Special Session on Modern Software Tools for Analytic Combinatorics

AofA 2018, Uppsala

June 27, 2018

UPPSALA UNIVERSITET

Program

- **Jérémie Lumbroso**, *Open-source Analytic Combinatorics*
  1. Modern Programming Tools
  2. Reluctant Walks
  3. *Can* you specify it?

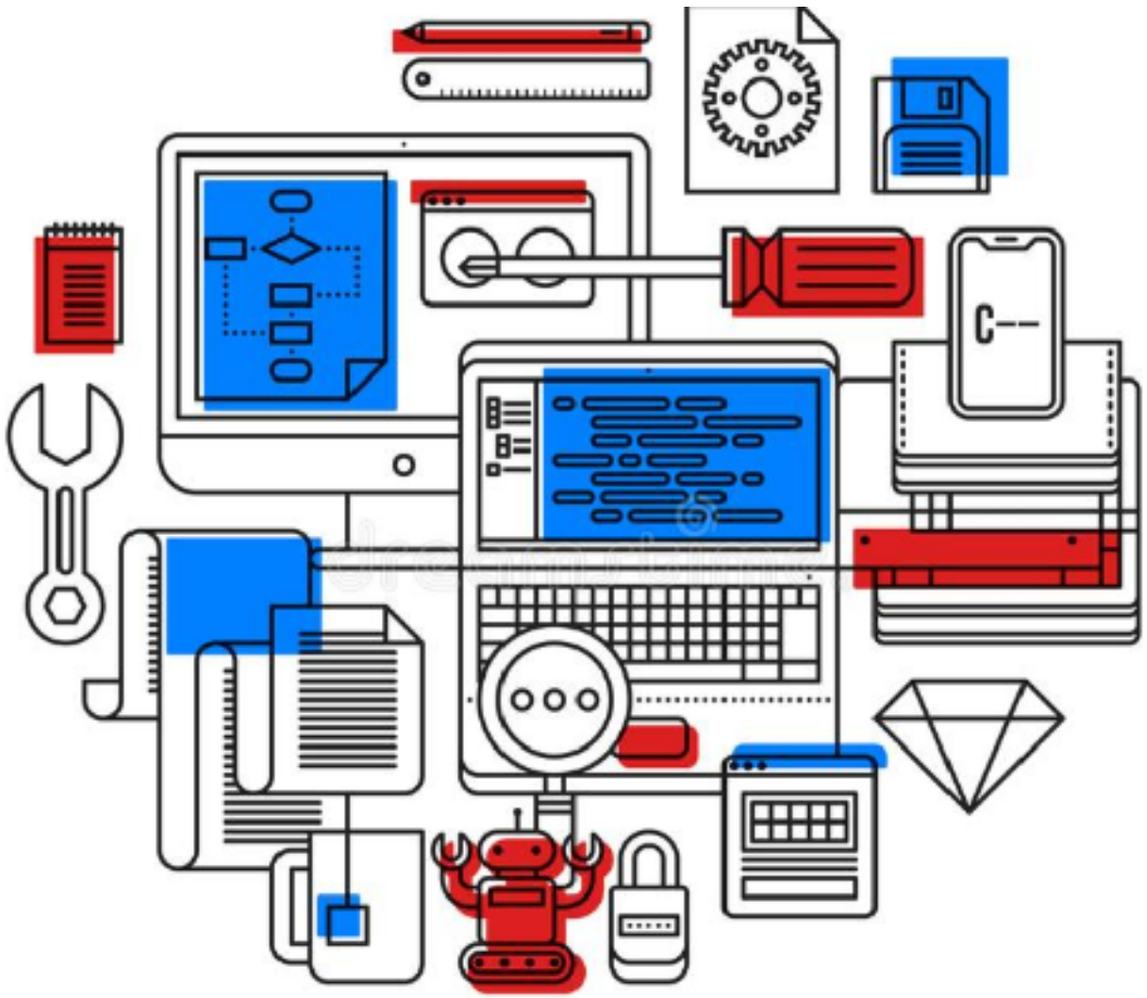
- **Maciek Bendkowski**, *Multi-parametered samplers*

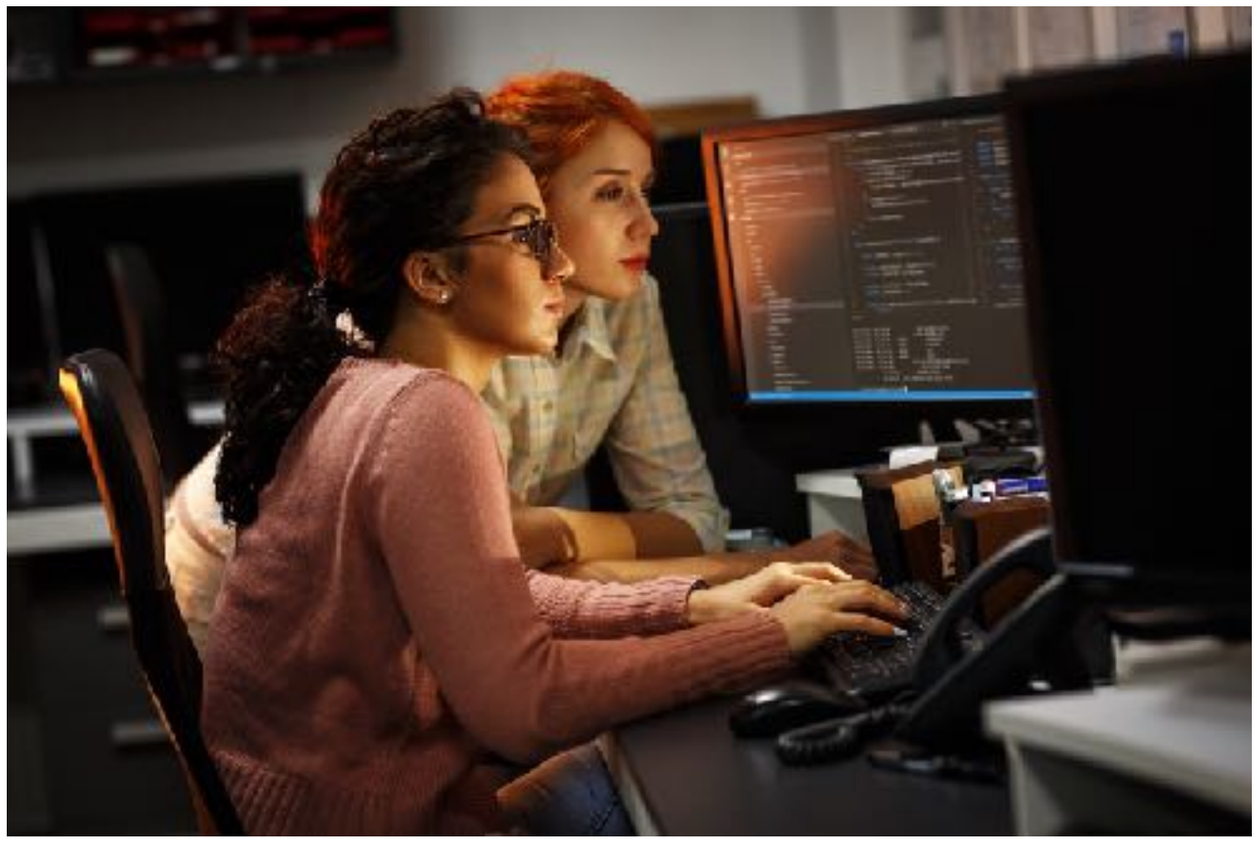
- **Daniel Krenn**, *Asymptotic analysis in SAGE*

Modern Open-Source for Analytic Combinatorics

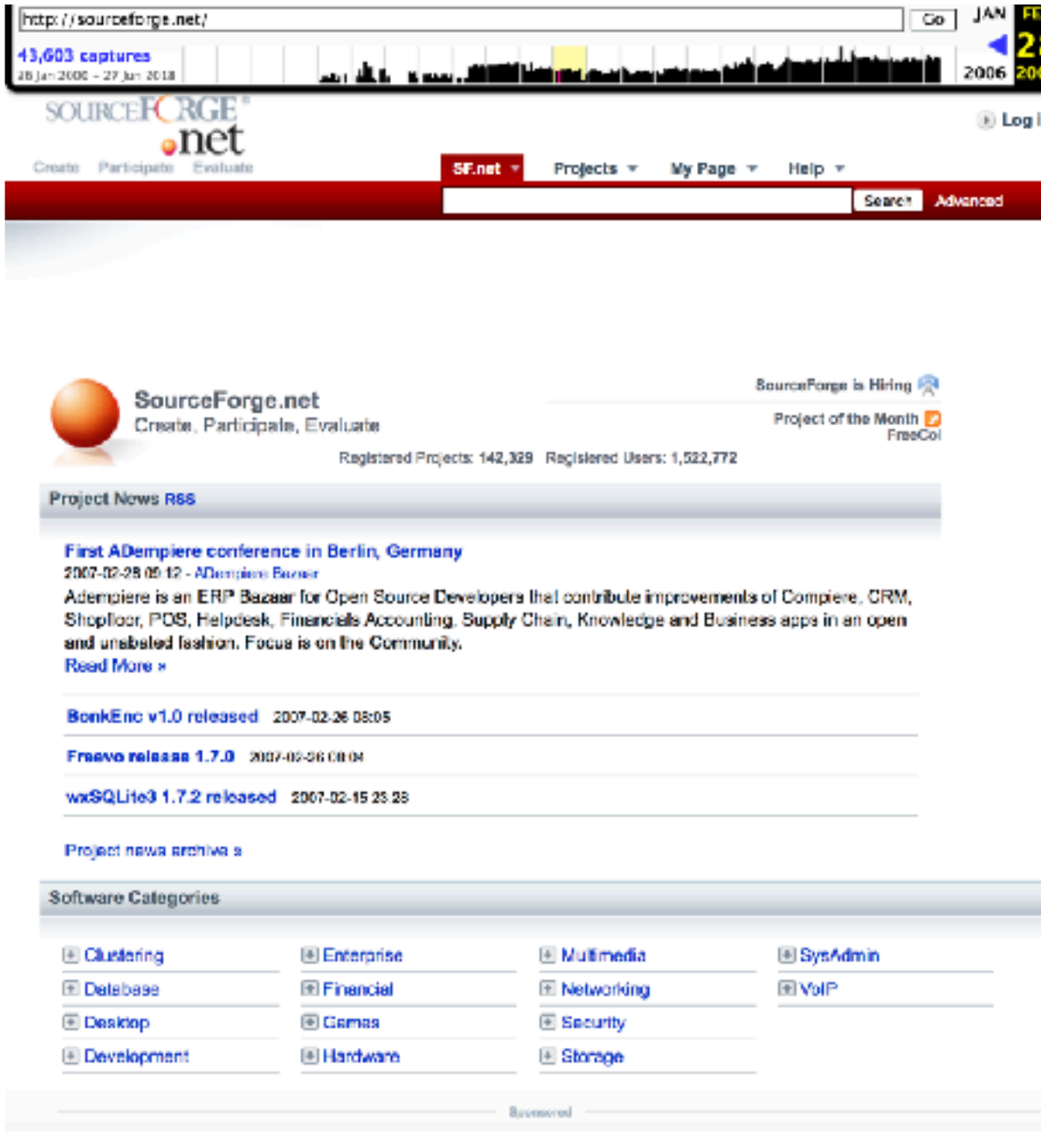
Jérémie Lumbroso
Princeton University

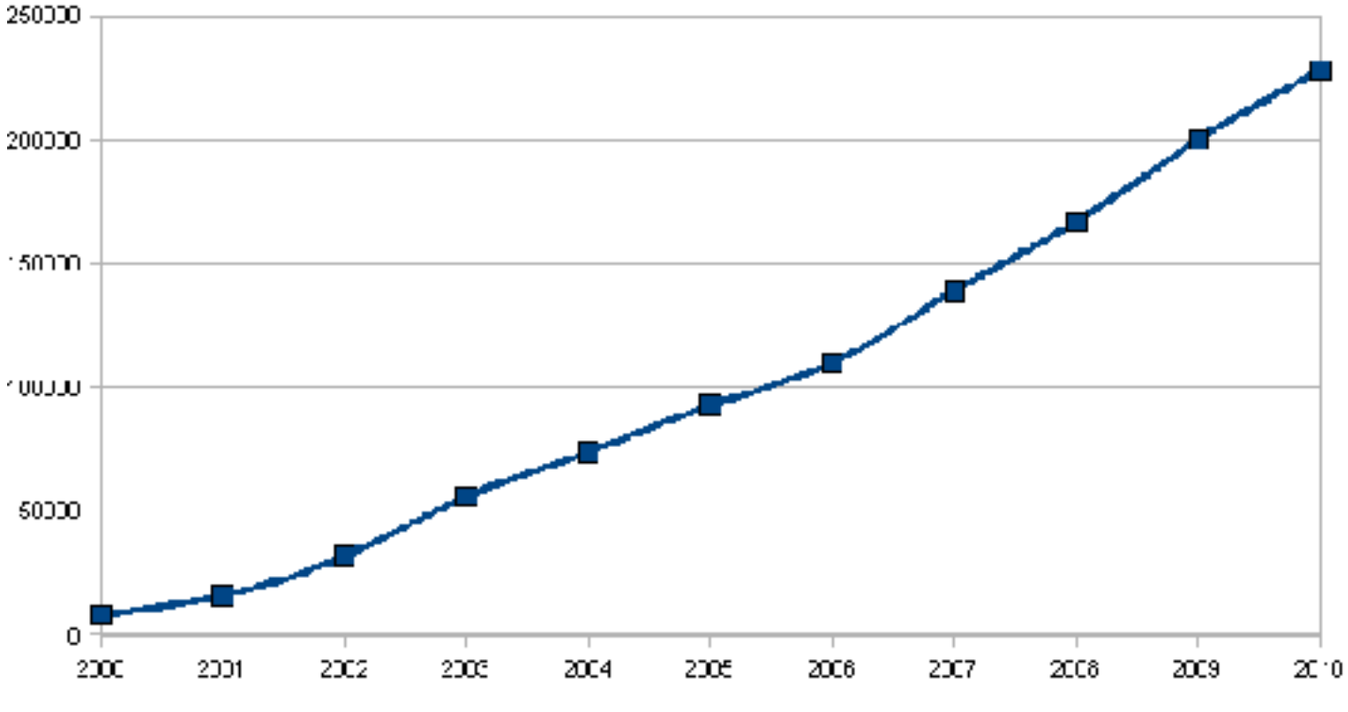
1. Modern Programming Tools





Open-Source 18 years ago





(Non-Exhaustive List of) Open-Source Projects

Back in the 2000s

Top Downloads

1 eMule
2 Azureus - BitTorrent Client
3 BitTorrent
4 DC++
5 Shareaza
6 VirtualDub
7 eMule Plus
8 CDex
9 ABC [Yet Another Bittorrent Client]
10 guliverkli

Most Active

1 Gaim
2 eGroupWare: Enterprise Collaboration
3 FCKeditor
4 MinGW - Minimalist GNU for Windows
5 Azureus - BitTorrent Client
6 Exponent Content Management System
7 7-Zip
8 phpMyAdmin
9 openCRX - Limitless Relationship Mgmt
10 WebCalendar

BitTorrent

SDGE

python

TensorFlow

More recently

Mozilla Firefox

Apache HTTP Server

MySQL

Apache Subversion

PHP

Linux Kernel

Bash

Firebug

Ubuntu

Git

OpenCV

OpenSSL

OpenCL 2.0

Linux

OpenGL

(since 2008)

GitHub

"HOW GITHUB CONQUERED GOOGLE, MICROSOFT, AND EVERYONE ELSE"
http://bit.ly/WiredGitHub

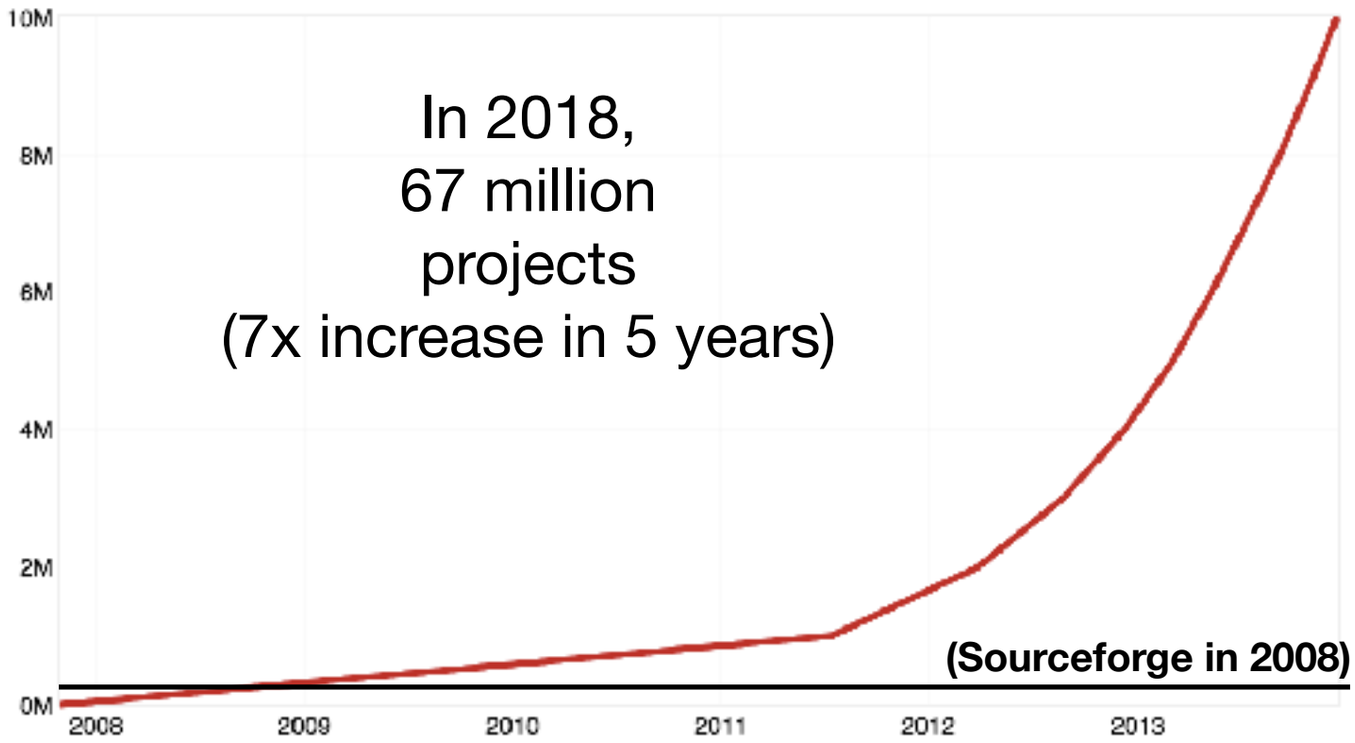
the engine of the open-source revolution

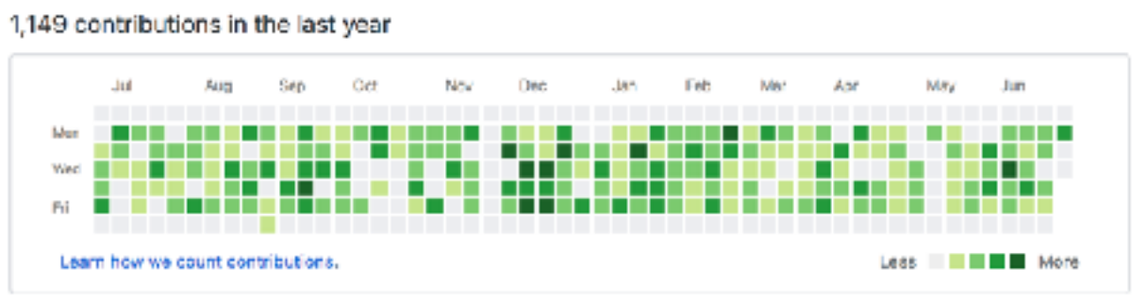
- First social media targeting developers
- Used gamification and policy to incentivize positive community contributions; guaranteed free hosting
- Streamlined collaboration by **many orders of magnitude**
- Took an active role building the open-source community

Position of GitHub in Alexa Top 50.

USA:	34
Sweden:	37
Denmark:	37
Portugal:	40
Austria:	44
U.K.:	46
China:	47

Not in Top 50 for Australia, Belgium, France, Germany, Italy, Spain.



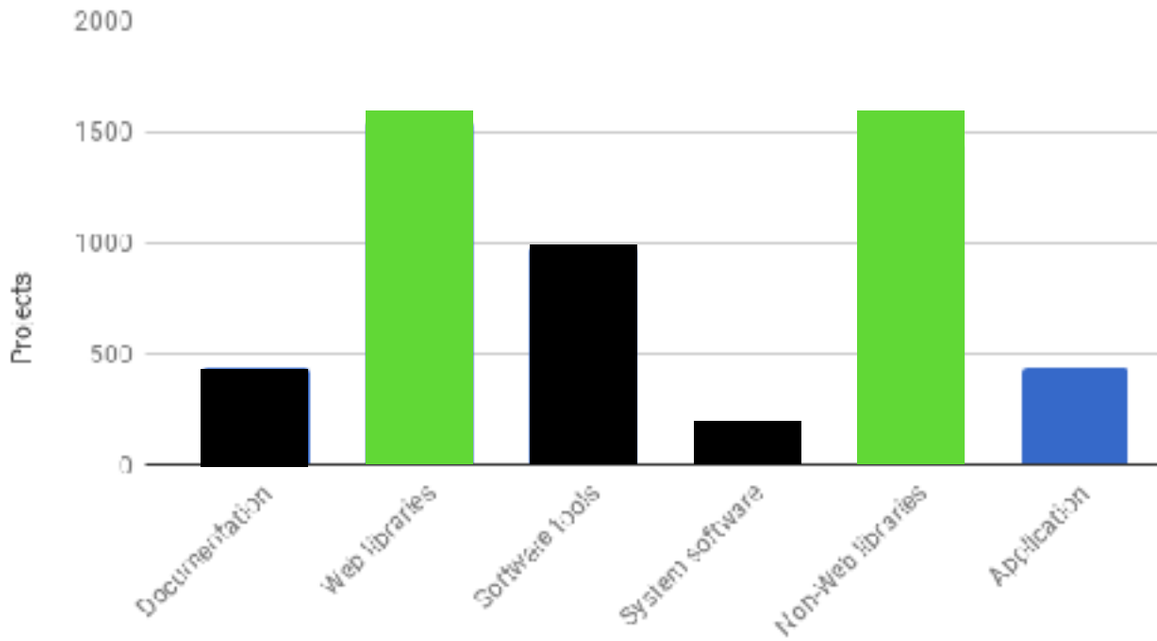


https://octoverse.github.com/ (2018)

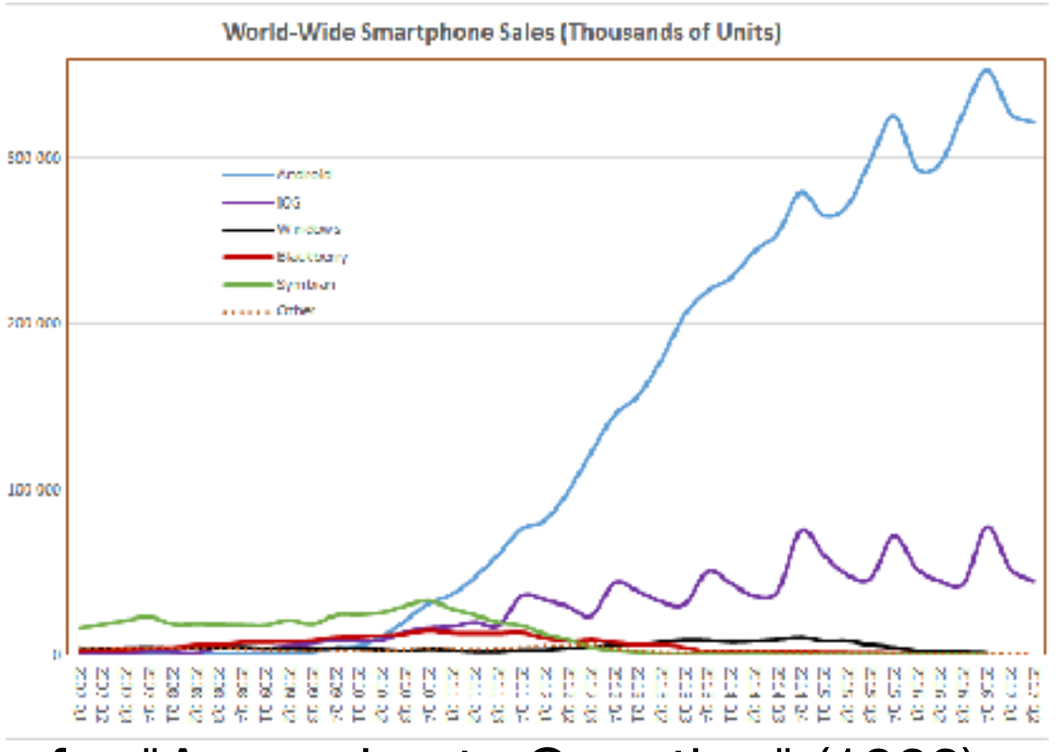
Modularity of design

Doug McIlroy*, inventor of Unix pipes:
"**Write programs that do one thing and do it well.** Write programs to work together. Write programs to handle text streams, because that is a **universal interface**."

(nowadays text streams = APIs)



@aserg.ufmg on Medium.



- The second wave of "open-source focused on libraries

- This modular design: Pioneered by Unix (Linux now dominates all OSes: macOS/iOS, Android)

- **Modular design is future-proof**

* also provided Philippe with the idea for "Approximate Counting" (1983)

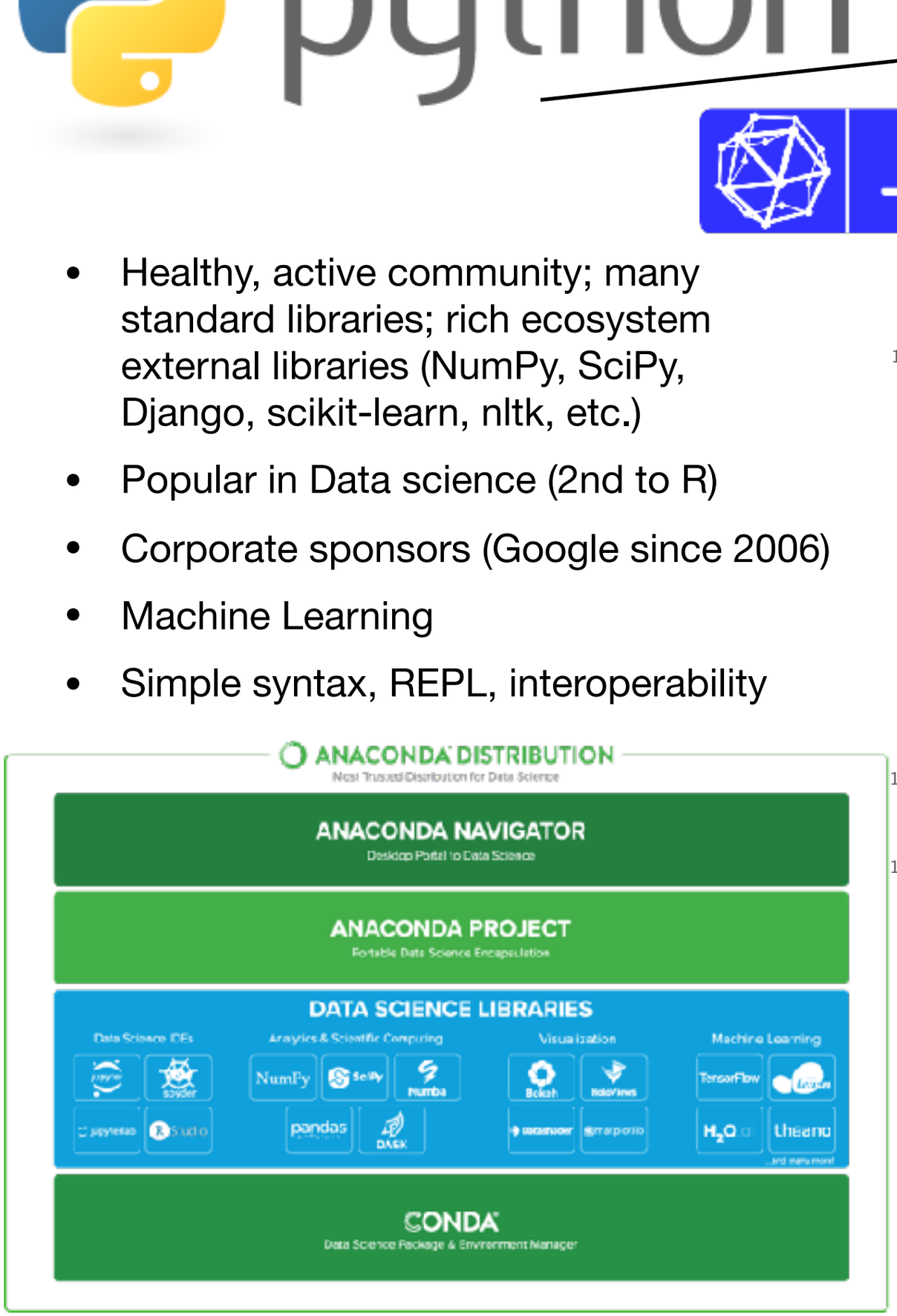
(since 1991)

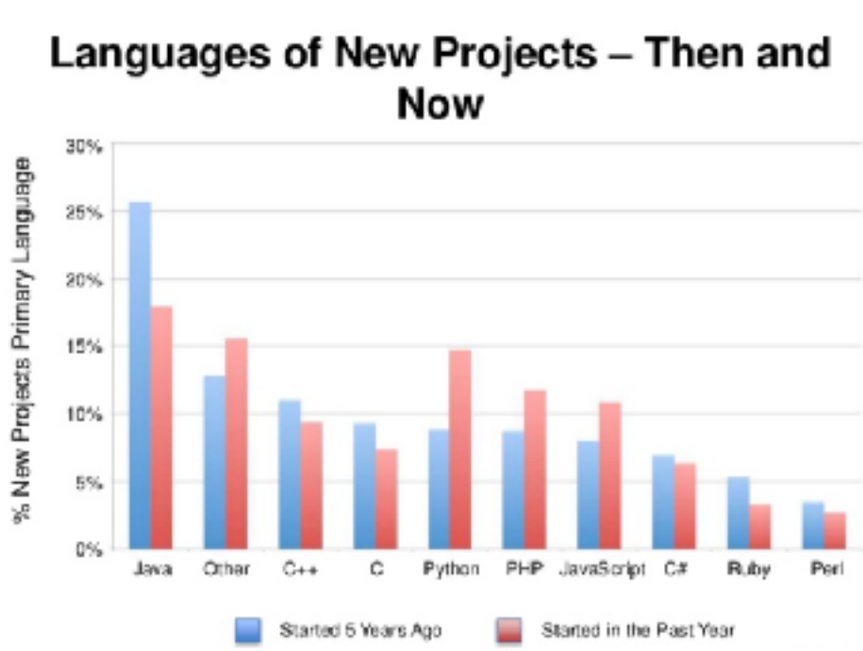
python™

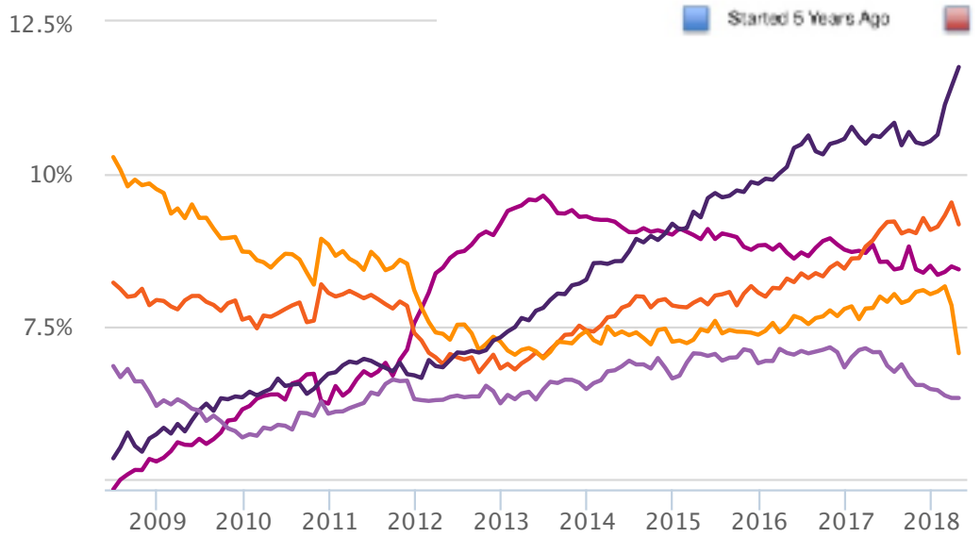
(since 2005)

sage

- Healthy, active community; many standard libraries; rich ecosystem external libraries (NumPy, SciPy, Django, scikit-learn, nltk, etc.)
- Popular in Data science (2nd to R)
- Corporate sponsors (Google since 2006)
- Machine Learning
- Simple syntax, REPL, interoperability

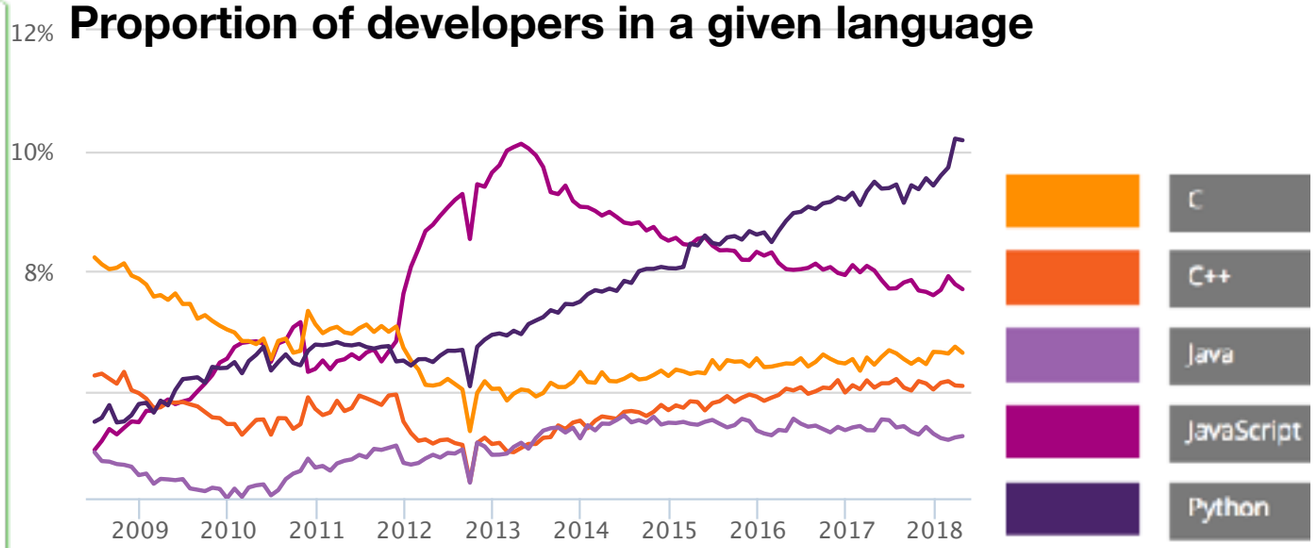






TensorFlow

Proportion of developers in a given language



Proportion of projects in a given language

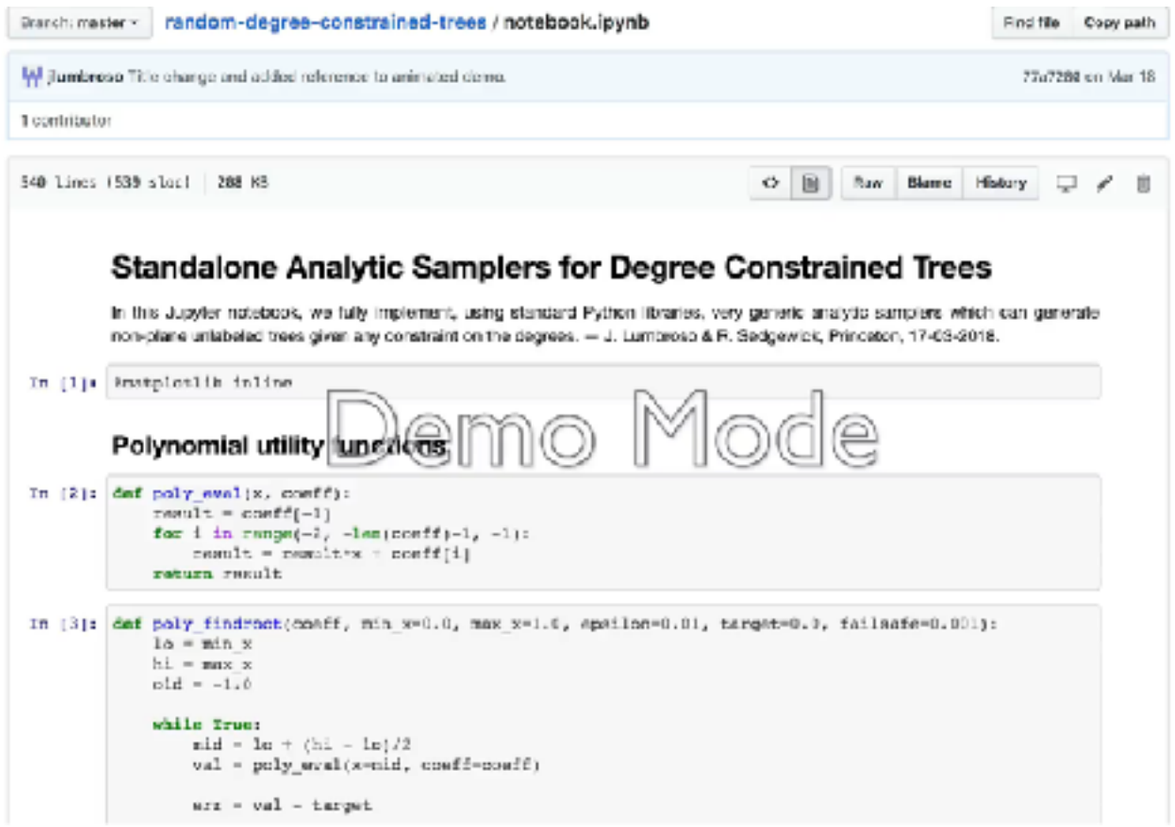
BLACKDUCK | Open Hub

- Jupyter (JUlia-PYThon-R, the three founding languages) is a standalone REPL environment, designed to **reproducible research** comfortable - also supports SAGE

- REPL (Read-Eval-Print-Loop) is how a symbolic systems work, but relatively rare for a programming language

- Python's REPL instrumental in popula

- Jupyter integrates with everything (graphic, interaction, *etc.*), and can be displayed on the web



Anecdote from R. J. Lipton

Philippe, is this conjecture true??

Conjecture (Turán): Suppose for all s with real part greater than 1, the partial sum

$$\sum_{n=1}^{M} \frac{1}{n^s}$$

is always non-zero, for M ≥ 1. Then the Riemann Hypothesis is true.

Completely false! Never seen this conjecture, but it only took me a day to *compute* several counter-examples!

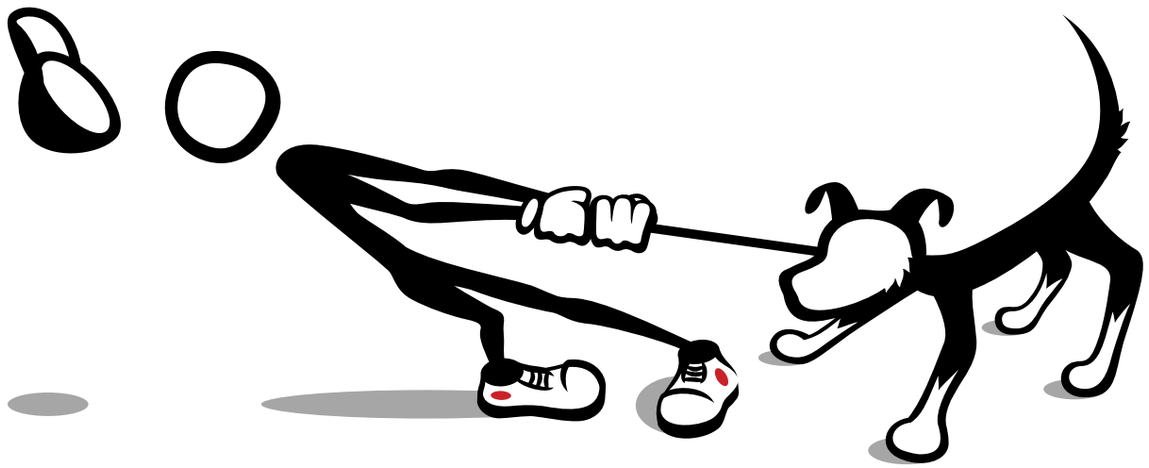
My bad. Actually, I now realize Montgomery (1983) just came out with a proof of what you have already shown me!

Zeros of approximations to the zeta function

by

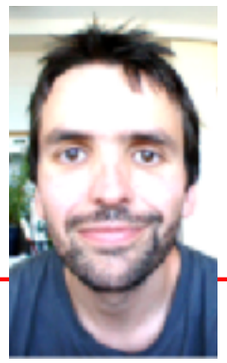
H. L. MONTGOMERY* (Ann Arbor)

2. Reluctant Walks



rejected walks

accepted walks



Lumbroso, Mishna, Ponty (2016)

"*Taming Reluctant Walks in the Quarter Plane*"

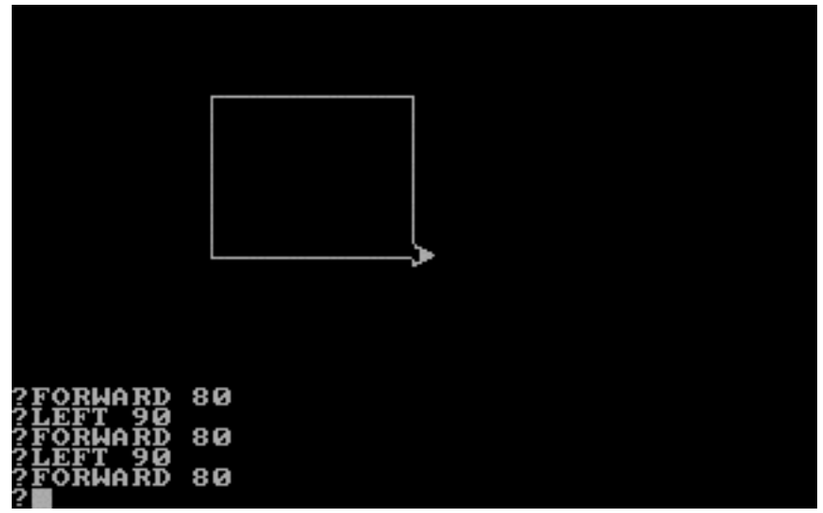
BEST SLOPE FOR THIS STEPSET

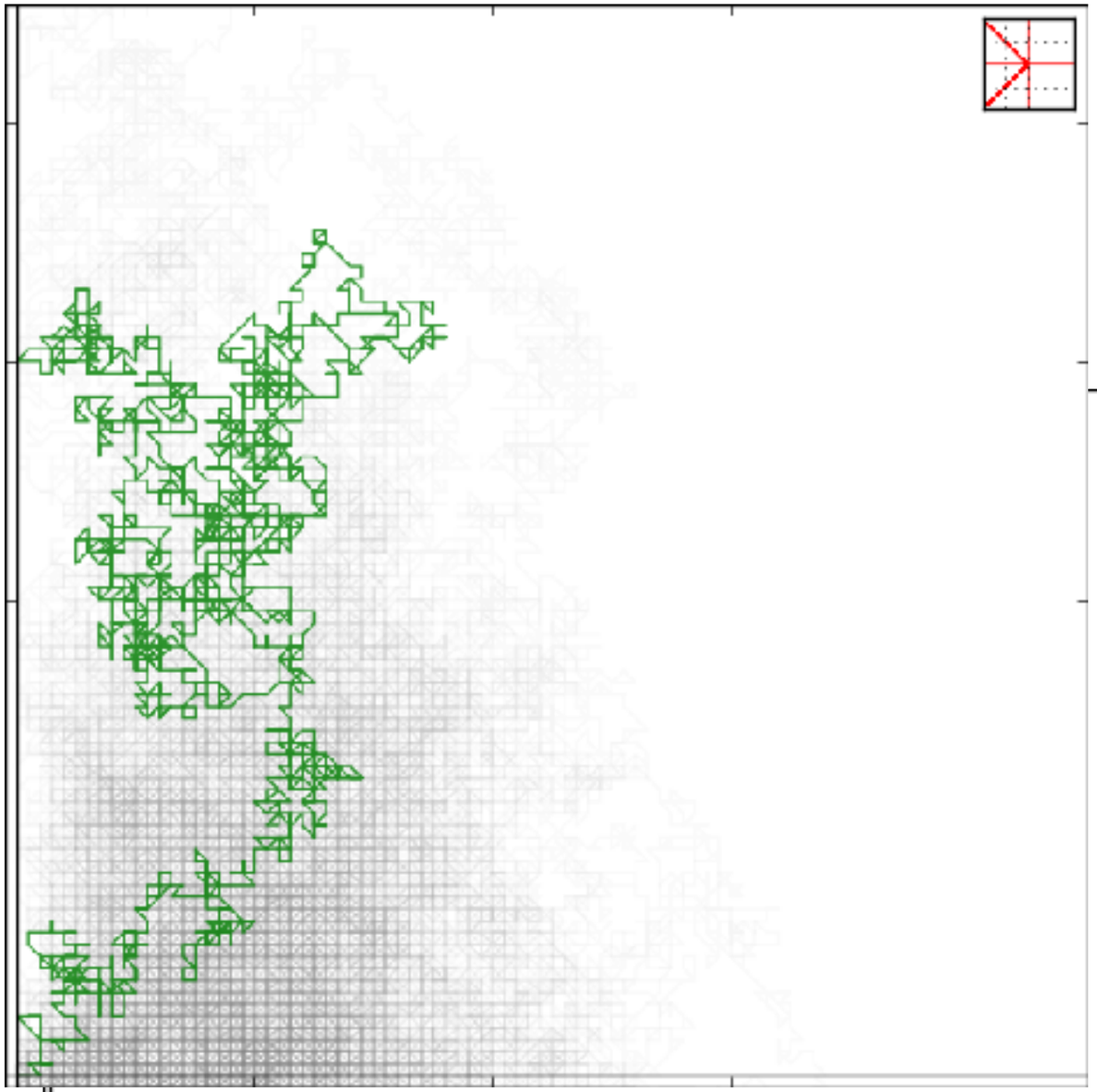
- Generating walks constrained in a half-plane is easy (1 degree of freedom = grammar)
- So generate in **a** half-plane + reject what is not in quarter-plane
- Using prior result (Johnson *et al.*), pick *best* half-plane (with least rejection)

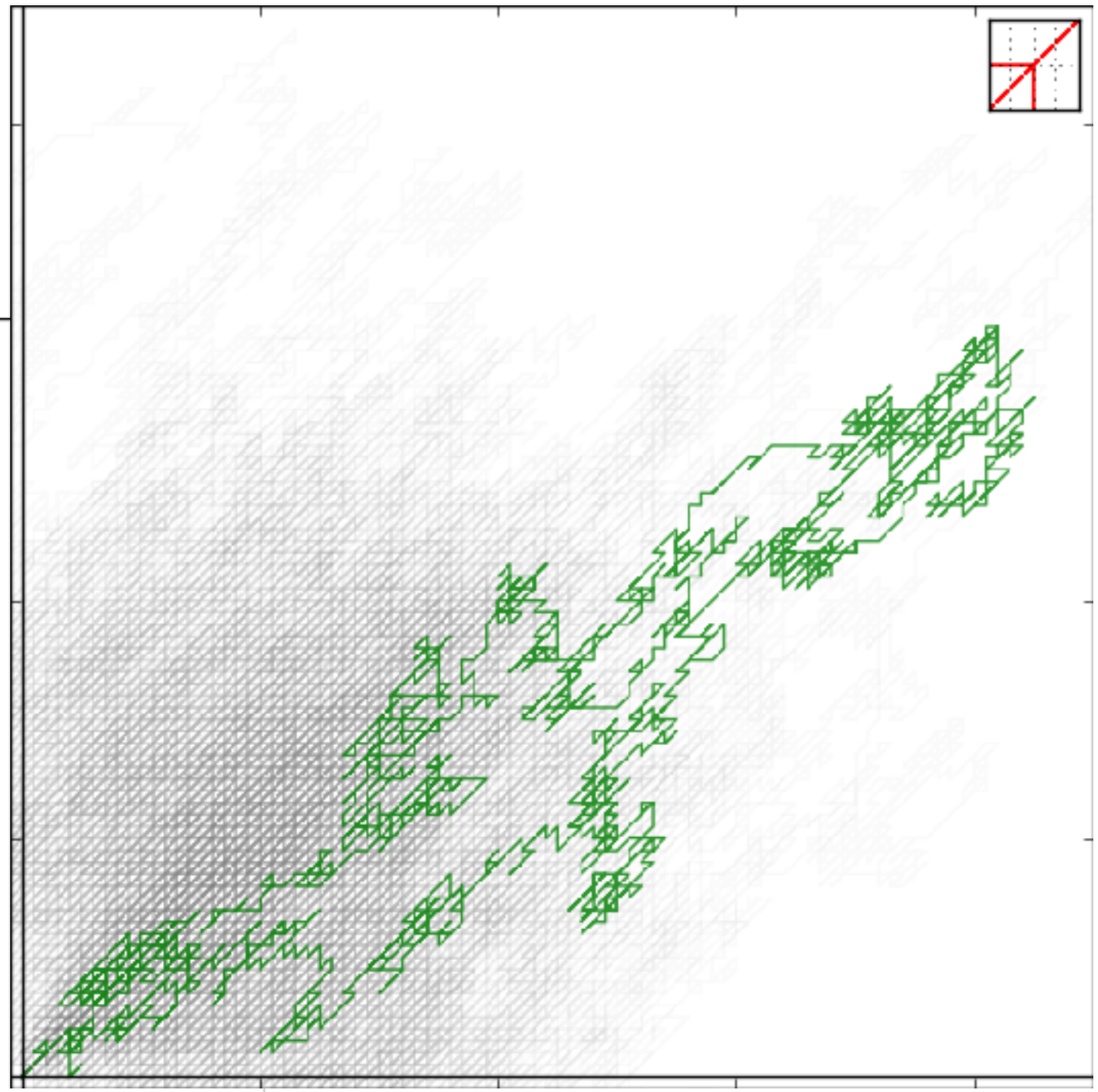
Lumbroso, Mishna, Ponty (2016)

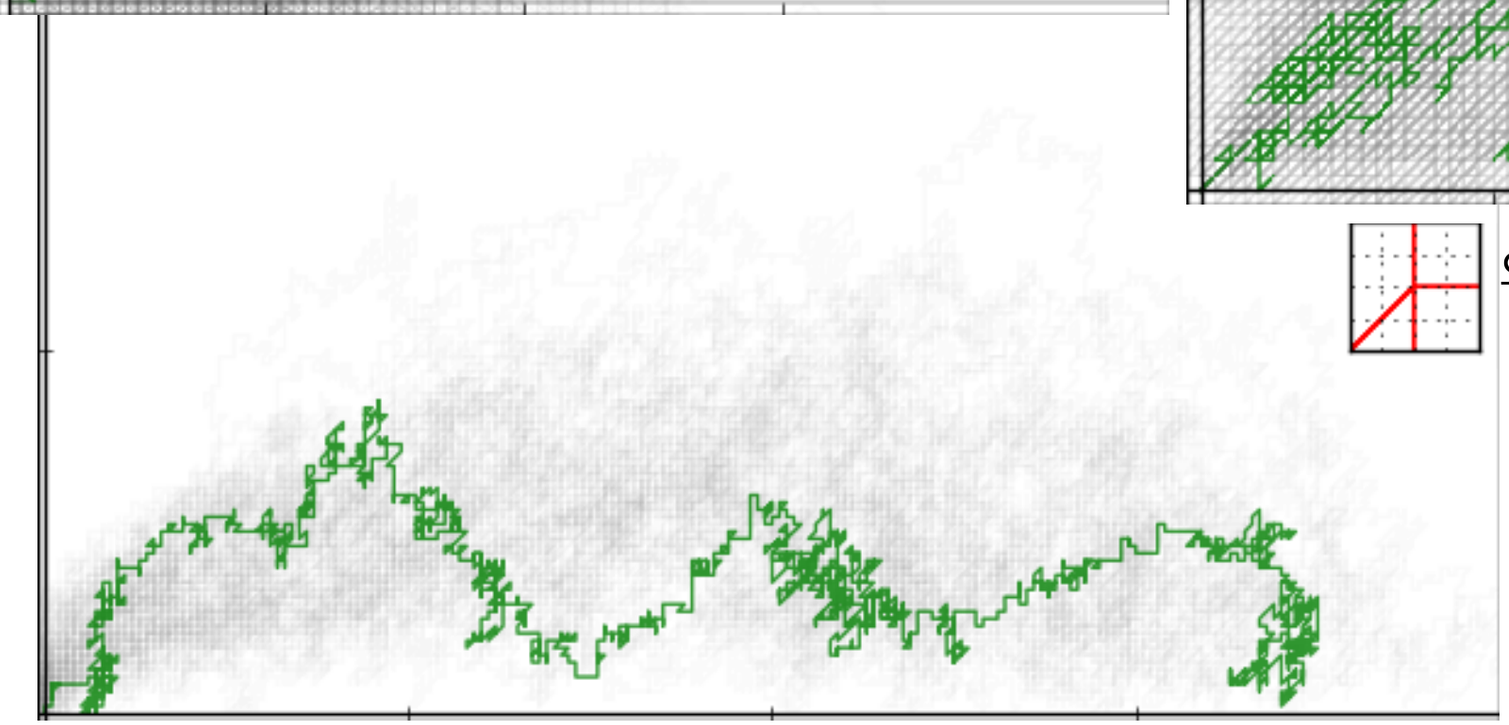
A big (91760 steps), *very difficult to generate* reluctant quarter plane walk.

`walk-6_6-91760.pdf`









github.com/jlumbroso/reluctant-walks

(2018)

jlumbroso / reluctant-walks

Code | Issues 0 | Pull requests 0 | Wiki | Insights

Branch: master · reluctant-walks / examples / notebooks / VisualizingWalks

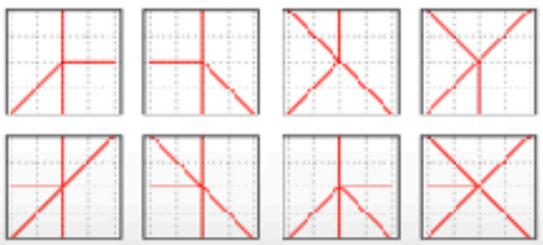
jlumbroso Moved notebooks folder.

1 contributor

334 lines (333 sloc) 310 KB

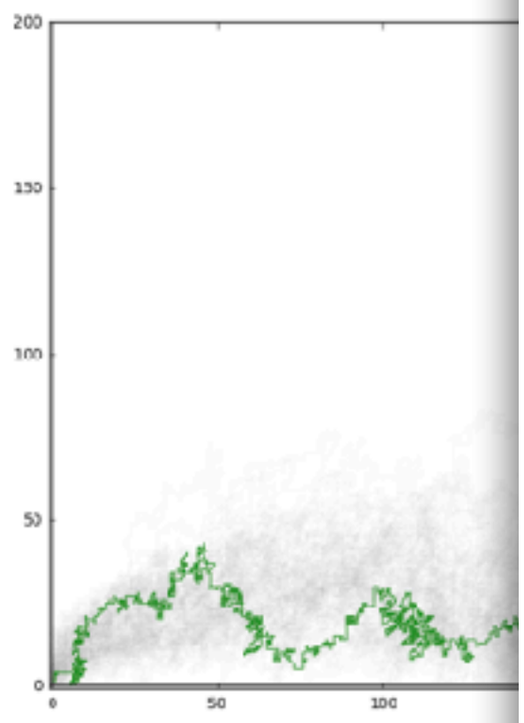
Examples of different walks

```
In [1]: %matplotlib inline

In [2]: # Fix path to have the root of the 'reluctant_walks'
        import sys
        sys.path.append("..")

In [3]: import matplotlib.pyplot as plt

        import reluctant_walks

        from reluctant_walks import Step, StepSet
        from reluctant_walks.compilers.gearyens import GenRGe
        from reluctant_walks.compilers.combstruct import Comb

        from reluctant_walks.reference import is_quarter_plan
        model, POSSIBLE_NT_SLOPES

        from reluctant_walks.graphics import plot_walk
        from reluctant_walks.graphics import plot_walk_region

        from reluctant_walks.graphics import plot_stepsets, p

In [4]: plt.rcParams['figure.figsize'] = (6.0, 6.0)

In [5]: by_best_slope = list(POSSIBLE_NT_SLOPES)
        if 0 in by_best_slope:
            del by_best_slope[by_best_slope.index(0)]
        models_1 = get_nontrivial_qw_model(by_drift=[-1], by_
        st_slope=by_best_slope)
        models_2 = get_nontrivial_qw_model(by_drift=[-2], by_
        st_slope=by_best_slope)
        models = models_1
        model = models[2]
        model

Out[5]: {'best_slope': (3, 1),
         'coord': (4, 10),
         'drift': -1,
         'id': 17,
         'size': 4,
         'steps': [(0, 1), (1, -1), (-1, -1), (-1, 1)],
         'stepset': StepSet
             St1: ( 0, +1) weight: 1
             St2: (+1, -1) weight: -1
             St3: (-1, -1) weight: -1
             St4: (-1, +1) weight: 1
         }

In [7]: ax = plot_stepsets(map(lambda x: x['stepset'], models_
        len(models_1)

Out[7]: 11
```

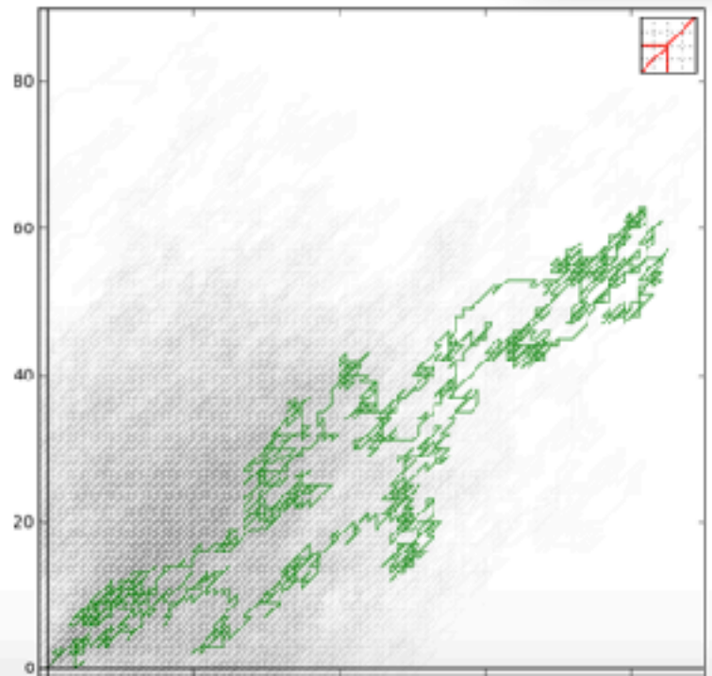


Plotting different models

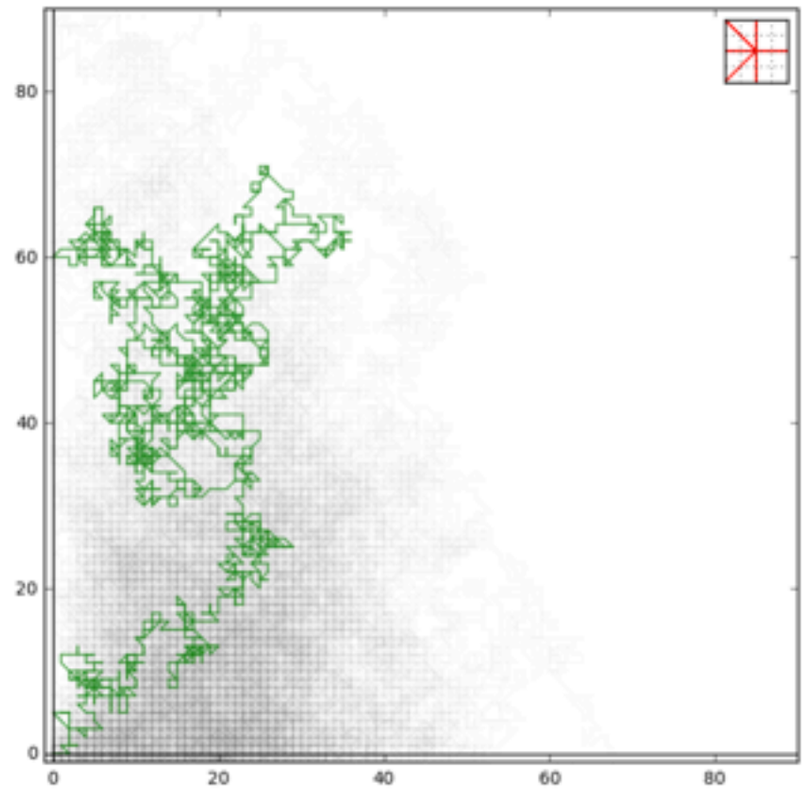
```
In [12]: a_ssp = sel_models[0]
         a_ssp.slope = a_ssp.get_best_slope()
         a_ggwc = GenRGenSWalkCompiler(a_ssp)
         a_u_walks = a_ggwc.generate(100, 2000)
         a_r_walks = filter(is_quarter_plane, a_u_wa
         a_b_walks = filter(lambda w: not is_quarter

In [13]: (afig, aax) = plot_walk_region(a_b_walks +
          200, 200), inset_stepset=a_ssp, inset_loc=
```




```
In [14]: b_ssp = sel_models[1]
         b_ssp.slope = b_ssp.get_best_slope()
         b_ggwc = GenRGenSWalkCompiler(b_ssp)
         b_u_walks = b_ggwc.generate(100, 2000)
         b_r_walks = filter(is_quarter_plane, b_u_wa
         b_b_walks = filter(lambda w: not is_quarter

In [15]: (bfig, bax) = plot_walk_region(b_b_walks + [b_r_walks[3]], borders=(-1,
          90, 90), inset_stepset=b_ssp, inset_loc=1)
```




```
In [16]: c_ssp = sel_models[2]
         c_ssp.slope = c_ssp.get_best_slope()
         c_ggwc = GenRGenSWalkCompiler(c_ssp)
         c_u_walks = c_ggwc.generate(100, 2000)
         c_r_walks = filter(is_quarter_plane, c_u_walks)
         c_b_walks = filter(lambda w: not is_quarter_plane(w), c_u_walks)

In [17]: (cfig, cax) = plot_walk_region(c_b_walks + [c_r_walks[1]], borders=(-1,
          90, 90), inset_stepset=c_ssp, inset_loc=1)
```



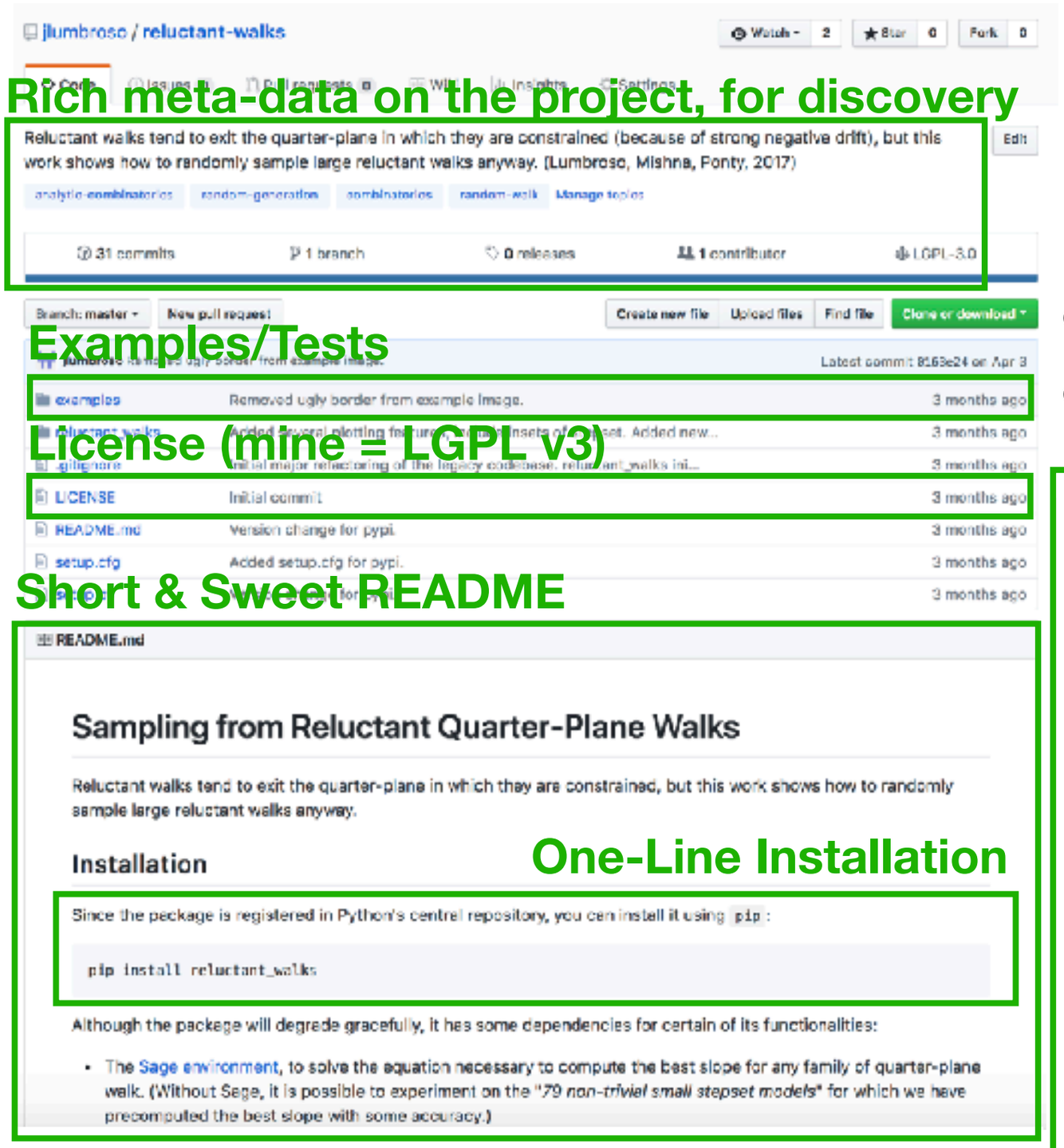

```
In [18]: afig.savefig("example_a.png")
         bfig.savefig("example_b.png")
         cfig.savefig("example_c.png")
```

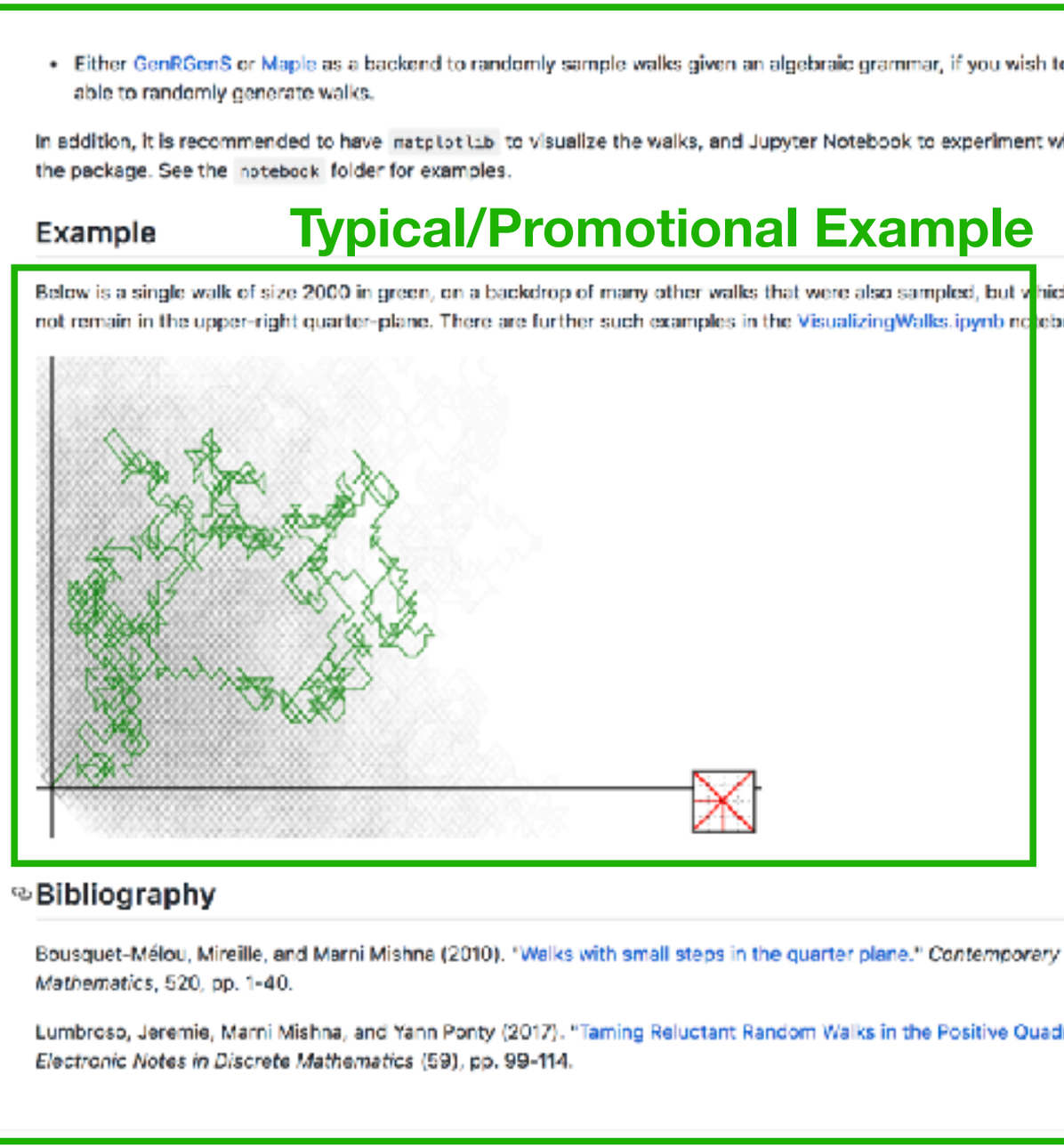
Powered by:

Hoping for a healthier project

- Hosting the project on GitHub
- Other can fork and modify

- Repository self-documented, both for potential/actual users, and future contributors/person who will take-over eventually





Learn by experimentation!

Studies in Automatic Combinatorics

Welcome! | Research Topics | People | Seminars | Software | On-Line Applications
Algo's Publication List | Books | Case Studies

Combinatorics meets computer algebra!

Here is a series of notes that describe interactions between combinatorial analysis, discrete mathematics, and computer algebra. They discuss combinatorial explorations using the Maple system for symbolic computation, in conjunction with packages like *Combstruct*, *Gfun* and *Mgfun* that are described under the topic Library.

- Introductory worksheets
- Volume I (1996)
- Volume II (1997)
- Volume III (2001–2003)

The documents are mostly in the form of Maple Worksheets (mws), Postscript (ps) and Html (html) files.

These pages are maintained by Frédéric Chyzak, Philippe Flajolet, and Bruno Salvy.

Volume III (2001–2003)

- **Special Functions Manipulations**
  - *Borel Resummation of Divergent Series Using Gfun* [Frédéric Chyzak, Marianne Durand, and Bruno Salvy]. For some "irregular singular" problems coming from differential equations, there exist formal power series solutions that are everywhere divergent. These power series turn out to make sense as asymptotic expansions of actual solutions. The Borel summation technique is used to recover convergent representations for these actual functions solutions. For a fairly large class of integrands, this technique leads to algorithmic calculations using Gfun. [mws | ps | html]
    (This session is based on a talk by Lutz at our seminar, of which a summary is also available [ps | pdf].)
  - *An Algolib-aided version of Apéry's proof of the irrationality of zeta(3)* [Bruno Salvy]. This worksheet gives a complete proof of this irrationality. A central part of it has already been discussed in our Volume II (Variations on the Sequence of Apéry Numbers). [mws|ps|html]
  - *A Computer Aided Proof of a Corrected Version of 10.2.32 in Abramowitz & Stegun's HMS* [Frédéric Chyzak]. This work derives a closed-form for the derivative of the modified Bessel function of the first kind, $I_\nu(x)$ with regard to the parameter ν, evaluated at $\nu=1/2$, in terms of exponential integrals. The original formula in the celebrated Handbook of Mathematical Functions has a sign error. Here, we rediscover the correct expression. [mw | ps | pdf]

Volume II (1997)

- **Combinatorics**
  - *Combinatorics of Non-Crossing Configurations* [Frédéric Cazals]. Take points on a circle and consider graphs based on these points such that no edges cross. A fairly complete theory of these constrained random graphs can be developed. Planarity entails a very strong combinatorial decomposability that is especially well suited to a detailed treatment by Combstruct. [mws (138kb) | ps (407kb) | html]
  - *Constrained Permutations and the Principle of Inclusion-Exclusion* [Philippe Flajolet]. This is a Maple worksheet (Maple, version 5.4) based on the Combstruct and Gfun packages. It shows how to enumerate many classes of permutations with constraints on [illegible]

That was the plan all along....

Recall the AofA Motto: "If you can *specify* it..."

- Emphasis on grammars as a first-order tool
- Emphasis on automated theorems

3. *Can* you Specify it?

Rebuilding The Tower Of Babel is a painting by Marcel Flisiuk which was uploaded on January 18th, 2011, fair use.

2000, 500 x 600 mm, Gesso, Acrylics and oil Pencil on Board.
Greg Bridges, Commissioned by Der Speigel Magazine, fair use

Tarek Sebastian Al-shammaa. *Tower of Babel, 2016, Acrylic and oil on canvas, 183 x 184 cm, fair use.*

`reluctant_walks` architecture

(typical output)

- reluctant_walks built on software of community
- that meant producing very different formats of grammar specifications

Different Specs Yesterday

```
TYPE = GRAMMAR

SYMBOLS = LETTERS
```

GenRGenS

```
RULES =
 S -> a S b S;
 S -> c S;
 S -> ;
```

(Ponty, Denise, 2006)

```
{
  S = Union(
        Prod(Prod(a, S), Prod(b, S)),
        Union(Prod(c, S),
              Epsilon)),
  a = Atom,
  b = Atom,
  c = Atom,
}
```

combstruct (Maple/Zimmerman, Mishna, ... 94)

boltzoc (Darrasse, 2010, recently Lumbroso, 2016–)

```
s = sys_new(4);
sys_add_eq(s, 1, sum(prod(
                       prod(ref(2), ref(1)), prod(ref(3),
ref(1))),
                     sum(prod(ref(4), ref(1),
                         epsilon()))));
sys_add_eq(s, 2, atom());
sys_add_eq(s, 3, atom());
sys_add_eq(s, 4, atom());
```

Different Specs Today

```
set zstart 0.01
set min 10
set max 200
set try 50000

Tree ::= Serie + Parallel
Serie ::=  Leaf * <z> + P * P * SEQ(P)
P ::= Parallel + Leaf * <z>
Parallel ::= Leaf * <z> +  S * S * SEQ(S)
S ::= Serie + Leaf * <z>
```

Arbogen
(Peschanski, Dien, 2014)

```
-- Motzkin trees

@module     Sampler
@precision  1.0e-12
@maxiter    30

@withIO     y
@withLists  y
@withShow   y

M = Leaf | Unary M [0.3] | Binary M M.
```

Boltzmann Brain
(Bendkowski, Bodini, Dovgal, 2018)

```
{-# LANGUAGE DeriveDataTypeable #-}

import Test.QuickCheck
import Data.Data
import Boltzmann.Data

data Term = Lambda Int Term | App Term Term | Var Int
  deriving (Show, Data)

instance Arbitrary Term where
  arbitrary = sized $ generatorPWith [positiveInts]

positiveInts :: Alias Gen
positiveInts =
  alias $ \() -> fmap getPositive arbitrary :: Gen Int

main = sample (arbitrary :: Gen Term)
```

BoltzmannSamplers
(Li-yao Xia, 2017)

(The prog. language connections is in large part thanks to Darrasse and Canou, and APR.)

(See RDOS for proof!)

lipn.univ-paris13.fr/rdos/index.php

(2013)

- online website to run our tools without installing them
- was designed before the recent API-craze
- one pitfall is that all tools require very different input parameters

RDOS - Random Discrete Object

Menu

Home

Browse Generators

Links

About/Contact

News

2017-04-02 - NFAGenerator online
A tool for the random generation of non-deterministic automata.

2017-03-31 - Arbogen online
A tool for the random generation of trees using the Boltzmann method is now online.

2013-06-13 - RDOS preview at MAGNUM
First glimpse at the software, hoping to get people onboard!!

Search

Cannot find your generator in the list below?
Please check this page for instructions on submitting your new generato

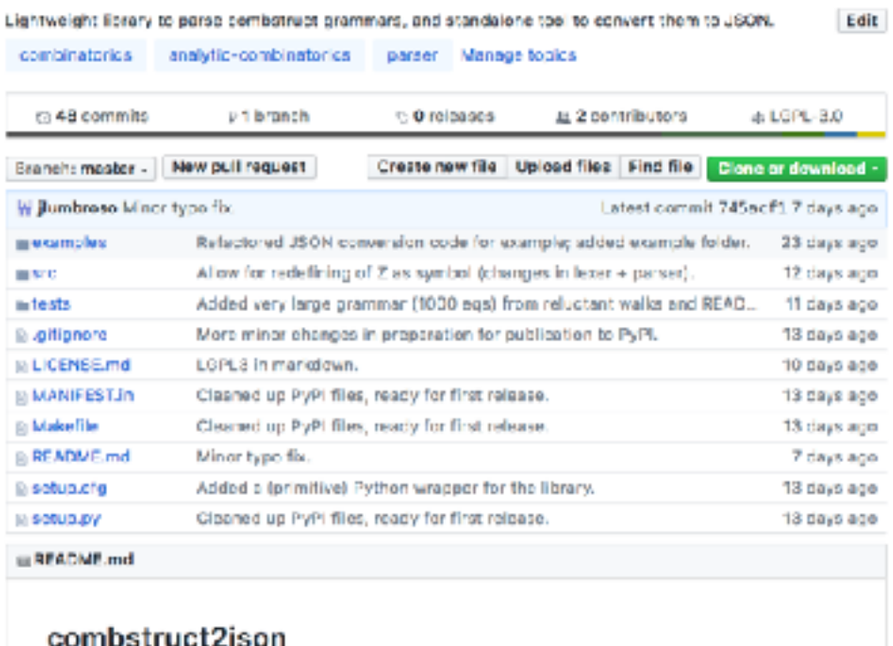
By output

- SEQUENCE
  - GenRGenS - Markov
  - NewtonGF - Tool
- AUTOMATON
  - Regal - Deterministic Automaton
  - DAAS - Acyclic Automata
  - NFAGenerator - Non-deterministic Automaton
- TREE
  - Arbogen - Tool
- WALK
  - Weakly Directed - Walk
  - Weakly Prudent - Walk

combstruct2json (2018)

Using a common input format

- Optimized library written in C
- 5000+ eqs, 115.95 MB only takes 1 min. system on my Macbook
- Wrapper in Python/Sage (next page)




```
$ pip install combstruct2json

$ cat tests/cographs

G  = Set(Co),
Co = Union(Ge, Gc, v, Prod(v,v)),
Ge = Union(Set(Sc, card=2), Prod(Sc,v)),
Gc = Set(Union(v, Sc), card>=3),
Sc = Set(Union(v, C), card>=2),
C  = Set(Union(v, Sc), card>=2),
v  = Atom

$ cat example.py

import combstruct2json
d = combstruct2json.read_file("tests/cographs")
print("Top-level symbols:")
print(d.keys())

$ ./example.py

Top-level symbols:
[u'C', u'Co', u'G', u'Ge', u'Gc', u'v', u'Sc']
```

github.com/jlumbroso/combstruct2json

(2018)

combstruct2json (2018)

Using a common input format

Lightweight library to parse combstruct grammars, and standalone tool to convert them to JSON. Edit

combinatorics analytic-combinatorics parser Manage topics

48 commits 1 branch 0 releases 2 contributors LGPL-3.0

Branch: master · New pull request Create new file Upload files Find file Clone or download ·

jbumbaso Minor typo fix. Latest commit 745acf1 7 days ago

examples	Refactored JSON conversion code for example; added example folder.	23 days ago
src	Allow for redefining of Z as symbol (changes in lexer + parser).	12 days ago
tests	Added very large grammar (1030 eqs) from reluctant walks and READ...	11 days ago
.gitignore	More minor changes in preparation for publication to PyPI.	13 days ago
LICENSE.md	LGPL3 in markdown.	10 days ago
MANIFEST.in	Cleaned up PyPI files, ready for first release.	13 days ago
Makefile	Cleaned up PyPI files, ready for first release.	13 days ago
README.md	Minor typo fix.	7 days ago
setup.cfg	Added a (primitive) Python wrapper for the library.	13 days ago
setup.py	Cleaned up PyPI files, ready for first release.	13 days ago

README.md

combstruct2json

This project provides:

1. A highly optimized, extensible, lightweight linkable library to parse `combstruct` grammars, implemented in C/C++.
2. An independently available Python wrapper that, given a grammar file, outputs a `dict` according to the JSON specification below.
3. A standalone command-line utility that can be piped to other tools (or called as an external system command from a host language such a Ruby, or JavaScript).

The folder `examples` contains usage examples for the linkable library, the folder `tests` contains sample grammars that can be parsed.

This library is already used by projects, such as boltzoo, the fast Analytic Sampler Oracle in C.

Example

```
$ cat tests/cographs
G  = Set(Co),
Co = Union(Ge, Sc, v, Prod(v,v)),
Ge = Union(Set(Sc, card=2), Prod(Sc,v)),
Sc = Set(Union(v, Sc), card>=3),
Se = Set(Union(v, G), card>=2),
C  = Set(Union(v, Sc), card>=2),
v  = Atom
```

would then produce the following JSON output:

```
$ make all
$ ./combstruct2json tests/cographs
{ "G": [ "Type": "Op", "Op": "Set", "param": [{ "Type": "Id", "Id": "Co" }] ], "C
```

which can be prettified, for instance, using Python, for better legibility:

```
$ ./combstruct2json tests/cographs | python -m json.tool | head
{
    "C": {
        "op": "Set",
        "param": [
            {
                "op": "Union",
                "param": [
                    {
                        "id": "v",
                        "type": "id"
                    },
...
```

If you build and install the Python wrapper, you may also read a grammar directly from a Python program:

```
import combstruct2json
d = combstruct2json.read_file("tests/cographs")
print("Top-level symbols:")
print(d.keys())
```

which would print out:

```
Top-level symbols:
[u'C', u'Co', u'E', u'Ge', u'Gc', u'v', u'Sc']
```

Installation

You can build the project from scratch, if you have the necessary dependencies:

1. You may need to install `flex` and `bison`, if you don't already have them.
2. Run `make all` to create the executable `combstruct2json`, the static C/C++ library, and the Python wrapper library.
3. Run `./combstruct2json <filename>` to print the parsed JSON output, from the grammar contained in the given file.

Alternatively, you may also install the Python library directly from PyPI (possibly in your user directory). This step will fetch the latest source after it has been processed by lex/yacc and so does not require they be installed:

```
$ pip install combstruct2json
```

Draft specification of JSON output

Because one purpose to enable easier interoperability with existing work using symbolic specifications in Maple, this library uses Maple's specification for `combstruct` grammars as a starting point.

The output is a JSON string which represents a dictionary mapping symbol names to an abstract syntax tree. Each node of the grammar is represented by a node in the JSON tree:

- For unit elements (with or without a weight), the `type` is `unit`; the available field is `unit` to describe the type of element (protatom, or epsilon). Example:

  ```
  { "type": "unit", "unit": "Epsilon" }
  ```

- For variable references, the `type` is `id`; the available field is `id` which should specify the

```c
#include <Python.h>
#include "../combstruct2json.h"

/*************************************
 Adapted from:
 https://dfm.io/posts/python-c-extensions/
 *************************************/

/* Exception */
static PyObject *Combstruct2JsonError;

/* Docstrings */
static char module_docstring[] =
    "This module provides an interface " +
    "for parsing combstruct grammars.";
static char read_file_docstring[] =
    "Parse the combstruct grammar file " +
    "and return JSON string.";

/* Available functions */
static PyObject *combstruct2json_read_file(
    PyObject *self, PyObject *args);

/* Module specification */
static PyMethodDef module_methods[] = {
    {"read_file", combstruct2json_read_file,
     METH_VARARGS, read_file_docstring},
    {NULL, NULL, 0, NULL}
};

/* Initialize the module */
void initcombstruct2json(void)
{
    PyObject *m = Py_InitModule3("combstruct2json",
                                 module_methods,
                                 module_docstring);

    if (m == NULL)
        return;

    // Initializing our custom exception
    Combstruct2JsonError = PyErr_NewException(
        "combstruct2json.error", NULL, NULL);
    Py_INCREF(Combstruct2JsonError);
    PyModule_AddObject(m, "error", Combstruct2JsonError);
}
```

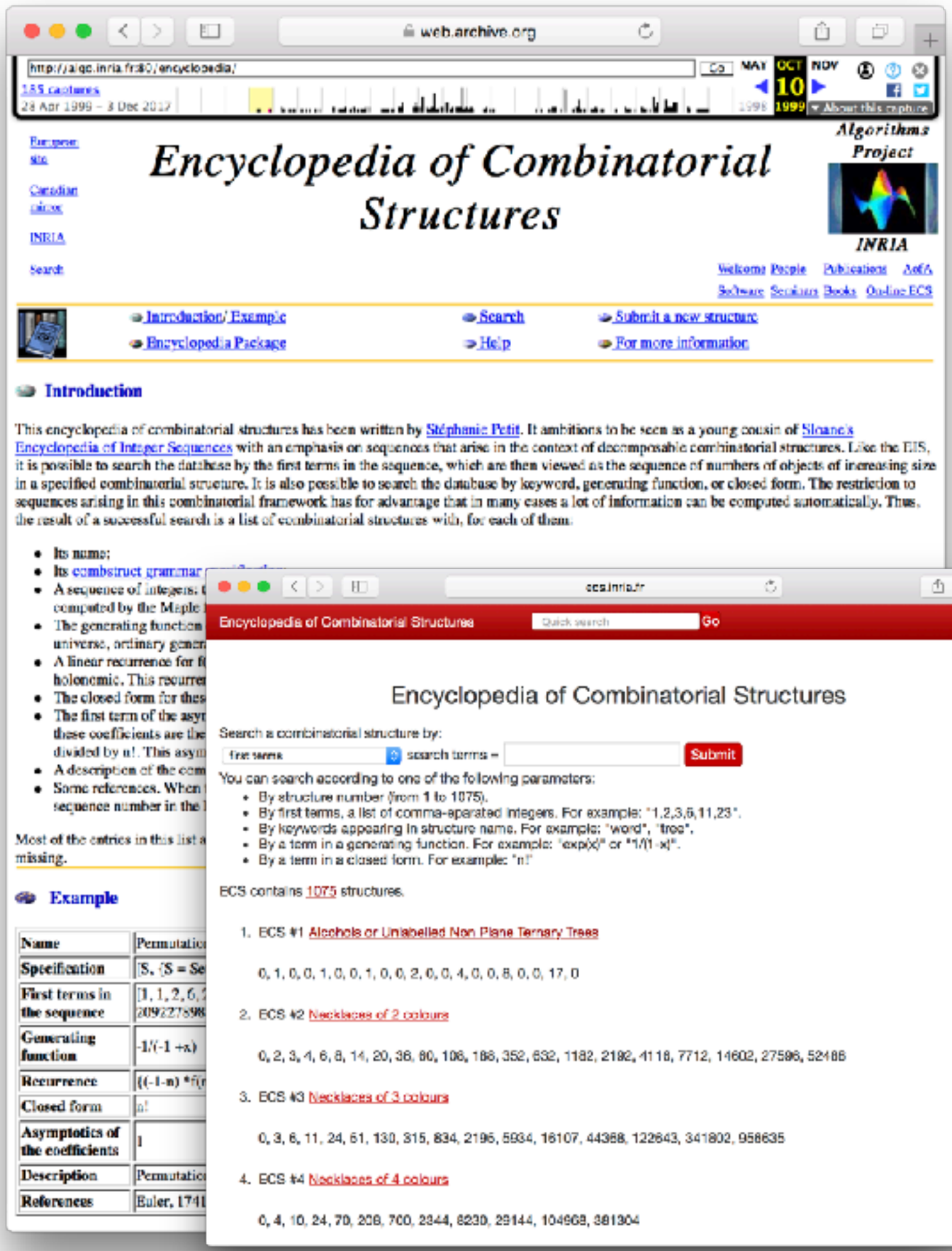
Actual external call

```c
static PyObject *combstruct2json_read_file(PyObject *self,
                                           PyObject *args)
{
    char *arg_filename;

    /* Parse the input tuple */
    if (!PyArg_ParseTuple(args, "s", &arg_filename)) {
        PyErr_SetString(Combstruct2JsonError,
            "Parsing filename for `read_file' failed.");
        return NULL;
    }

    /* Call the external C function to parse the grammar. */
    Grammar* root = readGrammar(arg_filename);

    /* Convert to JSON string. */
    char *ret_jsonstr = root->toJson(root);
    if (ret_jsonstr == NULL) {
        free(root);
        PyErr_SetString(Combstruct2JsonError,
            "Parsing grammar failed for unknown reasons.");
        return NULL;
    }

    /* Build the Python output string. */
    PyObject *py_ret_jsonstr = Py_BuildValue("s", ret_jsonstr);

    /* Run "import json; json.loads(s)" to return dictionary. */
    PyObject* myModuleString = PyString_FromString((char*)"json");
    PyObject* myModule = PyImport_Import(myModuleString);
    PyObject* myFunction = PyObject_GetAttrString(myModule,
                                                  (char*)"loads");
    PyObject* myArgs = PyTuple_Pack(1, py_ret_jsonstr);
    PyObject* py_ret_json = PyObject_CallObject(myFunction,
                                                myArgs);


    /* Clean up. */
    free(root);
    free(ret_jsonstr);

    Py_DECREF(myModuleString);
    Py_DECREF(myModule);
    Py_DECREF(myFunction);
    Py_DECREF(myArgs);

    /* Return output. */
    return py_ret_json;
}
```

github.com/jlumbroso/encyclopedia-of-combinatorial-structures-data

Encyclopedia of Combinatorial Structures

- Is a website
- Is now also a dataset




```
{
  "1": {
    "id": 1,
    "name": "Alcohols or Unlabelled Non Plane
             Ternary Trees",
    "description": "Alcohols or unlabelled non
                    plane ternary Trees",
    "specification": "{S = Union(Z,Prod(Z,Set(S,
                          card = 3))), Z = Atom}",
    "labeled": false,
    "symbol": "S",
    "terms": [
      0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 2,
      0, 0, 4, 0, 0, 8, 0, 0, 17, 0
    ],
    "references": [
      "EIS A000598"
    ]
  },
  ...
```


What is the point of making an effort?

- We have terrific results, which perhaps could have more impact, particular externally

- Results grounded in theory; users of the theory don't necessarily want (or are able) to understand anything about it

- Case in point:

> Unfortunately, that is subsumed by [redacted] in expressiveness and in performance, even though Boltzmann generators theoretically have the best asymptotic complexity.

Summary

- Open-source has dramatically changed: From dominated (by Microsoft, etc.), to dominating (phone OSes, Internet libraries, backend libraries and servers, etc.)

- **reluctant_walks:** a project for random generation of reluctant walks that integrates well with Sage; GitHub repository is a template of one model for sustainable package development in our community

- First set of integrated libraries and tools:
    - **combstruct2json:** a project to unify grammar specification languages
    - **ecs-data:** Encyclopedia of Combinatorial Structures in JSON+combstruct format, to provide robust base dataset in the grammar specification format
    - **boltzoc:** standard oracle for algebraic (tree) grammars
- Tools can be integrated in SAGE (eventual goal) or any other project
- Recommend **a regular session software at every edition of AofA**

And now...