tau'\cdot\mathsf{Add}^\gamma_{i,j}$ is a prefix of τ then i,j are active in τ' and the edge labeled γ was not already added in τ' between $\chi_{\tau'}(i)$ and $\chi_{\tau'}(j)$.

In the following, a well-formed word over Ω_k is called a k-word. The set $\mathsf{W}_k\subseteq\Omega_k^*$ of k-words is clearly regular.

▶ **Lemma 10.** **1.** *Given a path decomposition $V_1,\dots,V_N$ of width at most k of a (Γ,Σ)-graph G, we can construct in linear time wrt. $|G|$ a k-word τ such that $\{\!|\tau|\!\}=(G,\varnothing)$.*

2. *Given a k–word τ, we can construct a path decomposition of width at most k of the graph G_τ defined by τ: $\{\!|\tau|\!\}=(G_\tau,\chi_\tau)$.*

Proof. 1. We construct by induction a sequence of k-words τ_ℓ for $0 \le \ell \le N$ such that $\{\!|\tau_\ell|\!\} = (G_\ell, \chi_\ell)$ where G_ℓ is the subgraph of $G = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda)$ induced by the vertices $V_1 \cup \cdots \cup V_\ell$, and $\chi_\ell([k]) = V_\ell \cap V_{\ell+1}$ (with $V_0 = V_{N+1} = \emptyset$). We let $\tau_0 = \epsilon$.

Let now $0 \le \ell < N$ and assume that τ_ℓ has been constructed. Let $C_\ell = \mathsf{dom}(\chi_\ell) \subseteq [k]$ be the active colors in τ_ℓ. By induction, we know that $|C_\ell| = |V_{\ell+1} \cap V_\ell|$. Let $V_{\ell+1} \setminus V_\ell = \{u_1, \ldots, u_m\}$. Since the decomposition is of width at most k, we have $|V_{\ell+1}| \le 1 + k$ and we find $i_1 < \cdots < i_m$ available colors in $[k] \setminus C_\ell$. We define $\tau'_{\ell+1} = \tau_\ell \cdot (i_1, \lambda(u_1)) \cdots (i_m, \lambda(u_m))$. Let $D = \{i_1, \ldots, i_m\}$ and let $\{\!|\tau'_{\ell+1}|\!\} = (G', \chi')$. We have $\mathsf{dom}(\chi') = C_\ell \cup D$, $\chi'(C_\ell) = V_{\ell+1} \cap V_\ell$ and $\chi'(D) = V_{\ell+1} \setminus V_\ell$. For each $\gamma \in \Gamma$, $i \in C_\ell \cup D$ and $j \in D$ such that $(\chi'(i), \chi'(j)) \in E_\gamma$ (resp. $(\chi'(j), \chi'(i)) \in E_\gamma$), we append $\mathsf{Add}^\gamma_{i,j}$ (resp. $\mathsf{Add}^\gamma_{j,i}$) to the word $\tau'_{\ell+1}$. We obtain a k-word $\tau''_{\ell+1}$ which defines the subgraph $G_{\ell+1}$ of G induced by $V_1 \cup \cdots \cup V_{\ell+1}$. Notice that, from the third condition of a path decomposition, we have $V_{\ell+1} \setminus V_\ell = V_{\ell+1} \setminus (V_1 \cup \cdots \cup V_\ell)$ and the edges in $G_{\ell+1}$ which were not already in G_ℓ are between some vertex in $V_{\ell+1} \setminus V_\ell$ and some vertex in $V_{\ell+1}$. Finally, for each $i \in C_\ell \cup D$ such that $\chi'(i) \notin V_{\ell+2}$, we append Forget_i to the word $\tau''_{\ell+1}$. We obtain the k-word $\tau_{\ell+1}$ satisfying our invariant.

Finally, from the invariant we deduce that $\{\!|\tau_N|\!\} = (G, \emptyset)$, which concludes the first part of the proof.

2. Let τ be a k-word and $n = |\tau|$ be its length. For $0 \le \ell \le n$, let τ_ℓ be the prefix of τ of length ℓ. Let $\{\!|\tau_\ell|\!\} = (G_\ell, \chi_\ell)$ and $V_\ell = \chi_\ell([k])$ be the subset of vertices which are colored in $\{\!|\tau_\ell|\!\}$. We show that $V_1, \ldots, V_n$ is a path decomposition of $G = G_n = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda)$.

Let $u \in V$ be a vertex of G. For some $1 \le \ell \le n$, we have $\tau_\ell = \tau_{\ell-1} \cdot (i, a)$ with $\chi_\ell(i) = u \in V_\ell$. This proves that the first condition of a path decomposition is satisfied.

Let $(u, v) \in E_\gamma$ for some $\gamma \in \Gamma$. For some $1 < \ell < n$, we have $\tau_{\ell+1} = \tau_\ell \cdot \mathsf{Add}^\gamma_{i,j}$ with $\chi_\ell(i) = u$ and $\chi_\ell(j) = v$. We deduce that $u, v \in V_\ell$, which proves that the second condition of a path decomposition is satisfied.

For the third condition, let $1 \le i \le j \le m \le n$ and $u \in V_i \cap V_m$. We deduce that for some $\ell \in [k]$, we have $u = \chi_i(\ell) = \chi_m(\ell)$ and that color ℓ was not forgotten between τ_i and τ_m. Therefore, $u = \chi_j(\ell) \in V_j$ as desired. ◀

The problem k-PW-FVal is to compute $[\![\mathcal{T}]\!](G)$, given a WTS $\mathcal{T}$ and a (Γ, Σ)-graph G of path-width at most k.

▶ **Theorem 11.** *The problem k-PW-FVal can be solved in linear time wrt. the input graph G and polynomial time wrt. the input WTS $\mathcal{T}$.*

Proof. The evaluation algorithm for bounded path-width graphs proceeds in three steps:

1. From the input graph G, which is assumed to be of path-width at most k, we compute in linear time a path decomposition $V_1, \ldots, V_n$ using Bodlaender's algorithm [3]. Then, using Lemma 10, we compute in linear time a k-word τ such that $\{\!|\tau|\!\} = (G, \emptyset)$.
2. By Lemma 12 below, we construct in time polynomial in $\mathcal{T}$ a weighted word automaton $\mathcal{B}_k$ which is equivalent to $\mathcal{T}$ on graphs of path-width at most k.
3. We compute $[\![\mathcal{B}_k]\!](\tau)$. It is well-known that the value of a weighted word automaton $\mathcal{B}$ on a given word w can be computed in time $\mathcal{O}(|\mathcal{B}| \cdot |w|)$ assuming that sum and product in the semiring take constant time. ◀

A weighted word automaton over alphabet Σ is usually given as a tuple $\mathcal{B} = (Q, T, I, F, \mathsf{wgt})$ where $I, F \subseteq Q$ are the subsets of initial and final states, $T \subseteq Q \times \Sigma \times Q$ defines the transitions and $\mathsf{wgt}\colon T \to S$ gives weights to transitions. This is an equivalent representation of a WTS over $(\{\to\}, \Sigma)$.

Table 1 Transitions of the weighted word automaton $\mathcal{B}_k$.

$\delta \xrightarrow{(i,a)} \delta'$	if $i \notin \mathsf{dom}(\delta)$. Then, $\mathsf{dom}(\delta') = \mathsf{dom}(\delta) \cup \{i\}$, $\delta'(j) = \delta(j)$ for all $j \in \mathsf{dom}(\delta)$, and $\delta'(i) = (f_\varnothing, q, a, f_\varnothing)$ for some $q \in Q$. The weight of this transition is $1_\mathbb{S}$.
$\delta \xrightarrow{\mathsf{Forget}_i} \delta'$	if $i \in \mathsf{dom}(\delta)$. Then δ' is the restriction of δ to $\mathsf{dom}(\delta') = \mathsf{dom}(\delta) \setminus \{i\}$. The weight of this transition is $\mathsf{wgt}(\delta(i))$.
$\delta \xrightarrow{\mathsf{Add}^\gamma_{i,j}} \delta'$	if $i, j \in \mathsf{dom}(\delta)$, $i \neq j$, $\gamma \notin \mathsf{dom}(f_{\text{out}}(i))$ and $\gamma \notin \mathsf{dom}(f_{\text{in}}(j))$. Then, $\mathsf{dom}(\delta') = \mathsf{dom}(\delta)$, $\delta'(\ell) = \delta(\ell)$ for all $\ell \in \mathsf{dom}(\delta) \setminus \{i,j\}$, $\delta'(i) = (f_{\text{in}}(i), q(i), a(i), f_{\text{out}}(i) \cup [\gamma \mapsto q(j)])$, $\delta'(j) = (f_{\text{in}}(j) \cup [\gamma \mapsto q(i)], q(j), a(j), f_{\text{out}}(j))$. The weight of this transition is $1_\mathbb{S}$.

▶ **Lemma 12.** *Given a WTS $\mathcal{T}$ over (Γ, Σ)-graphs and $k > 0$, we can compute in polynomial time wrt. $\mathcal{T}$, a weighted word automaton $\mathcal{B}_k$ which is equivalent to $\mathcal{T}$ over graphs of path width at most k. That is, for all k-words τ with $\{\!|\tau|\!\} = (G, \varnothing)$, we have $[\![\mathcal{T}]\!](G) = [\![\mathcal{B}_k]\!](\tau)$.*

Proof. Let $\mathcal{T} = (Q, \Delta, \mathsf{wgt})$ be a WTS over (Γ, Σ)-graphs. By adding tiles with weight $0_\mathbb{S}$, we may assume wlog that Δ contains all possible tiles. Fix $k \geq 1$.

A state of $\mathcal{B}_k$ is a partial map $\delta \colon [k] \to \Delta$. When reading a k-word τ with $\{\!|\tau|\!\} = (G, \chi)$, the automaton will guess a labelling $\rho \colon V \to Q$ of vertices of G with states of $\mathcal{T}$ and will reach a state δ satisfying the following two conditions:

1. $\mathsf{dom}(\delta) = \mathsf{dom}(\chi) \subseteq [k]$ is the set of active colors,
2. for each active color $i \in \mathsf{dom}(\chi)$, $\delta(i) = (f_{\text{in}}(i), q(i), a(i), f_{\text{out}}(i)) = \mathsf{tile}_\rho(\chi(i))$ is the current ρ-tile at vertex $\chi(i)$ in G.

The only initial state is the empty map $\delta_\varnothing$ with $\mathsf{dom}(\delta_\varnothing) = \varnothing$. This is also the only final state, which is reached on a k-word τ if all colors have been forgotten: $\{\!|\tau|\!\} = (G, \chi_\varnothing)$.

Transitions of the word automaton $\mathcal{B}_k$ are given in Table 1. As above, we write $\delta(i) = (f_{\text{in}}(i), q(i), a(i), f_{\text{out}}(i))$ and $\delta'(i) = (f'_{\text{in}}(i), q'(i), a'(i), f'_{\text{out}}(i))$.

The number of partial maps from A to B is $(1 + |B|)^{|A|}$. Hence, the number of states of $\mathcal{B}_k$ is $(1 + |\Delta|)^{1+k}$. In a tile $(f_{\text{in}}, q, a, f_{\text{out}}) \in \Delta$, both f_{in} and f_{out} can be seen as partial maps from Γ to Q. Hence, $|\Delta| = (1 + |Q|)^{2|\Gamma|} \cdot |Q| \cdot |\Sigma|$. Also, $|\Omega_k| = (1 + k)(|\Sigma| + 1) + (1 + k)^2 |\Gamma|$. We deduce that, if Σ, Γ, k are fixed, the automaton $\mathcal{B}_k$ can be constructed in polynomial time wrt. the given WTS $\mathcal{T}$. ◀

4.2 Bounded tree-width evaluation

We extend the efficient evaluation of WTS for graphs of bounded path-width to graphs of bounded tree-width, which is a larger class of graphs. For instance, nested words may have unbounded path-width but their tree-width is at most 2. As for path-width, tree-width can be defined via tree decompositions: instead of a sequence of subsets of vertices, we use a tree of subsets of vertices. Since we will use weighted tree automata to achieve the efficient evaluation over graphs of bounded tree-width, we define directly tree terms. These are similar to k-words, with an additional binary union $\oplus$.

Tree terms (TTs). form an algebra to define labeled graphs. With $a \in \Sigma$, $\gamma \in \Gamma$ and $i, j \in [k] = \{0, 1, \ldots, k\}$, the syntax of k-TTs over (Γ, Σ) is given by

$$\tau ::= (i, a) \mid \mathsf{Add}^\gamma_{i,j}\, \tau \mid \mathsf{Forget}_i\, \tau \mid \tau \oplus \tau$$

Each k-TT represents a colored graph $\{\!|\tau|\!\} = (G_\tau, \chi_\tau)$ where G_τ is a (Γ, Σ)-labeled graph and $\chi_\tau \colon [k] \to V$ is a partial injective function coloring some vertices of G_τ. Colors in $\mathsf{dom}\chi_\tau$ are said to be active in τ. The semantics is defined as for k−words: a leaf (i,a) creates a graph with a single a-labeled vertex with color i, Forget_i removes color i from the domain of the color map, and $\mathsf{Add}^\alpha_{i,j}$ adds an α-labeled edge between the vertices colored i and j (if such vertices exist). Formally, if $\{\!|\tau|\!\} = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda, \chi)$ then

- $\{\!|\,\mathsf{Add}^\alpha_{i,j}\,\tau|\!\} = (V, (E'_\gamma)_{\gamma\in\Gamma}, \lambda, \chi)$ with $E'_\gamma = E_\gamma$ if $\gamma \neq \alpha$ and

$$E'_\alpha = \begin{cases} E_\alpha & \text{if } \{i,j\} \nsubseteq \mathsf{dom}(\chi) \\ E_\alpha \cup \{(\chi(i), \chi(j))\} & \text{otherwise.} \end{cases}$$

- $\{\!|\,\mathsf{Forget}_i\,\tau|\!\} = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda, \chi')$ with $\mathsf{dom}(\chi') = \mathsf{dom}(\chi) \setminus \{i\}$ and $\chi'(j) = \chi(j)$ for all $j \in \mathsf{dom}(\chi')$.

The main difference with k-words is $\oplus$ which takes the union of the two graphs, merging vertices with the same colors, if any.

- Formally, consider $\tau' \oplus \tau''$ with $\{\!|\tau'|\!\} = (G', \chi') = (V', (E'_\gamma)_{\gamma\in\Gamma}, \lambda', \chi')$ and $\{\!|\tau''|\!\} = (G'', \chi'') = (V'', (E''_\gamma)_{\gamma\in\Gamma}, \lambda'', \chi'')$. Let $I = \mathsf{dom}(\chi') \cap \mathsf{dom}(\chi'')$ be the set of colors that are defined in both graphs. Wlog, we may assume that $V' \cap V'' = \chi'(I) = \chi''(I)$ and $\chi'(i) = \chi''(i)$ for all $i \in I$, i.e., we may rename the vertices so that the shared colors define the shared vertices. The union $\tau' \oplus \tau''$ is well-defined only if the shared vertices have the same labels: $\lambda'(\chi'(i)) = \lambda''(\chi''(i))$ for all $i \in I$. Then, $\{\!|\tau' \oplus \tau''|\!\} = (G' \cup G'', \chi' \cup \chi'') = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda, \chi)$ where $V = V' \cup V''$, $\lambda = \lambda' \cup \lambda''$, and $E_\gamma = E'_\gamma \cup E''_\gamma$ for all $\gamma \in \Gamma$.

The tree-width of a nonempty graph G is the least $k \geq 1$ such that $G = G_\tau$ for some k-TT τ.

Trees have tree-width 1, and as a special case, words also have tree-width 1. Nested words have tree-width (at most) 2 [21]. They are words with an additional binary relation from pushes to matching pops, which are used to represent behaviours of pushdown automata. On the other end, grids as used for instance in Example 4, have unbounded tree-width. More precisely, an $n \times n$ grid has tree-width n.

We will focus on a regular subset of terms which ensures that the semantics is well-defined and that the k-TTs do not contain redundant operations such as $\mathsf{Add}^\gamma_{i,j}\,\mathsf{Add}^\gamma_{i,j}\,\tau$ or $\mathsf{Add}^\gamma_{i,j}\,\tau_1 \oplus \mathsf{Add}^\gamma_{i,j}\,\tau_2$. A k-TT is *well-formed* if the following are satisfied:

1. if $\mathsf{Forget}_i\,\tau'$ is a subterm of τ then i is active in τ',
2. if $\mathsf{Add}^\gamma_{i,j}\,\tau'$ is a subterm of τ then i,j are active in τ' and the edge γ was not already added in τ' between $\chi_{\tau'}(i)$ and $\chi_{\tau'}(j)$.
3. if $\tau' \oplus \tau''$ is a subterm of τ then for all i,j that are active in both τ' and τ'', the vertices $\chi_{\tau'}(i)$ and $\chi_{\tau''}(i)$ have the same label from Σ, and we do not already have a γ-edge both between $(\chi_{\tau'}(i), \chi_{\tau'}(j))$ and $(\chi_{\tau''}(i), \chi_{\tau''}(j))$.

The problem k-TW-FVal is to compute $[\![\mathcal{T}]\!](G)$, given a WTS $\mathcal{T}$ and a (Γ, Σ)-graph G of tree-width at most k.

▶ **Theorem 13.** *The problem k-TW-FVal can be solved in linear time wrt. the input graph G and polynomial time wrt. the input WTS $\mathcal{T}$.*

Proof. The proof follows the same three steps as for Theorem 11 using tree terms instead of k-words and weighted tree automata instead of weighted word automata.

1. From the input graph G, which is assumed to be of tree-width at most k, we compute in linear time a tree decomposition using Bodlaender's algorithm [3]. Then, similarly to Lemma 10, we compute in linear time a well-formed k-TT τ such that $\{\!|\tau|\!\} = (G, \emptyset)$. In particular, $|\tau| = \mathcal{O}(|G|)$.

Algorithm 1 Evaluation algorithm for a weighted tree automaton $\mathcal{B} = (Q, T, F, \mathsf{wgt})$.

```
function MAIN($\tau$: term): value from $S$                    ▷ Computes $[\![\mathcal{B}]\!](\tau)$
    $\mathsf{val} \leftarrow$ TREEEVAL($\tau$); $x \leftarrow 0_{\mathbb{S}}$
    for all $q \in F$ do $x \leftarrow x + \mathsf{val}[q]$ end for
    return $x$
end function
function TREEEVAL($\tau$: term): array indexed by $Q$ of values from $S$
    ▷ TREEEVAL($\tau$)[$q$] is the sum of the weights of the runs of $\mathcal{B}$ on $\tau$ reaching state $q$.
    match $\tau$ with
        Leaf $a$:
            for all $q \in Q$ do $\mathsf{val}[q] \leftarrow \mathsf{wgt}(\perp, a, q)$ end for
        Unary $a(\tau_1)$:
            $\mathsf{val}_1 \leftarrow$ TREEEVAL($\tau_1$)
            for all $q \in Q$ do $\mathsf{val}[q] \leftarrow 0_{\mathbb{S}}$ end for
            for all $(q_1, a, q) \in T$ do $\mathsf{val}[q] \leftarrow \mathsf{val}[q] + \mathsf{val}_1[q_1] \times \mathsf{wgt}(q_1, a, q)$ end for
        Binary $a(\tau_1, \tau_2)$:
            $\mathsf{val}_1 \leftarrow$ TREEEVAL($\tau_1$); $\mathsf{val}_2 \leftarrow$ TREEEVAL($\tau_2$)
            for all $q \in Q$ do $\mathsf{val}[q] \leftarrow 0_{\mathbb{S}}$ end for
            for all $(q_1, q_2, a, q) \in T$ do
                $\mathsf{val}[q] \leftarrow \mathsf{val}[q] + \mathsf{val}_1[q_1] \times \mathsf{wgt}(q_1, q_2, a, q) \times \mathsf{val}_2[q_2]$
            end for
    end match
    return val
end function
```

2. Using Lemma 14 below, from the WTS $\mathcal{T}$ we construct in polynomial time an equivalent weighted tree automaton $\mathcal{B}_k$ on graphs of tree-width at most k: $[\![\mathcal{T}]\!](G) = [\![\mathcal{B}_k]\!](\tau)$.
3. We compute $[\![\mathcal{B}_k]\!](\tau)$ with Algorithm 1. The main complexity comes from the call TREEEVAL. Executing the body of this function (without the recursive calls) takes time $\mathcal{O}(|\mathcal{B}_k|)$. Hence, the overall time complexity of this evaluation is $\mathcal{O}(|\tau| \cdot |\mathcal{B}_k|)$. ◀

A weighted (binary) tree automaton over alphabet Σ is usually given as a tuple $\mathcal{B} = (Q, T, F, \mathsf{wgt})$ where $F \subseteq Q$ is the subset of accepting states, $T \subseteq (\{\perp\} \cup Q \cup Q^2) \times \Sigma \times Q$ defines the bottom-up transitions and $\mathsf{wgt}\colon T \to S$ gives weights to transitions. This is an equivalent representation of a WTS over $(\{\nearrow, \nwarrow\}, \Sigma)$.

▶ **Lemma 14.** *Given a WTS $\mathcal{T}$ over (Γ, Σ)-graphs and $k > 0$, we can compute in polynomial time wrt. $\mathcal{T}$, a weighted tree automaton $\mathcal{B}_k$ which is equivalent to $\mathcal{T}$ over graphs of tree-width at most k. Here, equivalent means that for all well-formed k-TTs τ with $\{\!|\tau|\!\} = (G, \varnothing)$, we have $[\![\mathcal{T}]\!](G) = [\![\mathcal{B}_k]\!](\tau)$.*

5 Discussions and conclusions

Connections with CSP. The quantitative versions of the constraint satisfaction problem (CSP) are closely related to the evaluation problem for weighted tiling systems and graphs. Classic (boolean) CSPs ask for the existence of a solution of a set of constraints, as non-deterministic automata ask for the existence of an accepting run. In the *valued*-CSP (see e.g. [20]), weights (costs) are assigned to each constraint depending on how the constraint is fulfilled, these weights are summed over all constraints and the aim is to minimize this total cost. This corresponds to our evaluation problem in the min-plus tropical semiring.

The weighted counting CSP (weighted #CSP) is defined similarly but uses a $(+, \times)$-semiring such as $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, ..., (see e.g. [7, 8]). The cost of a solution is the product of the weights over all constraints and the value of the weighted #CSP is the sum over all solutions. Counting CSP (#CSP) is obtained with semiring Natural when functions in the language only take values 0 or 1, thus counting the number of solutions of the classic CSP.

One of the main problems in CSP is to determine conditions under which the problems are tractable (polynomial time). Feder and Vardi conjectured [16] that, depending on the constraint language Γ, problems in CSP(Γ) are either in P or NP-complete. The dichotomy conjecture extends to #CSP(Γ), saying that such counting problems are either in FP or #P-complete, see e.g. [6, 15]. In this paper, we show that for WTS, the evalution problem is #P-complete (Theorem 7).

Most often the *non-uniform* complexity is considered, meaning that the language (for us the WTS) is not part of the input and the complexity only depends on the instance (for us the input graph). One such structural restriction is when the constraint graph of the instance has bounded tree-width. This is indeed related to our efficient evaluation described in Section 4. Our approach is different though since we reduce WTS to weighted word/tree automata and obtain a complexity linear in the input graph.

As future work, we plan to investigate more closely the relationship between weighted #CSP and the evaluation problem for WTS. In particular, it would be interesting to see whether results on approximate computation which are widely studied for quantitative CSP can be transfered to weighted tiling systems.

On the generality of the model. The model of WGA [10] additionally has occurrence constraints (boolean combinations of constraints of the form $\#\mathsf{tile} \geq n$, where $\mathsf{tile} \in \Delta$ and $n \in \mathbb{N}$). A run is valid only if the occurrence constraints are satisfied. We could allow these constraints as well, without compromising the complexity upper bounds. In fact, we can allow more expressive quantifier-free Presburger constraints on the tiles (e.g., $\#\mathsf{tile}_1 + \#\mathsf{tile}_2 = \#\mathsf{tile}_3$). The NP machine witnessing the upper bounds can compute the Parikh vector of the tiles used in a guessed run, and check in polynomial time whether the constraints are satisfied.

Variants. The evaluation problem Eval is a function problem. The decision variants correspond to threshold languages such as, is the computed weight $\{>, \geq, <, \leq, =, \neq\}$ s, s being a threshold. There are further variants depending on whether the threshold s is part of the input or is fixed. The complexity depend on the semiring as well as on the value of the threshold when it is fixed.

Conclusion. We have given tight complexity bounds for the evaluation problem for various semirings. Our complexity upper bounds allows weights to be given in binary for problems over $(+, \times)$-semirings. However for tropical semirings the weights are assumed to be given in unary. While our upper bounds hold for arbitrary graphs, lower bounds are given uniformly for pictures (grid graphs). Further if we assume that the input graph does not have unbounded grid as a minor (bounded tree-width), then we provide efficient evaluation algorithm.

References

1 C. Aiswarya, Paul Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 2012.

2 C. Aiswarya, Paul Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *(ATVA*, volume 8837 of *Lecture Notes in Computer Science*, pages 1–17, Sidney, Australia, November 2014. Springer. `doi:10.1007/978-3-319-11936-6_1`.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, December 1996.

4 Benedikt Bollig and Paul Gastin. Non-sequential theory of distributed systems. *CoRR*, abs/1904.06942, 2019. URL: `http://arxiv.org/abs/1904.06942`.

5 Benedikt Bollig and Ingmar Meinecke. Weighted distributed systems and their logics. In *International Symposium on Logical Foundations of Computer Science, LFCS 2007*, volume 4514 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2007.

6 Andrei A. Bulatov and Víctor Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. *Inf. Comput.*, 205(5):651–678, 2007.

7 Andrei A. Bulatov, Martin E. Dyer, Leslie Ann Goldberg, Markus Jalsenius, Mark Jerrum, and David Richerby. The complexity of weighted and unweighted #csp. *J. Comput. Syst. Sci.*, 78(2):681–688, 2012.

8 Clément Carbonnel and Martin C. Cooper. Tractability in constraint satisfaction problems: a survey. *Constraints An Int. J.*, 21(2):115–144, 2016.

9 Aiswarya Cyriac. *Verification of communicating recursive programs via split-width. (Vérification de programmes récursifs et communicants via split-width)*. PhD thesis, École normale supérieure de Cachan, France, 2014. URL: `https://tel.archives-ouvertes.fr/tel-01015561`.

10 Manfred Droste and Stefan Dück. Weighted automata and logics on graphs. In *Mathematical Foundations of Computer Science (MFCS'15)*, volume 9234 of *Lecture Notes in Computer Science*, pages 192–204. Springer, 2015.

11 Manfred Droste and Paul Gastin. The kleene-schützenberger theorem for formal power series in partially commuting variables. *Inf. Comput.*, 153(1):47–80, 1999.

12 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. Springer Berlin Heidelberg, 2009.

13 Manfred Droste, Christian Pech, and Heiko Vogler. A Kleene theorem for weighted tree automata. *Theory Comput. Syst.*, 38(1):1–38, 2005.

14 Manfred Droste and Heiko Vogler. Weighted tree automata and weighted logics. *Theor. Comput. Sci.*, 366(3):228–247, 2006.

15 Martin E. Dyer, Leslie Ann Goldberg, and Mark Jerrum. The complexity of weighted boolean #csp. *SIAM J. Comput.*, 38(5):1970–1986, 2009.

16 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

17 Ina Fichtner. Weighted picture automata and weighted logics. *Theory Comput. Syst.*, 48(1):48–78, 2011.

18 Paul Gastin and Benjamin Monmege. Adding pebbles to weighted automata: Easy specification & efficient evaluation. *Theoretical Computer Science*, 534:24–44, May 2014.

19 Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988. `doi:10.1016/0022-0000(88)90039-6`.

20 Andrei A. Krokhin and Stanislav Zivny. The complexity of valued csps. In Andrei A. Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 233–266. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

21 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 283–294. ACM, 2011.

22 Christian Mathissen. Weighted logics for nested words and algebraic formal power series. *Logical Methods in Computer Science*, 6(1), February 2010.

23 Ingmar Meinecke. Weighted logics for traces. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *First International Computer Science Symposium in Russia, CSR 2006*, volume 3967 of *Lecture Notes in Computer Science*, pages 235–246. Springer, 2006.

24 Wolfgang Thomas. On logics, tilings, and automata. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez Artalejo, editors, *Automata, Languages and Programming*, pages 441–454, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

25 L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.