Software Tools for Mathematics

Discreture

A **C++** library to generate simple combinatorial objects

Miguel Raggi
mraggi@gmail.com

Escuela Nacional de Estudios Superiores
UNAM

January 26, 2018

Index

Index

What is discreture?

Definition

*Discreture is an open source C++ library to produce combinatorial objects like **combinations**, **set-partitions**, **motzkin paths**, etc.*

https://github.com/mraggi/discreture

What is discreture?

Definition

Discreture is an open source C++ library to produce combinatorial objects like ***combinations***, ***set-partitions***, ***motzkin paths***, *etc.*

`https://github.com/mraggi/discreture`

- You can use it for (pretty much) any purpose (Apache license).

What is discreture?

Definition

*Discreture is an open source C++ library to produce combinatorial objects like **combinations**, **set-partitions**, **motzkin paths**, etc.*

https://github.com/mraggi/discreture

- You can use it for (pretty much) any purpose (Apache license).
- It has a focus on ease of use, compatibility with the STL, speed and correctness

What is discreture?

Definition

Discreture is an open source C++ library to produce combinatorial objects like ***combinations****,* ***set-partitions****,* ***motzkin paths****, etc.*

https://github.com/mraggi/discreture

- You can use it for (pretty much) any purpose (Apache license).
- It has a focus on ease of use, compatibility with the STL, speed and correctness (not necessarily in that order).

What is discreture?

Definition

Discreture is an open source C++ library to produce combinatorial objects like ***combinations****,* ***set-partitions****,* ***motzkin paths****, etc.*

`https://github.com/mraggi/discreture`

- You can use it for (pretty much) any purpose (Apache license).
- It has a focus on ease of use, compatibility with the STL, speed and correctness (not necessarily in that order).
- Contributor: Manuel Alejandro Romo de Vivar.

Motivation

- Suppose you wish to do something with every pair of objects in a set

Motivation

- Suppose you wish to do something with every pair of objects in a set
- Easy:

```
int n = 20;
for (int i = 0; i < n; ++i)
{
  for (int j = i+1; j < n; ++j)
  {
    // Do stuff
  }
}
```

Motivation

- Now, what if you want all triples?
- Easy: just add another for loop.

Motivation

- Now, what if you want all triples?
- Easy: just add another for loop.
- Quadruples? Quintuples?

Quintuples?

```
int n = 20;
for (int i = 0; i < n; ++i)
{
  for (int j = i+1; j < n; ++j)
  {
    for (int k = j+1; j < n; ++k)
    {
      for (int l = k+1; l < n; ++l)
      {
        for (int m = l+1; m < n; ++m)
        {
          // Do stuff
        }
      }
    }
  }
}
```

A better solution

This is what it looks like in discreture:

```
for (auto& x : combinations(20,5))
{
    // Do stuff with x[0], x[1], ...
}
```

Not just indices

Or maybe:

```
std::vector<MyObject> A;

//... fill A somehow

for (auto x : compound_combinations(A,5))
{
    // x[i] has type MyObject&
}
```

Combinatorial objects

- Combinations. Subsets of a specific size:
  - Example: $\{0,3,4\},\{0,1,5\} \in$ combinations(6,3)

Combinatorial objects

- Combinations. Subsets of a specific size:
  - Example: $\{0,3,4\},\{0,1,5\} \in$ combinations(6,3)
- Permutations. S_n
  - Example: [0,1,2], [2,0,1] $\in$ permutations(3)

Combinatorial objects

- Combinations. Subsets of a specific size:
  - Example: $\{0,3,4\},\{0,1,5\} \in$ combinations(6,3)
- Permutations. S_n
  - Example: [0,1,2], [2,0,1] $\in$ permutations(3)
- Partitions. Numbers that add up to a given number.
  - Example: $\{6,4,1\},\{3,3,3,1,1\} \in$ partitions(11)

Combinatorial objects

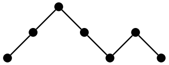
- Combinations. Subsets of a specific size:
  - Example: $\{0,3,4\},\{0,1,5\} \in$ combinations(6,3)
- Permutations. S_n
  - Example: [0,1,2], [2,0,1] $\in$ permutations(3)
- Partitions. Numbers that add up to a given number.
  - Example: $\{6,4,1\},\{3,3,3,1,1\} \in$ partitions(11)
- Set Partitions. Partitions of $\{0,...,n-1\}$ into disjoint sets.
  - Example: $\{[0,2],[1,3]\} \in$ set_partitions(4)

Combinatorial objects

- Combinations. Subsets of a specific size:
  - Example: $\{0,3,4\},\{0,1,5\} \in$ combinations(6,3)
- Permutations. S_n
  - Example: [0,1,2], [2,0,1] $\in$ permutations(3)
- Partitions. Numbers that add up to a given number.
  - Example: $\{6,4,1\},\{3,3,3,1,1\} \in$ partitions(11)
- Set Partitions. Partitions of $\{0,...,n-1\}$ into disjoint sets.
  - Example: $\{[0,2],[1,3]\} \in$ set_partitions(4)
- Multisets. How many to take of each index? (CW $\leq$)
  - Example: [2,1,3], [0,1,1] $\in$ multisets([3,1,3])

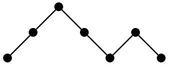
Combinatorial objects

- Combinations. Subsets of a specific size:
  - Example: $\{0,3,4\},\{0,1,5\} \in$ combinations(6,3)
- Permutations. S_n
  - Example: [0,1,2], [2,0,1] $\in$ permutations(3)
- Partitions. Numbers that add up to a given number.
  - Example: $\{6,4,1\},\{3,3,3,1,1\} \in$ partitions(11)
- Set Partitions. Partitions of $\{0,...,n-1\}$ into disjoint sets.
  - Example: $\{[0,2],[1,3]\} \in$ set_partitions(4)
- Multisets. How many to take of each index? (CW $\leq$)
  - Example: [2,1,3], [0,1,1] $\in$ multisets([3,1,3])
- Dyck Paths. From $(0,0)$ to $(2n,0)$ but y is never negative and always goes either up or down.

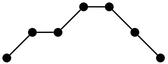
 $\in$ dyck_paths(3)

Combinatorial objects

- Combinations. Subsets of a specific size:
  - Example: $\{0, 3, 4\}, \{0, 1, 5\} \in$ combinations(6,3)
- Permutations. S_n
  - Example: [0,1,2], [2,0,1] $\in$ permutations(3)
- Partitions. Numbers that add up to a given number.
  - Example: $\{6, 4, 1\}, \{3, 3, 3, 1, 1\} \in$ partitions(11)
- Set Partitions. Partitions of $\{0, ..., n-1\}$ into disjoint sets.
  - Example: $\{[0, 2], [1, 3]\} \in$ set_partitions(4)
- Multisets. How many to take of each index? (CW $\leq$)
  - Example: [2,1,3], [0,1,1] $\in$ multisets([3,1,3])
- Dyck Paths. From $(0, 0)$ to $(2n, 0)$ but y is never negative and always goes either up or down.

 $\in$ dyck_paths(3)

- Motzkin Paths. Same, but you can go horizontally

 $\in$ motzkin_paths(6)

Index

How to install?

- **Header-only library**, so no need to install anything (just copy the files to your own project, or anywhere your compiler knows to look for header files).
- It does need a somewhat modern C++14 compiler and boost.

Basic use

To iterate over an object, use standard `C++` range-based for loop:

Basic use

To iterate over an object, use standard `C++` range-based for loop:

```
#include <iostream>
#include "discreture.hpp" //includes everything

using namespace std;
using namespace dscr;

int main()
{
  for (auto& x : combinations(6,3))
    cout << x << endl;
}
```

Basic use

To iterate over an object, use standard C++ range-based for loop:

```
#include <iostream>
#include "discreture.hpp" //includes everything

using namespace std;
using namespace dscr;

int main()
{
  for (auto& x : combinations(6,3))
    cout << x << endl;
}
```

Produces (without the brackets):

[0 1 2] [0 1 3] [0 2 3] [1 2 3] [0 1 4]
[0 2 4] [1 2 4] [0 3 4] [1 3 4] [2 3 4]

Another example:

```
for (auto& x : partitions(5))
    cout << x << endl;
```

This just prints all ways of adding up to 5 with positive integers:

```
[ 1 1 1 1 1 ]
[ 2 1 1 1 ]
[ 3 1 1 ]
[ 2 2 1 ]
[ 4 1 ]
[ 3 2 ]
[ 5 ]
```

Random Access Iterators

Combinations, Permutations and Multisets are "random access" containers:

```
permutations X(12);
cout << X[157122128] << endl;
```

(Instantly) Prints `[3 11 2 10 9 0 4 1 6 7 5 8]`, which is the 157,122,128-th permutation.

Random Access Iterators

Combinations, Permutations and Multisets are “random access” containers:

```
permutations X(12);
cout << X[157122128] << endl;
```

(Instantly) Prints `[3 11 2 10 9 0 4 1 6 7 5 8]`, which is the 157,122,128-th permutation.

This allows for pretty easy multi-threaded programs (see included examples).

Dyck and Motzkin

We can easily generate all correct ways of placing parenthesis:

```
dyck_paths X(3);
for (auto& x : X)
  cout << dyck_paths::to_string(x, "()") << endl;
```

which prints: ((()))   (()())   ()(())   (())()   ()()()

Algorithms

We can use standard C++ STL algorithms:

```
motzkin_paths X(10);
std::find_if(X.begin(), X.end(), condition);
```

finds the first motzkin path that satisfies a certain condition.

Standard algorithms

You can even do binary search:

```
// This has almost 10^13 combinations (!)
combinations X(46,23);

auto it = std::partition_point(X.begin(), X.end(),
  [](const auto& x) {
    return x.back() < 36;
  });
cout << *it << endl;
```

find_all

Combinations can even do branch-and-cut:

find_all

Combinations can even do branch-and-cut:

```
bool no_consecutive(const combination& x)
{
  int k = x.size();
  return k <= 1 || x[k-2]+1 != x[k-1];
}

// ...

  combinations X(10,5);
  for ( auto& x : X.find_all(no_consecutive) )
    cout << x << endl;
```

Prints `[ 0 2 4 6 8 ] [ 0 2 4 6 9 ] [ 0 2 4 7 9 ] ...`, which are combinations that don't have two consecutive numbers.

Index

Benchmarks

Go to Benchmarks

How does it compare?

How does it compare?

To generate all combinations in GSL (GNU scientific library), you do:

```
gsl_combination* c = gsl_combination_calloc(6, 3);
do
{
  // gsl_combination_get(c,i) to obtain the
  // i-th index
} while (gsl_combination_next(c) == GSL_SUCCESS);
gsl_combination_free(c);
```

Easier!

```
for (auto& x : combinations(6,3))
{
    // x[i] to access
    // i-th index
}
```

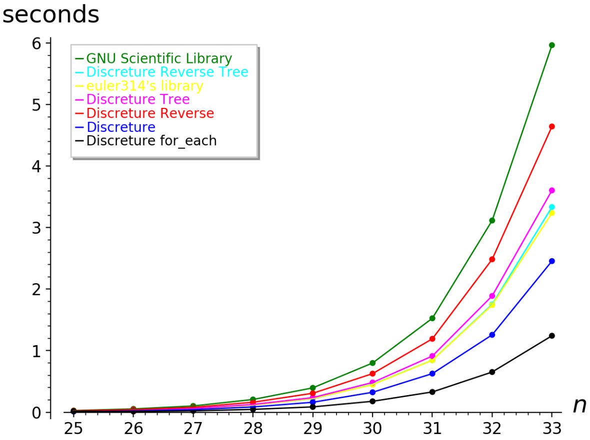
What about speed?

What about speed?

Time to iterate over all $\binom{n}{\lfloor n/2 \rfloor}$

What about speed?

Time to iterate over all $\binom{n}{\lfloor n/2 \rfloor}$



Comparisons with sage/gap?

- `Sage` and `GAP` also have many of the combinatorial objects mentioned.

Comparisons with sage/gap?

- `Sage` and `GAP` also have many of the combinatorial objects mentioned.
- It wouldn't be fair to compare them to discreture (C++ vs python).

Comparisons with sage/gap?

- `Sage` and `GAP` also have many of the combinatorial objects mentioned.
- It wouldn't be fair to compare them to discreture (C++ vs python).
- For example, in my computer, to see all $\binom{26}{13}$ combinations,

Comparisons with sage/gap?

- `Sage` and `GAP` also have many of the combinatorial objects mentioned.
- It wouldn't be fair to compare them to discreture (C++ vs python).
- For example, in my computer, to see all $\binom{26}{13}$ combinations,
  - `Sage` takes $\approx$34s.

Comparisons with sage/gap?

- `Sage` and `GAP` also have many of the combinatorial objects mentioned.
- It wouldn't be fair to compare them to discreture (C++ vs python).
- For example, in my computer, to see all $\binom{26}{13}$ combinations,
  - `Sage` takes $\approx$34s.
  - `GAP` crashes after a while for lack of memory :(

Comparisons with sage/gap?

- `Sage` and `GAP` also have many of the combinatorial objects mentioned.
- It wouldn't be fair to compare them to discreture (C++ vs python).
- For example, in my computer, to see all $\binom{26}{13}$ combinations,
  - `Sage` takes ≈34s.
  - `GAP` crashes after a while for lack of memory :(Increasing memory limit makes it take ≈33s.

Comparisons with sage/gap?

- `Sage` and `GAP` also have many of the combinatorial objects mentioned.
- It wouldn't be fair to compare them to discreture (C++ vs python).
- For example, in my computer, to see all $\binom{26}{13}$ combinations,
  - `Sage` takes $\approx$34s.
  - `GAP` crashes after a while for lack of memory :(Increasing memory limit makes it take $\approx$33s.
- Discreture takes $\approx$ 0s.

Index

Contributing

Container	Forward	Reverse	Random Access
Combinations	Yes	Yes	Yes
Permutations	Yes	Yes	Yes
Multisets	Yes	Yes	Yes
Dyck Paths	Yes	No	No
Motzkin Paths	Yes	No	No
Partitions	Yes	Yes	No
Set Partitions	Yes	No	No
Compositions	No	No	No
Graphs (nauty?)	No	No	No
Others?	No	No	No

IF there is time, let's see the solution to Jose Hernández's Problem.

Go to Jose's Problem

Thank you!

github.com/mraggi/discreture