*ACM Communications in Computer Algebra, Vol. 54, No. 2, Issue 212, June 2020*

# ISSAC 2020 Poster Abstracts

Communicated by Jonathan Hauenstein

## The Sage Package `comb_walks` for Walks in the Quarter Plane

Antonio Jiménez-Pastor*
Johannes Kepler University Linz, Doctoral Program Computational Mathematics, DK15
ajpastor@risc.uni-linz.ac.at

Alin Bostan, Frédéric Chyzak, Pierre Lairez†
INRIA, France
alin.bostan@inria.fr, frederic.chyzak@inria.fr, pierre.lairez@inria.fr

**Abstract**

We present in this extended abstract a new software designed to work with generating functions that count walks in the quarter plane. With this software we offer a cohesive package that brings together all the required procedures for manipulating these generating functions, as well as a unified interface to deal with them. We also display results that this package offers on a public webpage.

**Keywords**: Lattice walks, generating functions, D-algebraic functions, elliptic functions, Sage.

## 1 Introduction

During the last years, there have been several studies concerning the nature of generating functions that count lattice walks in the quarter plane $\mathbb{Z}^2_{\geq 0}$ [5, 3, 14]. The classification was completed for *unweighted models with small steps* in 2018 [7], when all generating functions were fully classified.

The methods deployed to perform this classification involve various tools, ranging from algebra to complex analysis, computer algebra, probability theory, number theory and (difference) Galois theory. Most of these methods were intended to work not only for unweighted models, but also for other models with probabilistic interpretation [10]. Results concerning these weighted models have appeared in recent publications [12, 9, 8, 6].

With the package `comb_walks` (Combinatorial Walks) that we are presenting in the present paper, we offer a new interface in Sage [16] to perform these operations automatically, hence making it possible for users to recover information available for unweighted models and to explore weighted models as well. Indeed, our package allows the user to create custom weighted (or unweighted) models and to perform computations in order to get the various functions, morphisms, local expansions, and functional equations of interest, whose definitions will be given in Section 2: the kernel $K(x, y, t)$, the specializations $b_1$ and $b_2$,

*AJP was supported in part by the Austrian Science Fund (FWF): W1214-N15, project DK15.

†AB, FC and PL were supported in part by the ANR DeRerumNatura project, grant ANR-19-CE40-0018 of the French *Agence Nationale de la Recherche*.

the curve $E_t$, its automorphisms $\iota_1$, $\iota_2$, $\tau$, the analysis of poles over the curve $E_t$, and the telescoping equations in $\mathbb{C}(E_t)$ w.r.t. the automorphism $\tau$.

In Section 2 we describe the theoretical environment, going from the combinatorial description of walk models to the geometric objects involved. In Section 3 we detail the features of our package `comb_walks`. Section 4 provides examples of use of the package, including the description of a website that compiles parameters of interest for several dozens of models of combinatorial walks.

This work is a spin-off from an ongoing work [2], in which the four authors strive to extend the theory of [7] in order to obtain algorithmically explicit differential algebraic equations for walk models, when they exist. As an intermediate step, the four authors decided to gather concrete results on the website that is presented in Section 4. The code in itself was written by AJP and it was used to generate the website.

## 2 Walks and their elliptic curves

In this work, we only consider models of quarter-plane walks with small steps, i.e., with steps taken from $\mathcal{U} := \{-1, 0, 1\}^2 \setminus \{(0, 0)\}$. More specifically, a walk model is defined by a fixed subset $\mathcal{S}$ of $\mathcal{U}$, which contains the allowed steps of the model. A walk is then just a finite sequence $w = (w_1, \ldots, w_k)$ with steps $w_i$ taken from $\mathcal{S}$. The enumeration we are interested in is by the *length* $k$, which we refine by counting the number of walks starting at $(0, 0)$ and ending at some $(i, j)$ after $k$ steps, according to the decomposition $(i, j) = w_1 + \cdots + w_k$. However, we more generally count walks in a weighted manner: given a fixed family of weights, $\mathcal{D} = (d_s)_{s\in\mathcal{S}}$, the weight of a walk $w = (w_1, \ldots, w_k)$ is defined as the product of the weights of its steps, $d_{w_1} \cdots d_{w_k}$.

For a walk model $\mathcal{D}$, we then denote as $q_{i,j,k}$ the sum of the weights of walks starting at $(0, 0)$ and ending at $(i, j)$ after $k$ steps while remaining in the quarter plane $\mathbb{Z}^2_{\geq 0}$. This last constraint corresponds to all partial sums $w_1 + \cdots + w_\ell$ having nonnegative coordinates for $1 \leq \ell \leq k$. In case $d_s = 1$ for each $s \in \mathcal{S}$, all relevant walks have weight 1, so $q_{i,j,k}$ just counts the number of walks with prescribed length and endpoints. We collect all those weighted values $q_{i,j,k}$ in a three-variable generating function:

$$Q(x, y, t) := \sum_{i,j,k\in\mathbb{Z}_{\geq 0}} q_{i,j,k} x^i y^j t^k \ \in \mathbb{Q}[x, y][[t]].$$

The classification completed in the last years [1, 7] sorts the models according to the nature of their generating function:

- Rational: there are $A(x, y, t), B(x, y, t) \in \mathbb{Q}[x, y, t]$ such that $Q(x, y, t) = A(x, y, t)/B(x, y, t)$.
- Algebraic: $P(Q) = 0$ for some $P(Z) \in \mathbb{Q}[x, y, t][Z] \setminus \{0\}$.
- D-finite: for all $v \in \{x, y, t\}$ there are $p^v_0, \ldots, p^v_{r_v} \in \mathbb{Q}[x, y, t]$, not all zero, such that

$$p^v_{r_v} \partial^{r_v}_v Q + \ldots + p^v_0 Q = 0.$$

- D-algebraic w.r.t. some $V \subseteq \{x, y, t\}$: $Q$ satisfies a non-trivial, possibly nonlinear, differential equation w.r.t. variables in $V$, and coefficients in $\mathbb{Q}$.
- D-transcendental w.r.t. $V$: $Q(x, y, t)$ is not D-algebraic w.r.t. $V$.

The starting point of all methods is that the generating function $Q$ satisfies the following functional equation:

$$Q(x, y, t)K(x, y, t) = xy - Q(0, 0, t)K(0, 0, t) + Q(x, 0, t)K(x, 0, t) + Q(0, y, t)K(0, y, t),$$

where $K(x,y,t)$ is called the *kernel polynomial* and only depends on the allowed steps and weights of the model:

$$K(x,y,t) := xy\left(1 - t\sum_{(i,j)\in\mathcal{S}} d_{i,j}x^i y^j\right) \in \mathbb{Q}[x,y,t].$$

Previous works [10, 1, 4, 8, 7, 5, 13] have focused on the study of the kernel polynomial. This polynomial defines a curve $E_t$ in $\mathbb{P}(\mathbb{C}) \times \mathbb{P}(\mathbb{C})$ which is either rational (genus 0) or elliptic (genus 1). In what follows *we focus on the elliptic case*, where $E_t$ has a natural structure of abelian group. Given a step model, there is a distinguished subgroup $G_t$ of the group of automorphisms of $E_t$, generated by two involutions $\iota_1$ and $\iota_2$. The composition of both involutions leads to another automorphism on the curve $\tau := \iota_2 \circ \iota_1$, which is the addition by a point of the elliptic curve $E_t$.

It has been proven [13, Theorem 4] that:

$$(\tau-1)(Q(x,0,t)K(x,0,t)) = \tau(y)\cdot(\tau(x)-x) =: b_1, \qquad (\tau-1)(Q(0,y,t)K(0,y,t)) = x\cdot(\tau(y)-y) =: b_2.$$

Moreover, there is a derivation $\delta$ associated to a holomorphic 1-form defined over $\mathbb{C}(E_t)$ that commutes with $\tau$. We say that a rational function $f \in \mathbb{C}(E_t)$ telescopes if there is a linear differential operator $L \in \mathbb{C}[\delta]$ (*telescoper*) and a rational function $g \in \mathbb{C}(E_t)$ (*certificate*) such that $L(f) = \tau(g) - g$.

Then, putting together results in [1, 8, 7] for unweighted models, the nature of $Q(x,y,t)$ depends only on the order of $\tau$ and on a telescoping property of the functions $b_1$ and $b_2$:

- $Q$ is D-finite $\iff \operatorname{ord}(\tau) < \infty$.
- $Q$ is algebraic $\iff \operatorname{ord}(\tau) < \infty$ and $b_i$ telescopes in $\mathbb{C}(E_t)$ for $i = 1, 2$.
- $Q$ is D-algebraic w.r.t. $x \iff Q$ is D-algebraic w.r.t. $y \iff \operatorname{ord}(\tau) = \infty$ and $b_i$ telescopes in $\mathbb{C}(E_t)$ for $i = 1, 2$.

For weighted models, (more or less) complete answers exist, both in the genus-0 case [8] and in the genus-1 case [9, 6]. For the five models with genus zero, $Q$ is D-transcendental w.r.t. $x$ and $y$ [8], and even w.r.t. $\{x, y\}$ [6]. For 42 models out of the 51 models with genus one, $Q$ is D-transcendental w.r.t. $x$ and $y$ [9], at least for *generic* weights $d_{i,j}$.

# 3 Description of the package

In this section we describe the package `comb_walks`: its installation and features. Some of the methods are based on algorithms whose precise description and correctness will appear elsewhere [2].

## 3.1 Installation and documentation

Readers willing to try the package are invited to install the current (development) version from the Git repository in their Sage installation using *pip*:

```
sage -pip install [--user] git+https://gitlab.inria.fr/discretewalks/comb_walks.git
```

The package can then be loaded by the Sage command

```
sage: from comb_walks import *
```

The package offers a full implementation in Sage of all its functionalities. However, whenever it is available on the system, it uses the package `algcurves` from Maple [15], which uses van Hoeij's algorithm [17] to compute the Weierstrass normal form, obtaining more manageable results and faster executions.

An extended documentation of the package is available online at https://discretewalks.gitlabpages.inria.fr/comb_walks/docs/.

### 3.2 End-user data structure: WalkModel

The user inputs a model to be used with our package by using the class `WalkModel`. An instance of this structure is created from a list of *tuples* of the format $(i, j, d_{i,j})$, where:

- $i$ and $j$ have to be integers, representing an allowed step $(i, j)$ for the walks.
- $d_{i,j}$ has to be a number that represents the *weight* of the step $(i, j)$ in the model. This value can be omitted, having a default value of 1. The current implementation only accepts rational weights.

The following considerations during the creation of a `WalkModel` object are taken:

- If a tuple is badly formatted (i.e., has too few elements or if the types do not match), an error is raised.
- If a tuple indicates a new weight for a step $(i, j)$ that is already included, this tuple is ignored.
- An extra argument `name` can be included as a string representing the model.

For example, if the user wants to create the instance for the Kreweras model [4] where the weight for the northeast step is 2, the user can type:

```
sage: WalkModel((-1,0), (0,-1), (1,1,2), name='my Kre')
Walk Model (my Kre)
```

This structure has several methods to visualize and generate walks fitting the model:

- `weight`: receives a tuple $(i, j)$ and returns the weight of such step. It returns 0 if the step is not included.
- `is_weighted`: checks whether all allowed steps have or not the same weight.
- `random_walk`: creates an valid random walk for the model. The probability distribution of the steps is proportional to the weight (i.e., two steps with the same weight have the same probability of being picked).
- `plot`: shows a picture with the allowed steps of the model. No weights are represented.
- `plot_random_walk`: shows a random walk for the model.

### 3.3 Geometric objects from a model

The structure `WalkModel` offers several methods to access the algebraic and geometric information of the model (as described in Section 2). Since the curve $E_t$ can be viewed in different ambient spaces, namely $\mathbb{P}^2(\mathbb{C})$ or $\mathbb{P}(\mathbb{C}) \times \mathbb{P}(\mathbb{C})$, all these methods have an argument `model` that allows the user to choose the representation:

- The representation `affine` (or `A`) uses the coordinates $x$ and $y$ that appear in the generating function. This adds a projectivization variable $z$ to put the object in $\mathbb{P}^2(\mathbb{C})$.
- The representation `projective` (or `P`) uses the coordinates $x$ and $y$ and projectivizing both independently, getting four variables $x = (x_0 : x_1)$ and $y = (y_0 : y_1)$ putting the objects in $\mathbb{P}(\mathbb{C}) \times \mathbb{P}(\mathbb{C})$.
- The representation `Weierstrass` (or `W`), only valid when the kernel polynomial represents an elliptic curve, uses a set of coordinates $(u : v : w)$ where the kernel is written in Weierstrass form $v^2w - 4u^3 - g_2uw^2 - g_3w^3$.

The methods available for a `WalkModel` are:

- `kernel`: given the representation (`A`, `P` or `W`), returns the kernel polynomial $K$ that defines the curve $E_t$.
- `curve`: given the representation, returns the curve defined by the kernel, represented as a Sage algebraic scheme.
- `iota`: given the representation and an index $j$, returns $\iota_j$.
- `tau`: given the representation, returns the map $\tau$.
- `order_tau`: method that computes the order of the map $\tau$. This methods receives an argument `bound` and checks if the order of $\tau$ is smaller than or equal to this bound, returning $\infty$ if it is bigger. The default value for `bound` is 10.
- `b`: method that receives an index $j$ and computes the rational function $b_j$ described in Section 2.
- `map`: given two possible representations of two elliptic curves, returns a birational map between the curves in their different representations.
- `is_elliptic`: returns if the curve $E_t$ is elliptic or not.
- `neutral_point`: if the curve is elliptic, returns the neutral point of the curve.
- `g2` and `g3`: if the curve is elliptic, returns the invariants $g_2$ and $g_3$ of the curve $E_t$.

### 3.4 Geometric methods for `WalkModel`

As we saw in Section 2, the classification of models requires the analysis of some rational functions over the curve $E_t$. This is allowed in the package using the following methods:

- `poles`: receives a rational function $f$ over $E_t$ and returns the list of poles of $f$ over $E_t$. This method relies on Sage's ability to factor univariate polynomials.
- `higher_polar_part`: receives a function $f$ and a point $P \in E_t$ and returns a rational function $g$ which has only a pole at $P$ and such that $f - g$ has at most a simple pole at $P$.
- `orbits`: receives a list of points $P_1, \dots, P_n$ and a bound $m$ and computes the $\tau$-orbits for those points, approximated by splitting orbits at "gaps" longer than $m$. This method returns lists of points $L_i = (Q_{i1}, \dots, Q_{ik_i})$ such that:
  - For all $i$ and all $j \in \{1, \dots, i_k - 1\}$, there is $1 \leq n_{ij} \leq m$ such that $\tau^{n_{ij}}(Q_{ij}) = Q_{i,j+1}$.
  - For all points $R_1, R_2$ in different lists and all $1 \leq n \leq m$, $\tau^n(R_1) \neq R_2$.
- `telescoping`: receives a rational function $f$ over $E_t$ and computes (when they exist) a differential operator $L$ and a rational function $g$ such that $L(f) = \tau(g) - g$.

### 3.5 Further algebraic geometry methods

Some of the operations already described require some computations based on algebraic geometry. The current implementation of Sage, although generic, did not fit our requirements, for example, applying a map between algebraic varieties may fail even when the map is defined for such point.

That is the reason we implemented several methods only based in algebraic geometry (i.e., without the assumptions of working over the kernel curve $E_t$) in the subpackage `alggeo`:

- `pullback`: given a rational map between varieties, returns its corresponding pullback.
- `order_morphism`: given a rational map $f$ and a bound $m$, checks if $f^n = id$ for $1 \leq n \leq m$.
- `zeros_bihom`: given a bivariate homogeneous polynomial, computes the points in $\mathbb{P}(\mathbb{C})$ where the polynomial vanishes.
- `asymptotics`: given a curve $E$, a rational function $f$ over $E$ and a point $P \in E$, this method computes the order of $f$ at $P$, a local parameter $s$ of $E$ at $P$ and the first non-zero coefficient of the expansion of $f$ at $P$ w.r.t. $s$.
- `expand_at_point`: given a curve $E$, a rational function $f$ over $E$, a point $P \in E$, and an integer $m \geq 0$, computes a local parameter $s$ of $E$ at $P$ and the first $m$ coefficients of the expansion of $f$ at $P$ w.r.t. $s$.

### 3.6 Default variables

When the user loads the package, several variables are available as default models and values. We describe them now:

- **Small steps**: there are simple variables like `N` or `SW` that represent small steps with weight 1. This means that the user can type

  ```
  sage: WalkModel(NE, S, W)
  ```

  to create the unweighted Kreweras model.

- **Unweighted models**: we offer several variables that include all the unweighted models treated in the literature [1, 7, 3] that are relevant:
  - `AllModels`: a list with all the models.
  - `FiniteGroup`: all models where the order of $\tau$ is finite. These models are numbered according to the numbering they received in [1].
  - `EllipticC`: all models where the curve is elliptic and the order of $\tau$ infinite. These models are named following the convention described in [7].
  - `NonEllipticC`: models whose curve is not elliptic.
  - `ModelDict`: a dictionary that, given the name, provides the corresponding model.

## 4 Examples of use

With the functionality described in Section 3, we offer a wide set of possibilities to use it: the user can use it as a white box, and use all the properties and objects offered to perform new and step-by-step computations over different models, or can use it as a black box to check bigger results like whether a rational function on the curve $E_t$ telescopes or not.

As a first example of use as a black box, we include here a simple and intuitive code that implements the classification criteria for unweighted models described in Section 2:

```
sage: def type_of_model(m):
....:     if is_weighted(m):
....:         return "Weighted model"
....:     order = m.order_tau()
....:     try:
....:         m.telescoping(m.b(2)(x = x0/x1, y = y0/y1))
....:         telescopes = True
....:     except:
....:         telescopes = False
....:     if order < Infinity and telescopes:
....:         return "Algebraic"
....:     elif order < Infinity:
....:         return "D-finite"
....:     elif telescopes:
....:         return "D-algebraic w.r.t.  'x' and 'y'"
....:     return "D-transcendental w.r.t.  'x' and 'y'"
```

This code is proven to be correct, because it is known [1] that, in the cases of $\tau$ having finite order, this can only be 2, 3 or 4. Hence, the return of `order_tau` is in this case always correct.

Now, we can use this function to quickly check the type of the model for several instances:

```
sage: type_of_model(WalkModel(NE, W, S))
Algebraic
sage: type_of_model(WalkModel(N, S, W, E))
D-finite
sage: type_of_model(WalkModel(N, E, S, SW))
D-algebraic w.r.t.  'x' and 'y'
sage: type_of_model(WalkModel(N, S, W, E))
D-transcendental w.r.t.  'x' and 'y'
```

Another black-box use is how we have applied our code to all the models described in the literature and compiled the result in a public web page that can be accessed together with the documentation of the package at the url https://discretewalks.gitlabpages.inria.fr/comb_walks/. (The list of displayed models is that in the variable `AllModels`; see Section 3.)

On this webpage, we offer a list of parameter of interest for all models that can be filtered to avoid too many information on the screen. Moreover, the user can search models by name and type, making it easier to browse through the results of several models.

## 5 Conclusions

We have presented a Sage software which, to our knowledge, is the first package to classify algorithmically walks in the quarter plane. This package offers a unified interface with seamless transitions between projective and affine representations, whose primary object is the model of walks.

Our implementation of the procedures that were discussed in the literature to classify the generating functions that count walks can be applied not only to these unweighted models, but also to other models where the steps have an associated weight. This allows users to do new computations or to check step-by-step procedures and computations.

In the future, we plan to extend the current implementation so as to obtain, beside the (differentially) algebraic nature of a model, corresponding algebraic or differential equations automatically, provided they exist. This could shed more light on whether these D-algebraic functions of combinatorial origin belong to more structured subclasses: are they quotients of D-finite functions? are they more complicated DD-finite functions [11]? We also intend to complete and enhance the tests and the documentation of the package. We also plan to increase the scope of the implementation, e.g., by allowing algebraic weights to the steps of a model.

We also aim at extending the website by offering extra features, like better filtering, a direct comparison between particular models or even a dynamic webpage where the user could include a new model interactively, so as to compare the results with other models.

## References

[1] O. Bernardi, M. Bousquet-Mélou, and K. Raschel. Counting quadrant walks via Tutte's invariant method. Preprint, https://hal.archives-ouvertes.fr/hal-01577762. 54 pages, 10 figures, 10 tables, Sept. 2017.

[2] A. Bostan, F. Chyzak, A. Jiménez-Pastor, and P. Lairez. Differential equations for walk models of genus 1, 2020. In preparation.

[3] A. Bostan, F. Chyzak, M. van Hoeij, M. Kauers, and L. Pech. Hypergeometric expressions for generating functions of walks with small steps in the quarter plane. *European Journal of Combinatorics*, 61:242–275, 2017.

[4] M. Bousquet-Mélou. Walks in the quarter plane: Kreweras' algebraic model. *The Annals of Applied Probability*, 15(2):1451–1491, 2005.

[5] M. Bousquet-Mélou and M. Mishna. Walks with small steps in the quarter plane. In *Algorithmic probability and combinatorics*, volume 520 of *Contemp. Math.*, pages 1–39. Amer. Math. Soc., Providence, RI, 2010.

[6] T. Dreyfus and C. Hardouin. Length derivative of the generating series of walks confined in the quarter plane, 2019. Preprint, https://arxiv.org/abs/1902.10558.

[7] T. Dreyfus, C. Hardouin, J. Roques, and M. F. Singer. On the nature of the generating series of walks in the quarter plane. *Inventiones Mathematicae*, 213(1):139–203, 2018.

[8] T. Dreyfus, C. Hardouin, J. Roques, and M. F. Singer. Walks in the quarter plane: Genus zero case. *Journal of Combinatorial Theory, Series A*, 174, 2020.

[9] T. Dreyfus and K. Raschel. Differential transcendence & algebraicity criteria for the series counting weighted quadrant walks. *Publications Mathématiques de Besançon*, (1):41–80, 2019.

[10] G. Fayolle, R. Iasnogorodski, and V. Malyshev. *Random walks in the quarter plane*, volume 40 of *Probability Theory and Stochastic Modelling*. Springer, 2nd edition, 2017.

[11] A. Jiménez-Pastor and V. Pillwein. A computable extension for D-finite functions: DD-finite functions. *Journal of Symbolic Computation*, 94:90 – 104, 2019.

[12] M. Kauers and R. Yatchak. Walks in the quarter plane with multiple steps. In *Proceedings of FPSAC 2015*, Discrete Math. Theor. Comput. Sci., pages 25–36, 2015.

[13] I. Kurkova and K. Raschel. On the functions counting walks with small steps in the quarter plane. *Publications Mathématiques. Institut de Hautes Études Scientifiques*, 116:69–114, 2012.

[14] I. Kurkova and K. Raschel. New steps in walks with small steps in the quarter plane: series expressions for the generating functions. *Annals of Combinatorics*, 19(3):461–511, 2015.

[15] Maplesoft, a division of Waterloo Maple Inc. *Maple*, 2019. Waterloo, Ontario.

[16] W. Stein et al. *Sage Mathematics Software (Version 8.1)*. The Sage Development Team, 2017. http://www.sagemath.org.

[17] M. van Hoeij. An algorithm for computing the weierstrass normal form. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation*, ISSAC '95, page 90–95, New York, NY, USA, 1995. Association for Computing Machinery.