Hansel Documentation

Release 0.0.5a

Sam Nicholls

Mar 12, 2017

Contents

A graph-inspired data structure for determining likely chains of sequences from breadcrumbs of evidence. Brother to Gretel.

CHAPTER 1

Hansel

A graph-inspired data structure for determining likely chains of ordered symbols from breadcrumbs of evidence. Brother to Gretel.

What is it?

Hansel is a probabilistically-weighted, graph-inspired, novel data structure. Hansel is designed to store the number of observed occurrences of a symbol *a* appearing at some position in space or time *i*, co-occurring with another symbol *b* at another position in space or time *j*.

One may traverse along ordered positions in time or space, each time predicting the next most likely symbol of the sequence to traverse to, given the previously selected symbols in the path. Hansel presents a user-friendly API for managing and interacting with the data stored within.

Requirements

```
pip install numpy
```

Install

```
pip install hanselx
```

Citation

Paper pending...

License

Hansel and Gretel are distributed under the MIT license, see LICENSE.

CHAPTER 2

About

What is it for?

Hansel was originally created as a means to store evidence of genetic variation observed across short sequences called *reads*. These reads can be aligned against one another to create longer sequences (contigs). Of interest, are the locations at which reads that overlap exhibit variation from other reads.

We want to recover the chains of symbols (DNA nucleotides) that are most likely to appear from the start to the end of the contig, over the genomic positions that have been demonstrated to vary.

We recognised that **Hansel** had additional potential outside of our use-case, not only to serve as a data structure for creation of future algorithms that want to interact with variation of DNA, but also in other fields entirely, such as computational linguistics.

Hansel can be used where you have a defined set of possible states or symbols that occur in time or space. For example:

- DNA, RNA or amino acids, over a sequence
- Words in order, from a book, or page
- States that occur in a simple ordered machine or system
- User or actor actions that occur over time

How is data stored?

Hansel is a four dimensional matrix. An element *H[a, b, i, j]* record the number of observations of a co-occurring pair of symbols a and b at positions i and j respectively. At first this structure may appear limited, but the data in H can easily be exploited to build other structures.

How does it work?

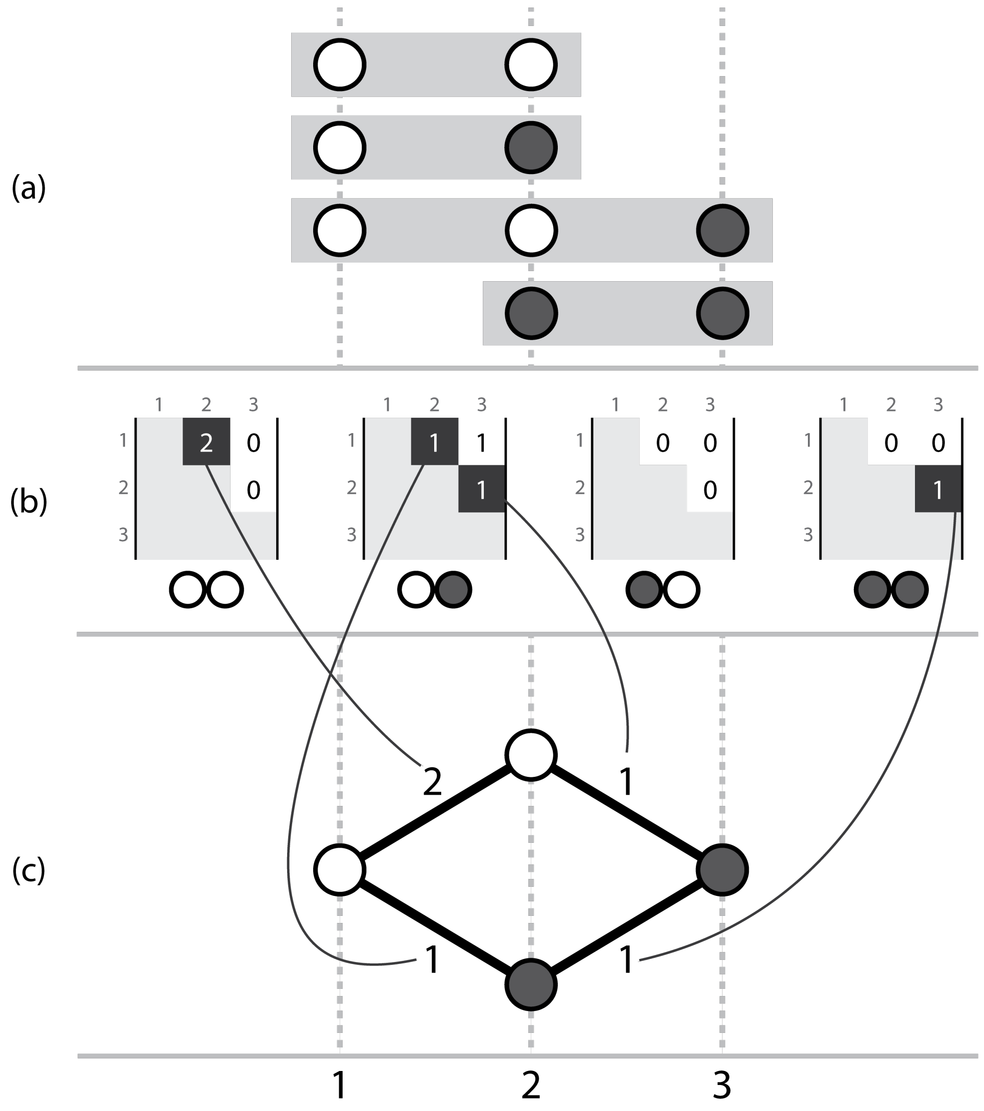

Fig. 2.1: Three corresponding representations, (a) a set of short ordered sequences, with symbols, (b) the actual Hansel structure where each possible pair of symbols (00, 01, 10, 11) has a matrix storing counts of occurrences of that ordered symbol pair between two positions across all of the aligned sequences in a, (c) a simple graph that can be constructed by considering the evidence provided by adjacent variant symbols. Note this representation ignores evidence from non-adjacent pairs, which is overcome by the dynamic edge weighting of the Hansel data structure's interface.

CHAPTER 3

Usage

Adding Observations

Simple

To construct and add observations to a **Hansel** data structure:

```
from hansel import Hansel
import numpy as np

symbols = ['A', 'C', 'G', 'T', '_']
unsymbols = ['_']
positions = [1, 3, 5, 7, 9]
L = 10 # Defines the 'lookback'

# Get some memory and pass it to the Hansel constructor
a = np.ndarray((
    len(symbols), len(symbols), len(positions)+2, len(positions)+2)
)
hansel = Hansel(a, symbols, unsymbols, L=L)

# Add some observations
hansel.add_observation(a, b, i, j)
# ...
```

Not so Simple

For very large data sets, or complicated parallel high-throughput methodologies, you may need to bypass the API for adding observations by reserving and populating the relevant memory yourself:

```
import ctypes
import numpy as np
```

```
from hansel import Hansel

symbols = ['A', 'C', 'G', 'T', '_']
unsymbols = ['_']
positions = [1, 3, 5, 7, 9]
L = 10 # Defines the 'lookback'

# Get some memory
h = np.frombuffer(Array(ctypes.c_float,
        (len(symbols)**2) * ((len(positions)+2)**2),
        lock=False),
dtype=ctypes.c_float)

# Shape the memory into a numpy array of the desired size
h = hansel.reshape(len(symbols), len(symbols), len(positions)+2, len(positions)+2)
h.fill(0.0)

# Add some observations
def __symbol_num(symbol):
    symbols_d = {symbol: i for i, symbol in enumerate(symbols)}
    return symbols_d[symbol]
h[__symbol_num(a), __symbol_num(b), i, j] += 1
# ...

# Feed the prefilled array to the Hansel constructor
hansel = Hansel(h, symbols, unsymbols, L=L)
```

Get Observation Counts

To find the number of times symbol *a* at position *i* has been seen with symbol *b* at position *j*:

```
hansel.get_observation(a, b, i, j)
```

Summarise Counts at Position

To fetch a dictionary of the raw counts for each symbol at a given position:

```
hansel.get_counts_at(at_position)
```

Marginal Distribution

To find the *log10* probability of a particular symbol appearing at a given position:

```
hansel.get_marginal_of_at(interesting_symbol, at_position)
```

Conditional Distribution

To find the *log10* probability of *a* at *i* appearing with *b* and *j*:

```
hansel.get_conditional_of_at(a, b, i, j)
```

Get Spanning Support

Get the number of times a symbol or state *b* appears at position *j*, on pieces of evidence that also covered space or time point *i*:

```
hansel.get_spanning_support(b, i, j)
```

Get Edge Weights

Given a sequence of symbols selected during traversal thus far, find the *log10* probabilities of traversing to the available symbols at your next position *j*:

```
hansel.get_edge_weights_at(j, current_path)
```

Reweight Observations

Reduce the element that support the observation of *a* at *i* and *b* at *j* co-occurring on the same piece of evidence together:

```
hansel.reweight_observation(a, b, i, j, ratio)
```

The original observation count is multiplied by the ratio, the result is then subtracted from the current value. It is recommended that *ratio* not be too large without good confidence. Aggressive reweighting can lead to spending (removing) the evidence in the Hansel matrix before your algorithm has had time to explore the paths properly.

CHAPTER 4

History

0.0.7

- Documentation

0.0.6

- Switch to dictionaries and sets generated at construction time to speed up the lookup of symbols

0.0.5

- Introduce *unsymbols* to cover cases where you would like to count observations from some *a* to an “invalid” *b* during marginal calculation, but still prevent the actual selection of the invalid *b* as a transition choice
- Member list *unsymbols* keeps track of symbols who should have no weighting when counting observations or calculating marginals.
- Allow a user to construct Hansel with an argument for the lookback order.
- Alter *get_spanning_support* and *get_counts_at* to not count observations that originate from an ignored unsymbol and transition to a real symbol. This alteration makes things work even if you’ve been a silly and filled Hansel with more crummy data than usual, hooray.
- Dramatically improve performance by using the correct ndarray indexing syntax [A, B, i, j] vs. [A][B][i][j]

0.0.4

- Add some documentation.

- Rename *get_marginal_at* to *get_counts_at*. As the function returns raw counts, not a marginal distribution, this is a less misleading name.
- Don't return the unused *curr_branches_tot* value from *get_edge_weights_at*.
- Remove *select_next_edge_at*, we need not concern ourselves with problem specific end-user behaviour. We just provide an API to the pseudo-graph.

0.0.3

- Add *observations* property for those who may find *crumbs* confusing or odd.
- Remove domain specific language ("SNP", "mallele") in favour of "symbol".
- Require symbol list on constuction, prevent empty list with casting/template.
- Ensure to catch an in-progress __new__ in __array_finalize__

0.0.2

- Abstract BAM specific loading to *gretel*.
- Rename *reads* attribute to *slices* (of bread)
- Add *sources* property for those who may find *slices* confusing or bizarre.

0.0.1

- Import repository from *claw*.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search