

Mini-course on representation theoretical methods in ML

5. Clausen's FFT

Risi Kondor (Gatsby Unit)

Define the **contiguous cycle**

$$[i, j] = (i, i+1, i+2, \dots, j)$$

The cosets $[1, n]\mathbb{S}_{n-1}, [2, n]\mathbb{S}_{n-1}, \dots, [n, n]\mathbb{S}_{n-1}$
form a partition of \mathbb{S}_n

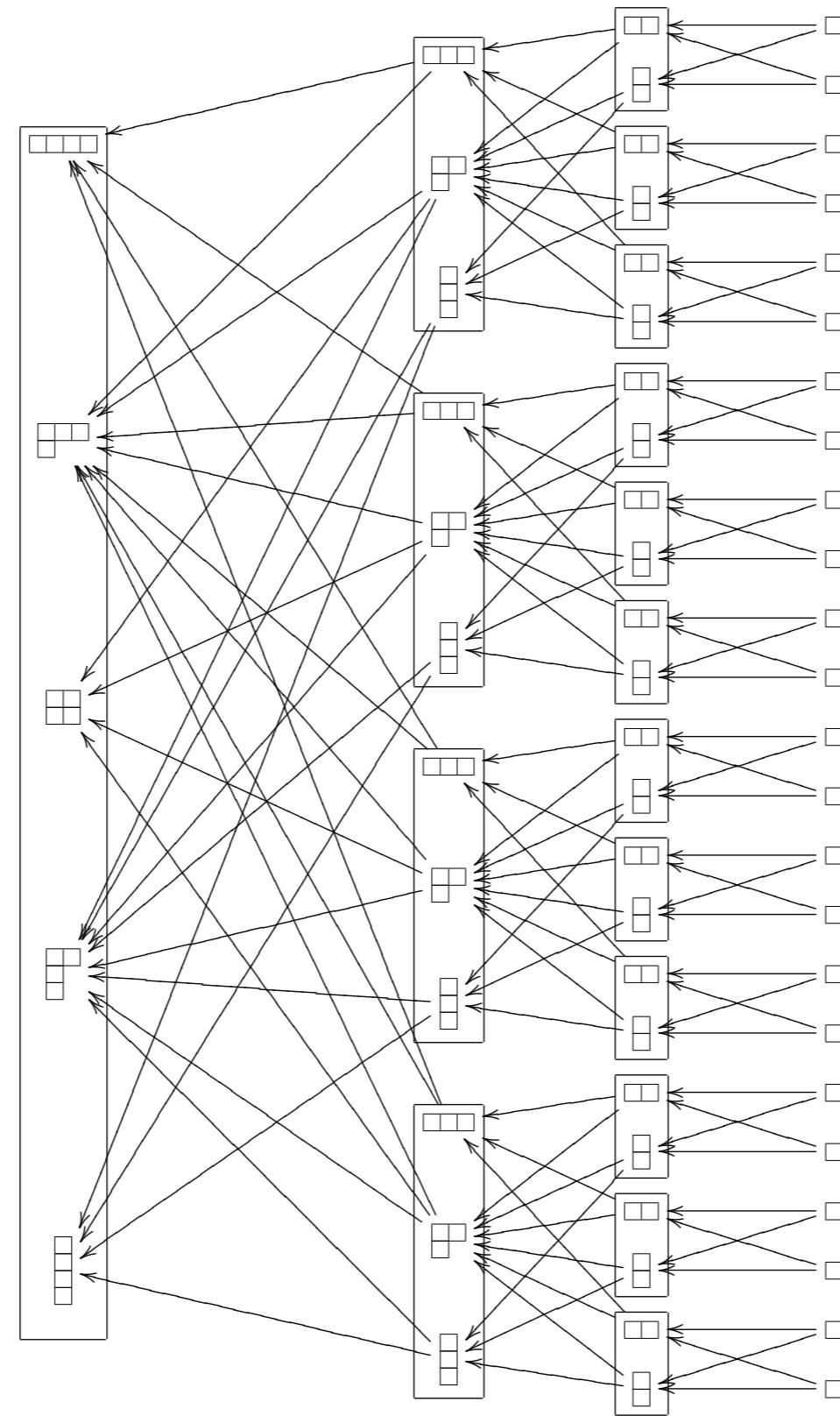
Idea: decompose FT over \mathbb{S}_n into n smaller
FTs over the $[i, n]\mathbb{S}_{n-1}$ cosets.

$$\widehat{f}(\lambda) = \sum_{\sigma \in \mathbb{S}_n} \rho_\lambda(\sigma) \, f(\sigma)$$

$$= \sum_{i=1}^n \sum_{\sigma' \in \mathbb{S}_{n-1}} \rho_\lambda(\llbracket i,n \rrbracket \sigma') \; f(\llbracket i,n \rrbracket \sigma')$$

$$= \sum_{i=1}^n \rho_\lambda(\llbracket i,n \rrbracket) \sum_{\sigma' \in \mathbb{S}_{n-1}} \rho_\lambda(\sigma') \; f_{\llbracket i,n \rrbracket \mathbb{S}_{n-1}}(\sigma')$$

$$= \sum_{i=1}^n \rho_\lambda(\llbracket i,n \rrbracket) \bigoplus_{\begin{array}{c} \lambda^- \vdash n-1 \\ \lambda^- < \lambda \end{array}} \widehat{f}_{\llbracket i,n \rrbracket \mathbb{S}_{n-1}}(\lambda^-)$$



Op count

- $\llbracket i, k \rrbracket$ is a product of $k - i$ adjacent transpositions
- Computing $\rho_\lambda(\llbracket i, j \rrbracket) \cdot M$ takes $2 d_\lambda^2$ time
- For $\lambda \vdash k$ $d_\lambda^2 = k!$
- Layer k has $n!/k!$ Fourier transforms.

$$\sum_{k=1}^n \sum_{i=1}^k 2(k-i)n! = n! \frac{(n+1)n(n-1)}{3}$$

S_n ob

Snob

A C++ library for fast Fourier transforms on the symmetric group.

author: Risi Kondor, Columbia University (risi@cs.columbia.edu)

Development version as of August 23, 2006 (unstable!):

Documentation: [\[ps\]](#) [\[pdf\]](#)

C++ source code: [\[directory\]](#)

BiBTeX entry: [\[bib\]](#)

Entire package: [\[tar.gz\]](#)

ALL SOFTWARE ON THIS PAGE IS DISTRIBUTED UNDER THE TERMS OF THE GNU
GENERAL PUBLIC LICENSE [\[site\]](#)

References:

1. Michael Clausen: **Fast generalized Fourier transforms**. Theoretical Computer Science **67**(1): 55-63, 1989.
2. David K. Maslen and Daniel N. Rockmore: **Generalized FFTs --- a survey of some recent results**. Proceedings of the DIMACS Workshop on Groups and Computation, 1997. [\[ps\]](#)
3. K.-J. Kueh, T. Olson, D. Rockmore and K.-S. Tan: **Nonlinear approximation theory on finite**

<http://www.cs.columbia.edu/~risi/SnOB>

```
#include <vector>

#include "base.h"
#include "Matrix.hpp"
#include <iostream>
#include "Sn.hpp"
#include "SnFunction.hpp"
#include "StandardTableau.hpp"

using namespace std;

class Sn::FourierTransform: FiniteGroup::FourierTransform{

public:

    friend class Sn::Function;
    friend class Sn::Ftree;

    FourierTransform(const Sn& _group);
    FourierTransform(const Sn& _group, int dummy):group(&_group),n(_group.n){};
    FourierTransform(const Sn& _group, const vector<Matrix<FIELD>*> matrices);
    FourierTransform(const Function& f);
    ~FourierTransform();

    Function* iFFT() const;

    FIELD operator()(const StandardTableau& t1, const StandardTableau& t2) const;

    double norm2() const {double result; for(int i=0; i<matrix.size(); i++) result+=1; return result;}

    string str() const;

    vector<Matrix<FIELD>*> matrix;

private:

    void fft(const Sn::Function& f, const int offset);
    void ifft(Sn::Function* target, const int _offset) const;

    const int n;
    const Sn* group;

};


```

Sn::Irreducible

Represents an irreducible representation ρ_λ of S_n .

Parent class: FiniteGroup::Irreducible

CONSTRUCTORS

`Irreducible(Sn* G, Partition& lambda)`

Construct the irreducible representation of the symmetric group G corresponding to the partition λ .

MEMBER FUNCTIONS

`Matrix<FIELD>* rho(const Sn::Element& sigma)`

Returns $\rho(\sigma)$, the representation matrix of permutation σ in Young's orthogonal representation.

`FIELD character(const Partition& mu)`

Returns $\chi(\mu)$, the character of this representation at permutations of cycle type μ .

`void computeTableaux()`

Compute the standard tableaux of this irreducible if they have not already been computed. Because this is an expensive operation, it is postponed until some function is called (such as `rho` or `character`) which requires the tableaux of this particular irreducible. `computeTableaux()` is called automatically by these functions, and once the tableaux have been computed they are stored for the lifetime of the Irreducible.

`StandardTableau* tableau(const int t)`

Return a new standard tableaux of index t . This works even if `tableauV` has not been computed.

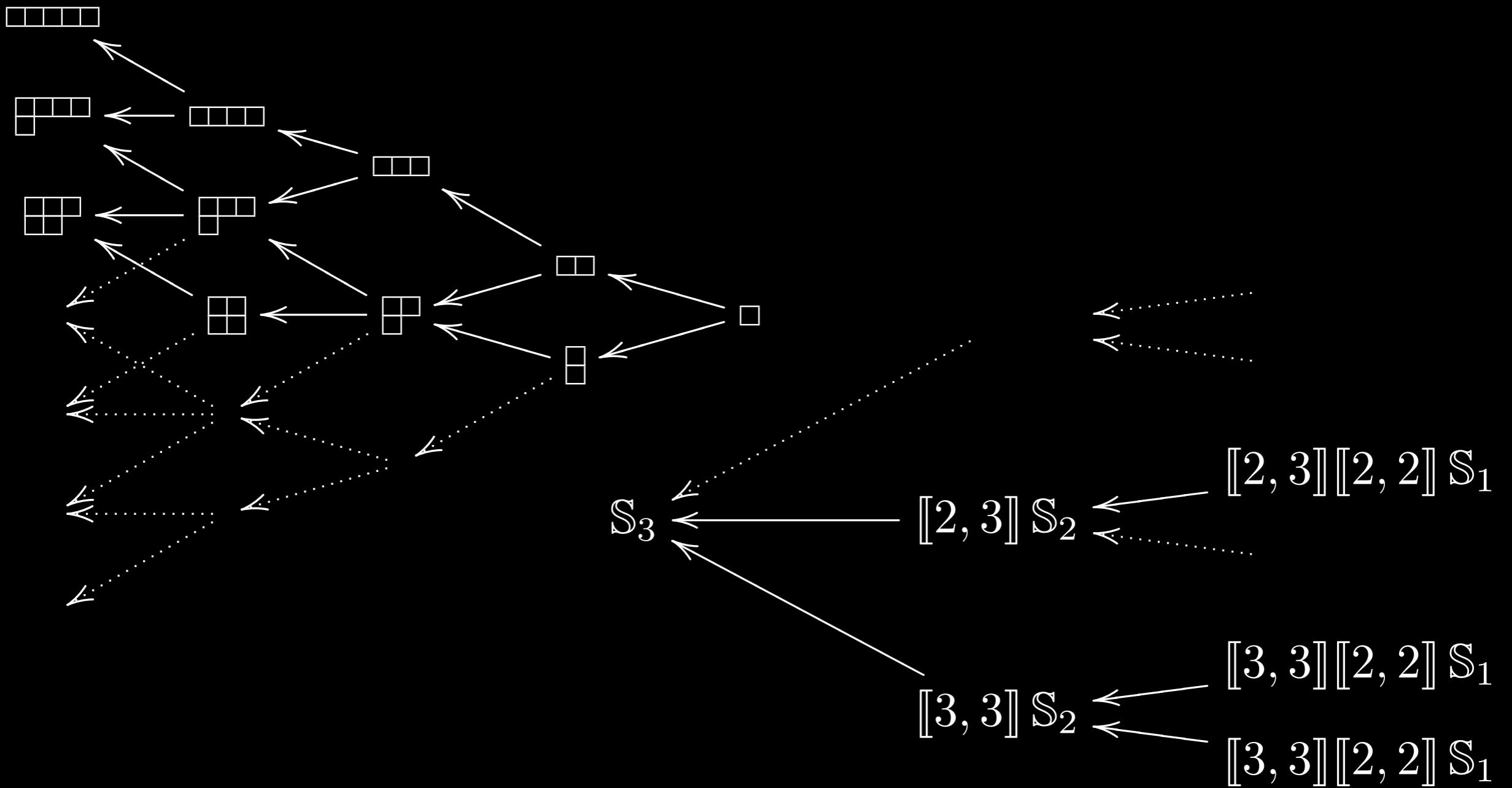
`void computeYOR()`

Compute and store the coefficients (2.5) and (2.6) in Young's orthogonal representation for all adjacent transpositions τ_k and all tableau t of shape λ . Because this is an expensive operation, these coefficients are not normally computed until they are demanded by functions such as `rho` or `character`. `computeYOR()` is called automatically by these functions, and once the tableaux have been computed they are stored for the lifetime of the Irreducible. `computeYOR()` also requires the tableaux, so it calls `computeTableaux()` if those have not been computed yet.

`void applyCycle(const int j, Matrix<FIELD>& M[, int m])`

`void applyCycle(const int j, Matrix<FIELD>& M[, int m])`

Sparse transforms



Twisted transforms

Restrict to two sided cosets $\sigma_L \mathbb{S}_k \sigma_R$