Tree sweeping algorithms with ITensorNetworks.jl

Lander Burgelman

GHENT UNIVERSITY

Outline

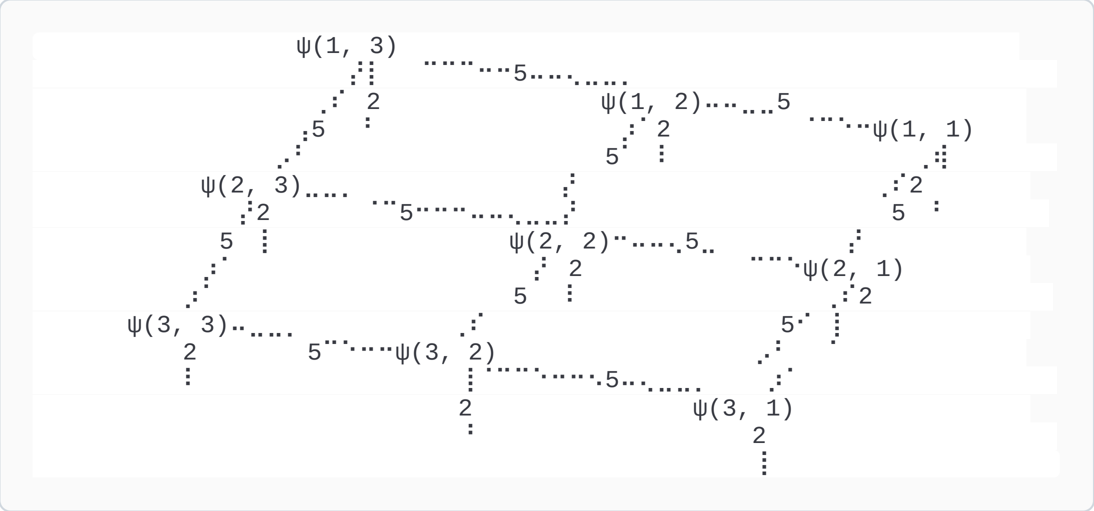
1. ITensorNetworks.jl?
2. Algorithms and challenges
3. Tree sweeping algorithms
4. Implementation highlights
5. Future directions

ITensorNetworks.jl?

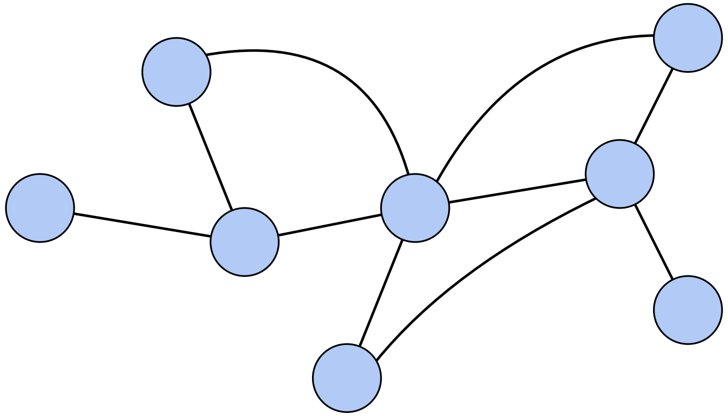
mtfishman / **ITensorNetworks.jl**

- ITensorNetwork.jl: simple and intuitive interface for general network tools
- Developers: **Matt Fishman**, Joseph Tindall, Linjian Ma, Miles Stoudenmire, LB

```julia
using NamedGraphs
using ITensors
using ITensorNetworks
using ITensorUnicodePlots

c = named_grid((3, 3))
s = siteinds("S=1/2", c)
ψ = ITensorNetwork(s; link_space=5)
@visualize ψ;
```



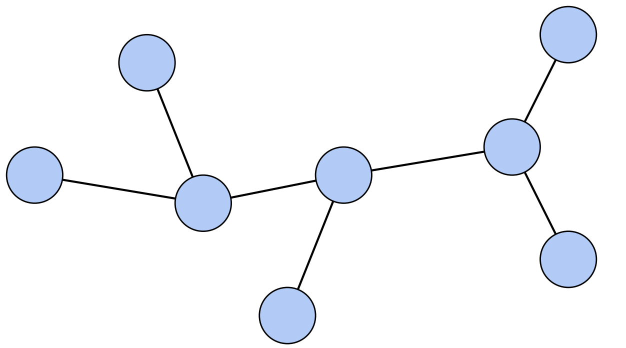
Algorithms and challenges

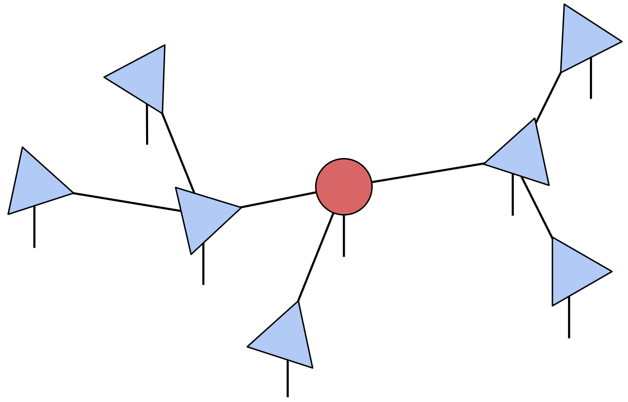


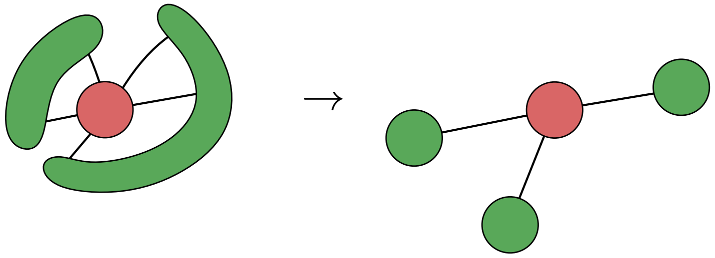
- Common algorithms:
  - Cost function optimization
  - Ground state and excitation search
  - Time evolution
  - ...
- **Exact contraction is intractable**
- Some current advances in ITensorNetworks.jl:
  - Approximate contraction routines
    Linjian Ma (UIUC)
  - Approximate evolution algorithms
    Joseph Tindall (Flatiron Institute)

Tree tensor networks

- Tree tensor network states
  - Defined on loop-free connected graph
  - **Robust local update routines**
- Readily available implementations for MPS (1D)
  - e.g. ITensorTDPV.jl
    - DMRG (ground states)
    - DMRG-X (excitations)
    - TDVP (time evolution)
    - MPO application (evolution)
- **Generalized to arbitrary trees in ITensorNetworks.jl**



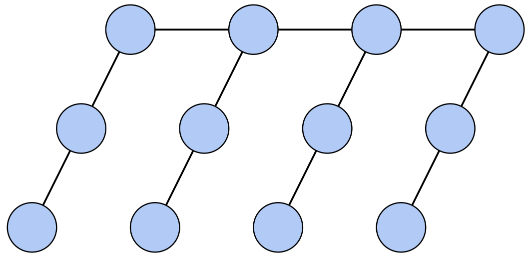




Example: ground state search on a tree

```julia
using NamedGraphs
using ITensors
using ITensorNetworks

# define system geometry
c = named_comb_tree((4, 3))
s = siteinds("S=1/2", c)

# define Hamiltonian
J1 = -1.0
J2 = 2
h = -0.2
Hos = ITensorNetworks.heisenberg(c; J1, J2, h)
H = TTN(Hos, s)

# run DMRG
nsite = 2
nsweeps = 10
cutoff = 1e-6
ψ0 = random_ttn(ComplexF64, s; link_space=10)
ψ = dmrg(H, ψ0; nsweeps, nsite, cutoff)
e = inner(ψ', H, ψ) / inner(ψ, ψ)
@show e;
```

- Define system geometry



- Define Hamiltonian as `ITensors.OpSum`
- Convert to tree tensor network operator
- Run `nsite` DMRG for `nsweeps`

Defining the Hamiltonian

$$H = J_1 \sum_{\langle i,j \rangle} \vec{S}_i \cdot \vec{S}_j + J_2 \sum_{\langle\langle i,j \rangle\rangle} \vec{S}_i \cdot \vec{S}_j + h \sum_i S_i^z,$$

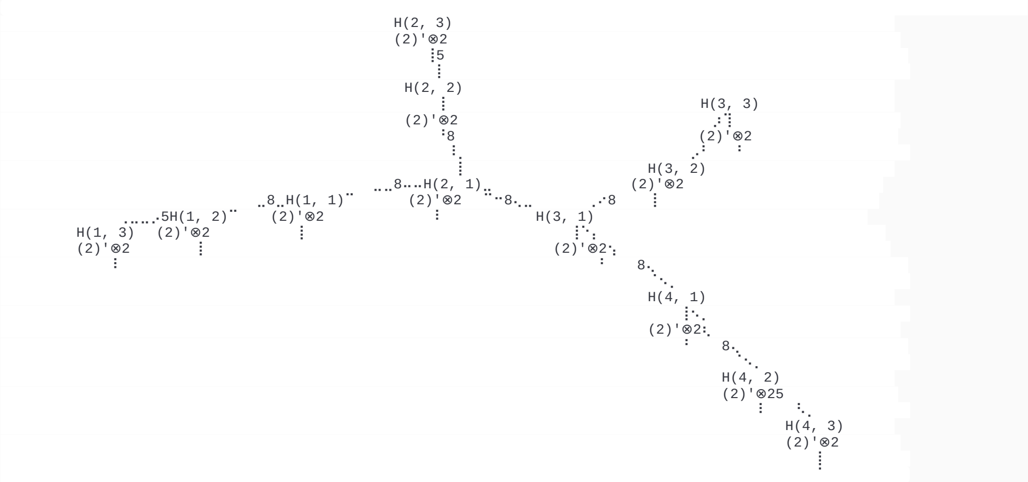
- As a lazy sum of tensor products

```
g = some_named_graph()
Hos = OpSum()
for (v1, v2) in nearest_neighbors(g)
    Hos += J1 / 2, "S+", v1, "S-", v2
    ...
end
for (v1, v2) in next_nearest_neighbors(g)
    Hos += J2 / 2, "S-", v1, "S+", v2
    ...
end
for v in vertices(c)
    ...
end
```

```
sum(
  -0.5 S+((1, 1),) S-((2, 1),)
...
  1.0  S-((2, 2),) S-((3, 1),)
...
  -0.2 Sz((2, 3),)
...
)
```

- Automatically converted to compressed dense tree tensor network operator

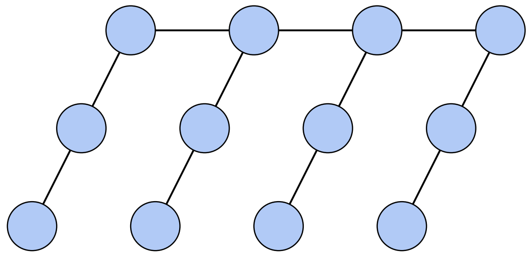
```
s = siteinds("S=1/2", c)
H = TTN(Hos, s)
@visualize H;
```



Example: time evolution on a tree

```julia
using NamedGraphs
using ITensors
using ITensorNetworks

# define system geometry
dims = (4, 3)
c = named_comb_tree(dims)
s = siteinds("S=1/2", c)

# define Hamiltonian
J1 = -1.0
J2 = 2
h = -0.2
Hos = ITensorNetworks.heisenberg(c; J1, J2, h)
H = TTN(Hos, s)

# run TDVP
nsite = 2
cutoff = 1e-6
t = 1.0
dt = 0.1
ψ0 = random_ttn(ComplexF64, s; link_space=10)
ψ = tdvp(H, -im * t, ψ0; time_step=-im * dt, nsite, cutoff);
```

- Define system geometry



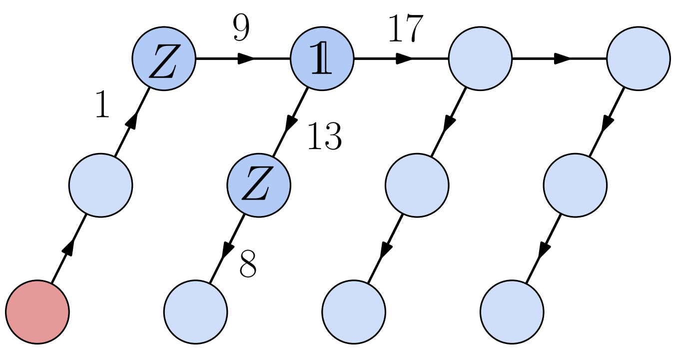
- Define Hamiltonian as `ITensors.OpSum`
- Convert to tree tensor network operator
- Run `nsite` TDVP for time `t` in steps of `dt`

Focus: Hamiltonian representation

- Conversion via symbolic representation as a network of sparse 'block tensors'

```
Hsparse = ITensorNetworks.finite_state_machine(Hos, s)
@show typeof(Hsparse)

    DataGraph{Tuple{Int64, Int64}, SparseArray{ITensors.Op}, ...}

@show typeof(Hsparse[2, 1])

    SparseArray{ITensors.Op, 3}
```

$$H = \cdots + 2Z_{(1,1)} \otimes Z_{(2,2)} + \dots$$



- Contraction of sparse network returns all Hamiltonian terms

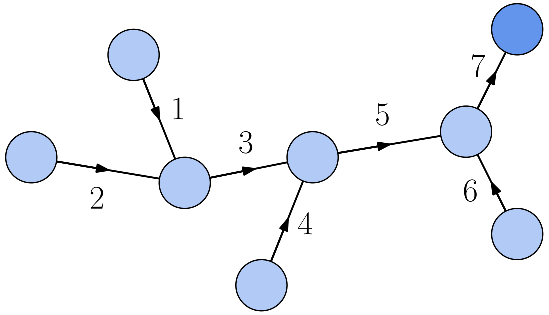
```
nonzero_pairs(Hsparse[1, 1])
  ⋮                            =>  ⋮
  CartesianIndex(1, 9)         => 2.0 Sz((1, 1),)
  ⋮                            =>  ⋮

nonzero_pairs(Hsparse[2, 1])
  ⋮                                 =>  ⋮
  CartesianIndex(9, 13, 17)         => 1.0 Id((2, 1),)
  ⋮                                 =>  ⋮

nonzero_pairs(Hsparse[2, 2])
  ⋮                            =>  ⋮
  CartesianIndex(13, 8)        => 1.0 Sz((2, 2),)
  ⋮                            =>  ⋮
```

- Similar sparse symbolic representations possible for operators on other geometries

Focus: customizing sweeping routines

- Builtin default sweeping patterns

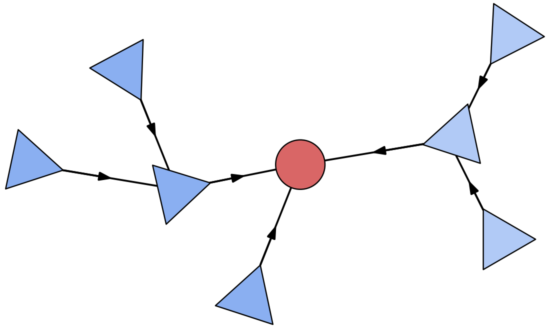
```
g = some_tree_graph()
sweep_regions = default_sweep_regions(nsite, g)
```



- Can provide custom sweeping pattern

- Standard solvers for specific algorithms

```
local_tensors = solver(H, local_tensors; kwargs...)
```



- Can supply custom local solver

Current features

- Familiar MPS-like capabilities, including:
  - Simple and intuitive Hamiltonian construction
  - Default one- and two-site sweeping patterns on general trees
  - Default local solvers for standard sweeping algorithms
- As well as
  - Custom sweeping patterns
  - Custom local solvers

Future directions

- Feature completeness, e.g.
  - QN conservation support
  - More tools for easy customization by users
- Novel features
  - Custom dynamic sweeping patterns
    - Miles Stoudenmire (Flatiron Institute)
  - Efficient subspace expansion
    - Benedikt Kloss (Flatiron Institute) and Wladislaw Krinitsin (FZ Jülich)
- Applications
  - Quantum circuit simulation
  - Quantum chemistry
  - Quantum dynamics
  - ...