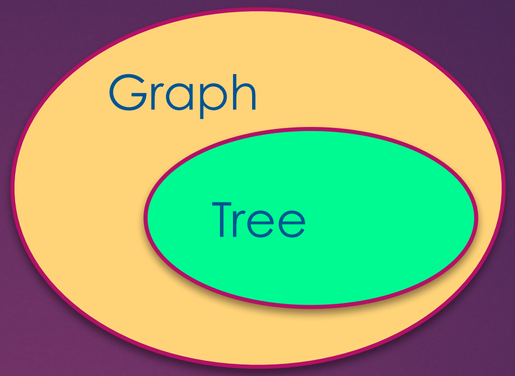
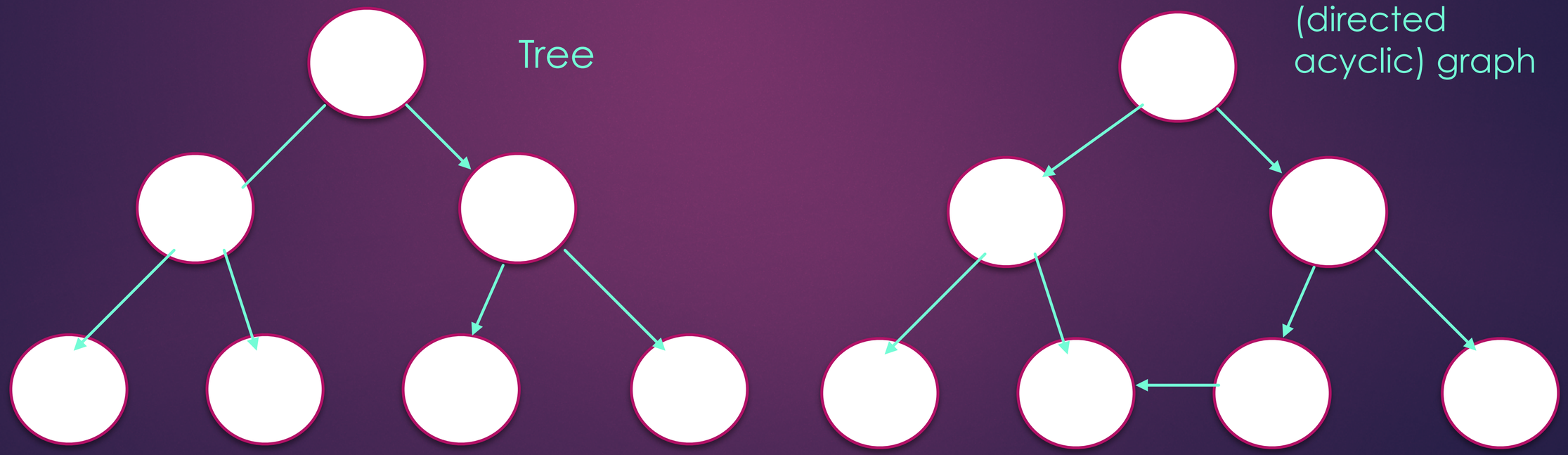
Current research theme

- CIRCUIT, DISTRICT HEATING NETWORKS
- GENERALIZED OPTIMAL SPARSE DECISION TREES (MULTIWAY SPLIT)

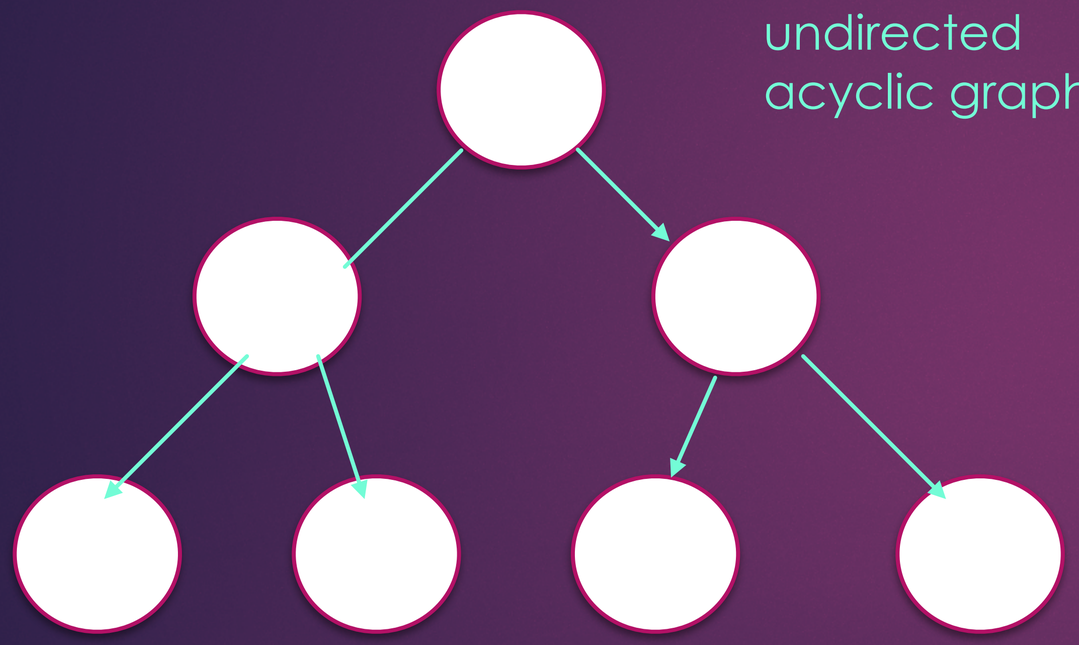
Networks

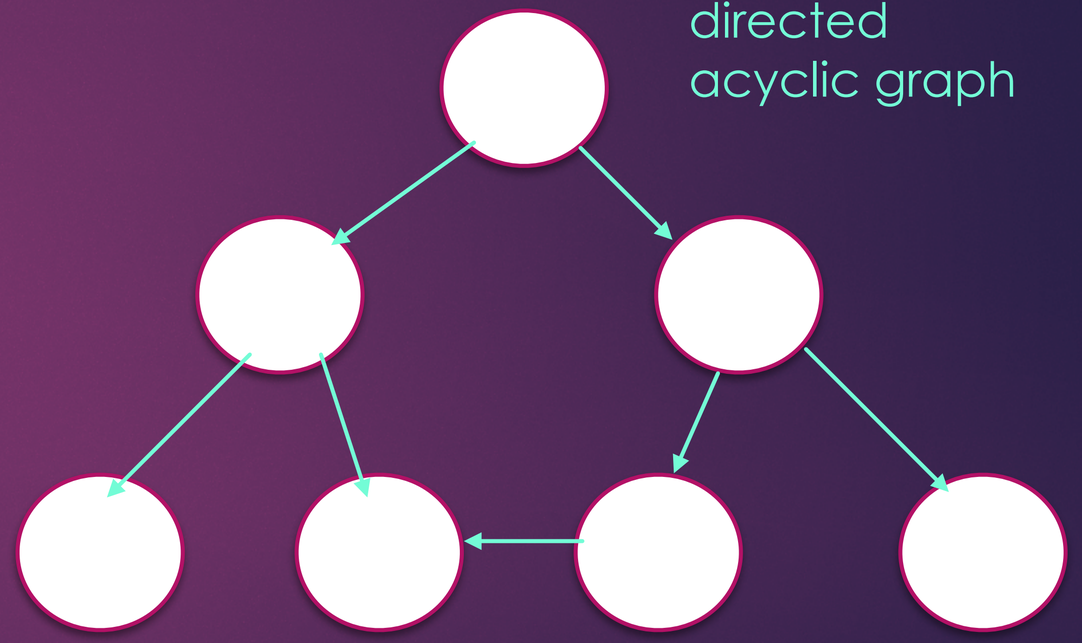


GRAPH, TREE



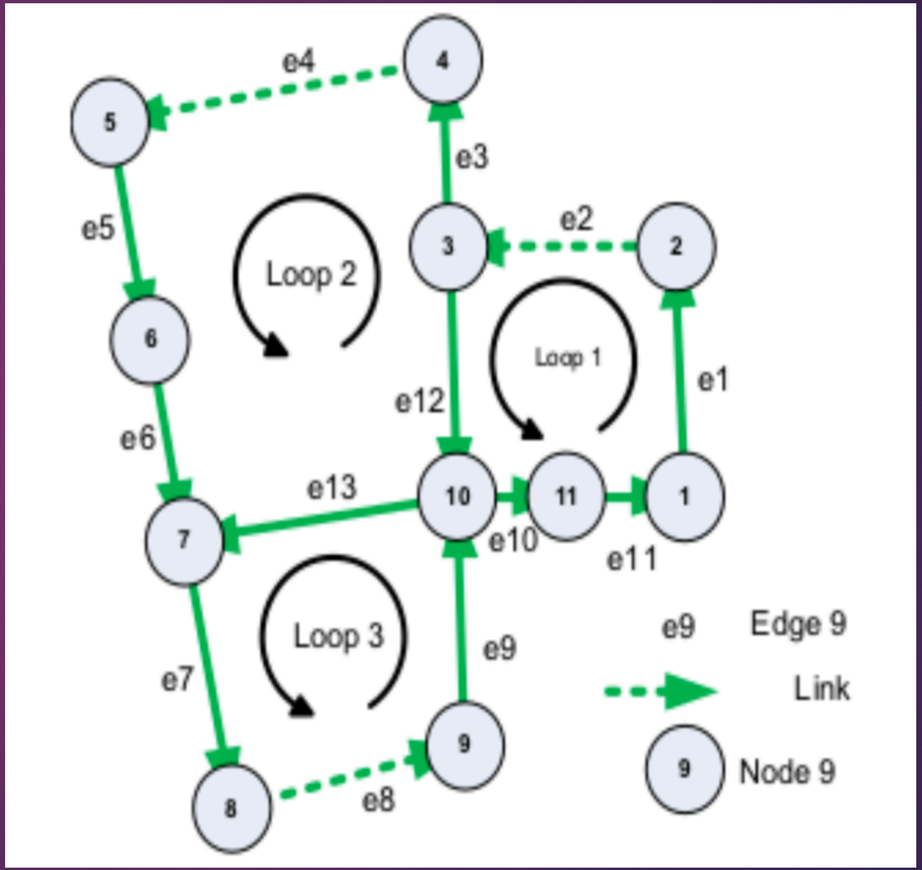
Graph vs Tree





Network modelling

- CIRCUIT (RADIAL)
- HYDRAULIC (POSSIBLY LOOPED)



Graph terminology

- G = (V, E) is a finite set of points and lines joining the points
- Can be directed or undirected
- Cycle : a path connects final node with initial node
- Tree : a connected graph with no cycles
- Spanning tree T : a subgraph that contains all nodes of G, with minimum number of edges

https://www.cs.sfu.ca/~ggbaker/zju/math/spanning-tree.html

- Incidence matrix $A = a_{ij}$, take 1 if jth edge is incident on ith node, 0 otherwise

- More detail on my Lachine summer lecture 2020:

Graph traversal

DEPTH FIRST SEARCH (DFS)

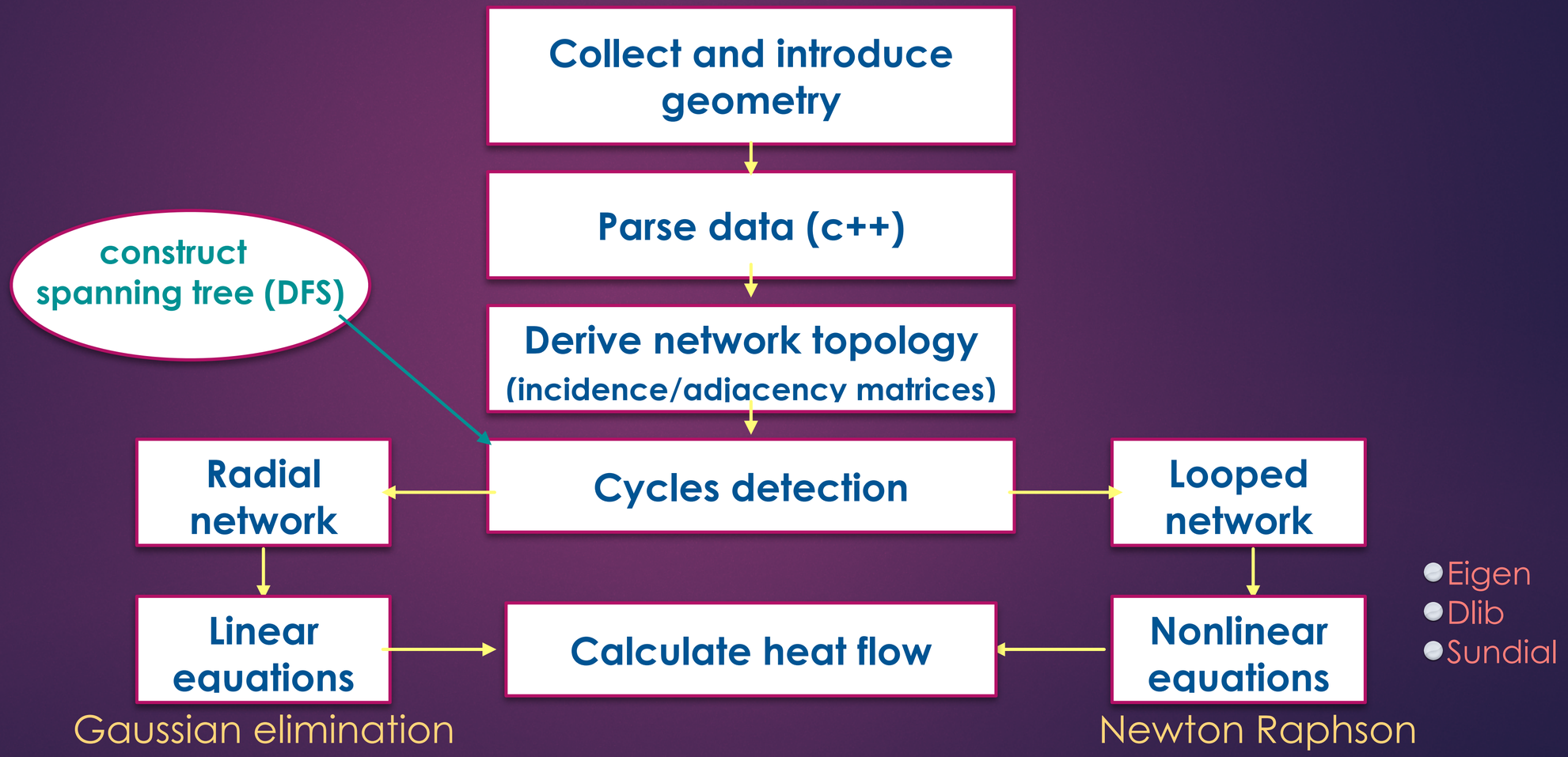
Find spanning tree via DFS:
Keep track of edges that form a tree

```
spanning_tree(Graph G) {
  for each node i: i.marked = false
  for some node i: f(i)
}
f(Node i) {
  i.marked = true
  for each j adjacent to i:
    if(!j.marked) {
      add(i,j) to output
      f(j) // DFS
    }
}
```

```
    DFS-iterative (G, s):                                    //Where G is
graph and s is source vertex
      let S be stack
      S.push( s )              //Inserting s in stack
      mark s as visited.
      while ( S is not empty):
          //Pop a vertex from stack to visit next
          v  =  S.top( )
         S.pop( )
         //Push all the neighbours of v in stack that are not visited
        for all neighbours w of v in Graph G:
            if w is not visited :
                      S.push( w )
                     mark w as visited


    DFS-recursive(G, s):
        mark s as visited
        for all neighbours w of s in Graph G:
            if w is not visited:
                DFS-recursive(G, w)
```

Network modelling(workflow)



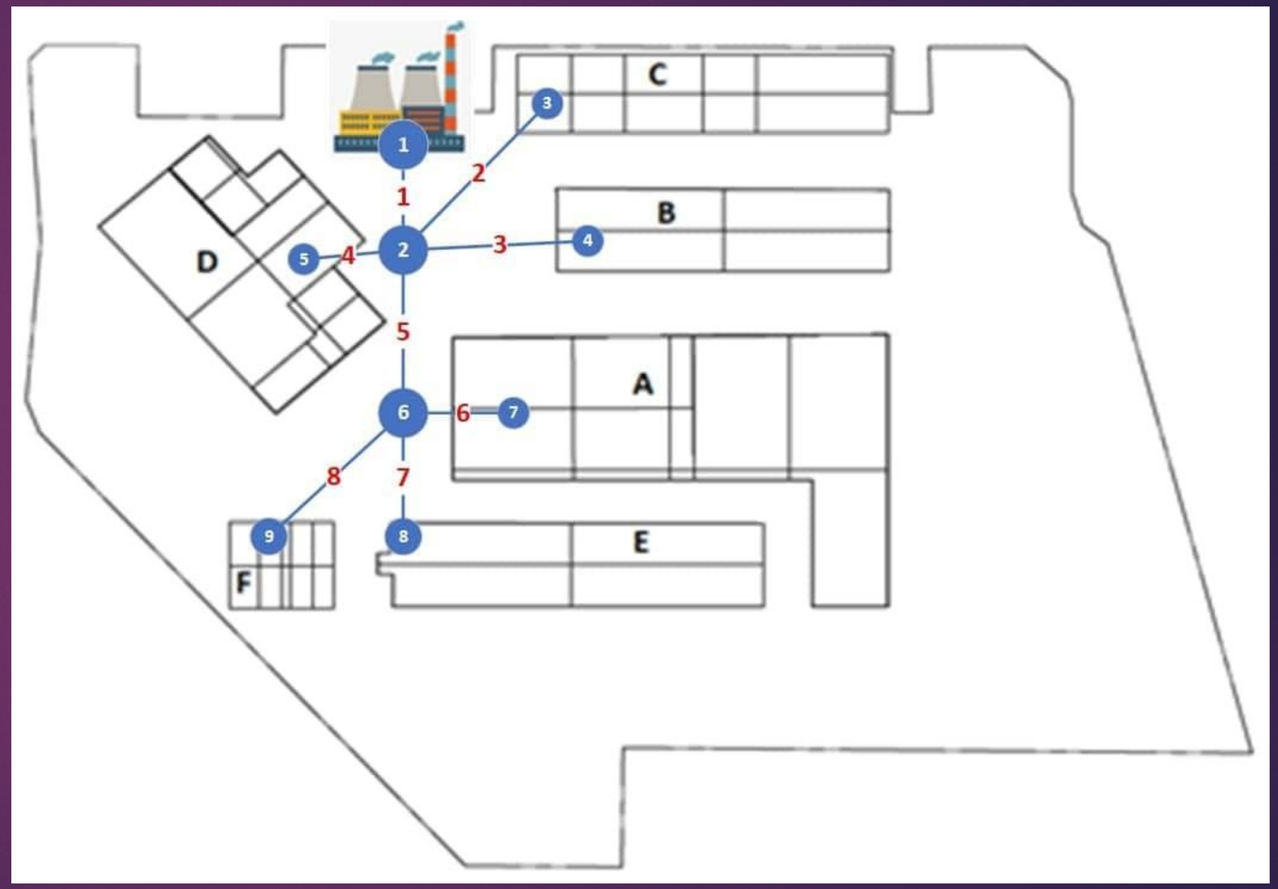
- Eigen
- Dlib
- Sundial

C++ implementations

- Circuit : https://github.com/leannejdong/autocircuit
- Parser (energy model): https://github.com/leannejdong/Parser_Pilar
- Hydraulic network
  https://github.com/leannejdong/SimpleHydraulicNetwork
  https://github.com/leannejdong/HydraulicNetwork
- C++ Toolchain:
  - Linux ubuntu 20.04 LTS, 8G RAM
  - IDE: CLion
  - Compiler: Clang/gcc 10.0
  - Build tool: CMake 3.12
  - Library: Eigen3.3, Boost
  - Unit testing: Doctest
  - Code style: clang-format
  - CI: Github Action

Linear Network

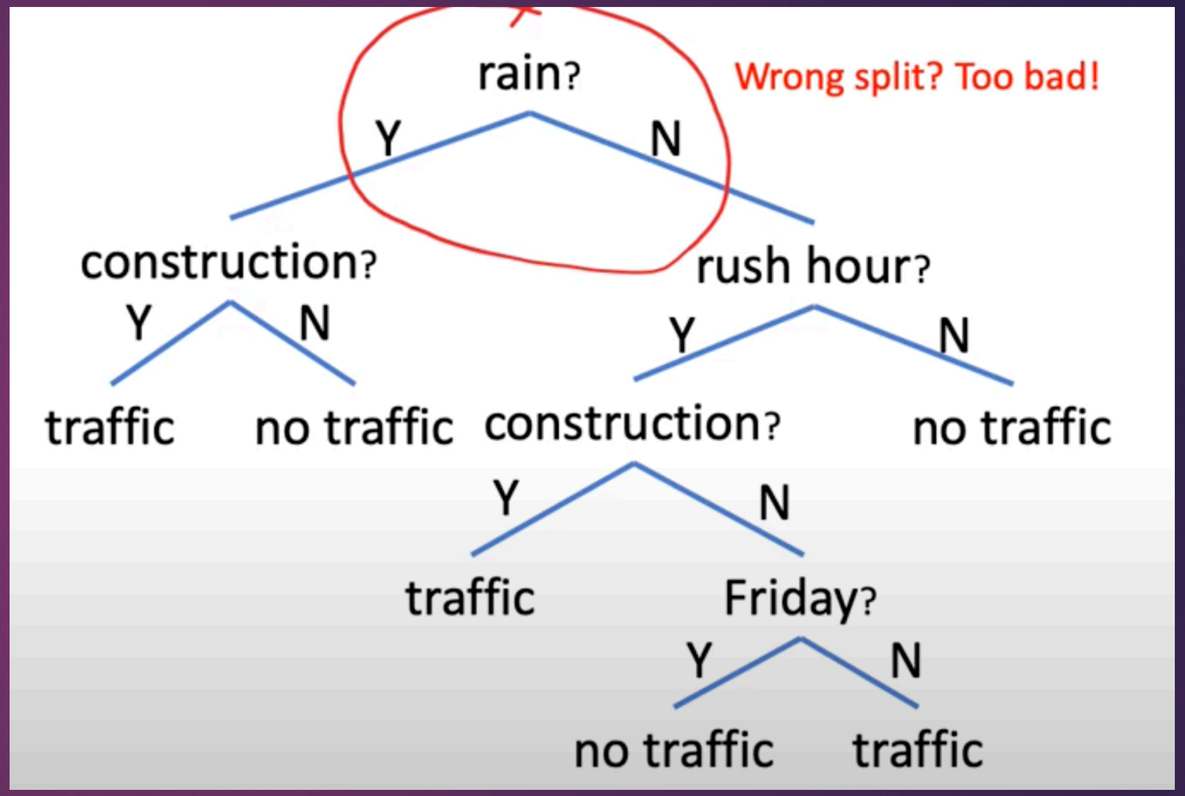
Paper:
A New Modeling Approach for Low-Carbon District Energy System Planning



Configurations of the buildings in the newly designed District Heating network

Multiway Split Optimal Sparse Decision Tree

- P: Decision problem can be "solved" in polynomial time (Easy)
- NP: Problem that can't be solved, but "verified" in polynomial time (Medium)
- NP-complete : These are verifiable problem (NP) for which any other problem in MP can be reduced to it in polynomial time. (Hard)
- A problem is **NP-hard** if "it is at least as hard as the hardest problem in NP-complete". (Hardest)

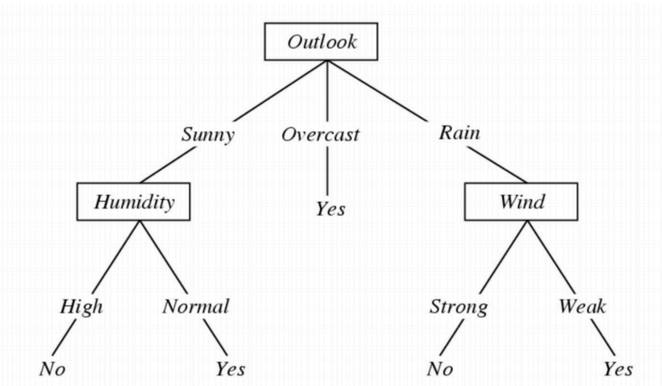


$$\min_{tree} \hat{L}(\text{tree}, \{x_i, y_i\}_i)$$

Operate on a weighted, additive, non-negative loss

$$= \underbrace{l(\text{tree}, \{x_i, y_i\}_i)}_{\text{Misclassification error}} + \underbrace{\lambda(\#\text{leaves in tree})}_{\text{sparsity}}$$

A decision tree
to decide whether to play tennis



Example of an optimal multiway tree on tennis dataset

Insights from GOSDT

Generalized and Scalable Optimal Sparse Decision Trees, *JMLR, 2020*
Jimmy Lin , Chudi Zhong , Diane Hu , Cynthia Rudin and Margo Seltzer

- Parallelism allow us to solve subproblems in parallel by delegating work to separate thread
- Analytical Bounds combined with Branch and bound allows us to prune the search space.
- Dynamic Programming decompose problem into subproblems which can be solved recursively through a function call.
- Represent subproblems by bit sets
- Permutation map : Discover identical trees already evaluated
- Caching of incremental computations.

Insights from MGOSDT

- m-way tree has been popular in CS applications. In graphics, m-way tree works much nicer on GPUs than binary trees. The shallowness usually help in the sense that we could compress it and save memory storing information. For instance, B-tree, a kind of balanced search tree that is optimized for external memory. This is due to its shallow structure, so the height of a B tree is minimized, meaning that we don't have to access all the items everytime we call the method recursively.

- In ML, Multiway split of a decision tree
  - Give a more natural way to handle multivalued categorical features than binary splits. Even if we have 3 categories, a 2-way split still seems less interpretable.
  - Give a more interpretable DT algorithm that can handle a larger variety of datasets.

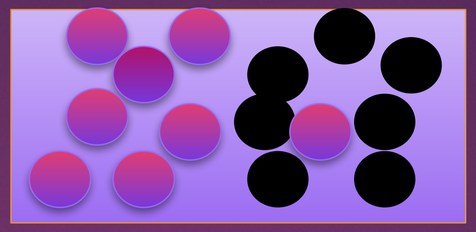
Primary data structures

GOSDT mains two primary data structures:

1. a concurrent **priority queue** is used to schedule problems to solve.
   A normal queue has a FIFO policy, whereas a priorrity queue sorts its items.
   The C++ stl implementation std::priority_queue is not thread-safe.
   The **concurrent** usage means that the values from all three can change due to push/pop methods in other threads. The tbb::concurrent_priority_queue support `std::priority_queue`'s methods of size, empty and swap and is thread-safe.

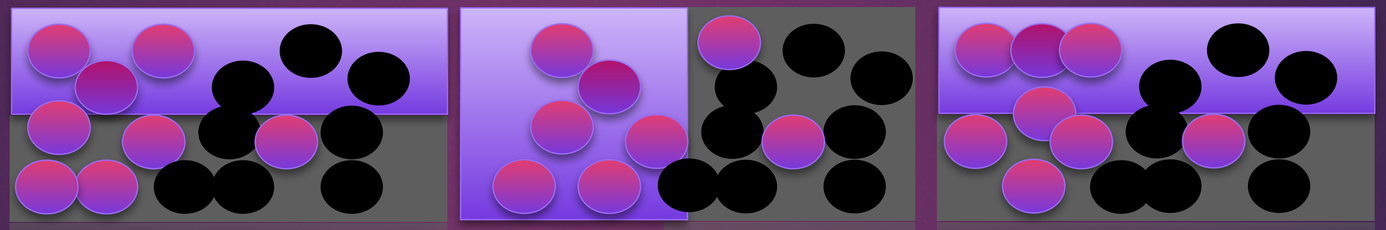
2. Dependency graph to store problems and their relationships

Start with the full dataset and naïve label

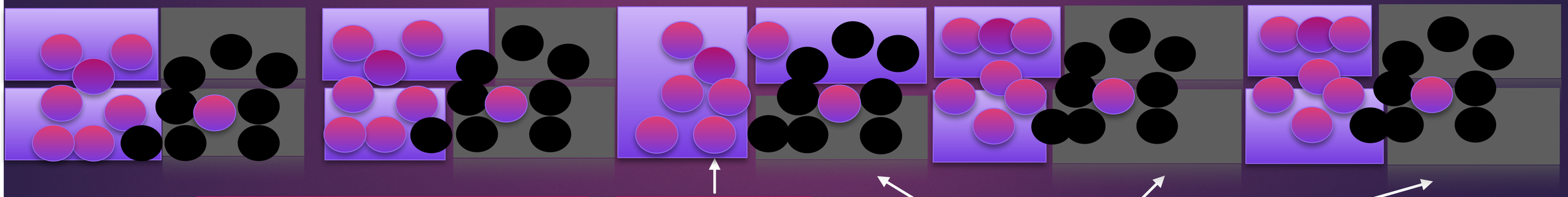


create every possible size 1 tree

Split it into subsets using each feature



Keep splitting (if permitted)



Consolidate duplication

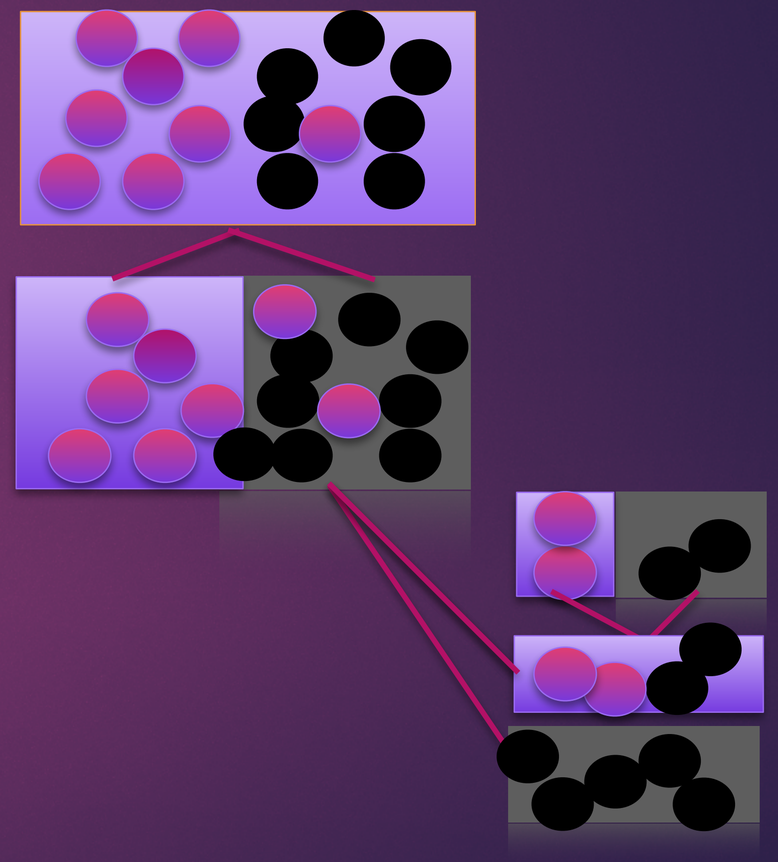
pure leave, can't split anymore

identical subproblems

Dynamic programming and dependency graph

Each subset must decide the best feature on which it should split

Once enough subsets have decided, the optimal tree emerges as a DAG of the best features

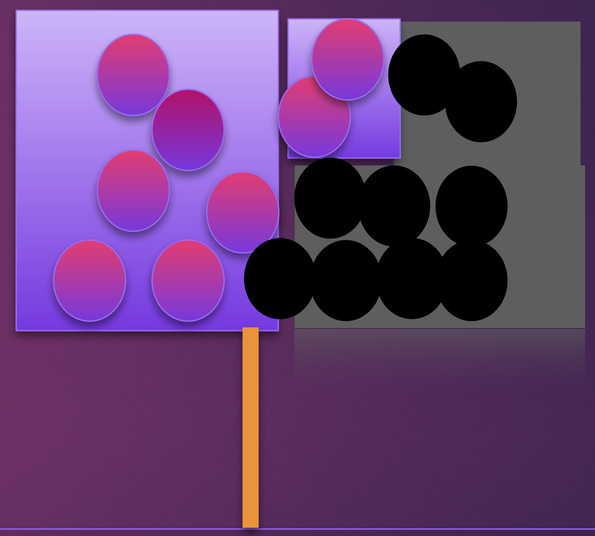


Dynamic programming and dependency graph

Each set is responsible for finding optimal features to sub-divide itself into additional subsets

Each subset must decide the best feature on which it should split

Once enough subsets have decided, the optimal tree emerges as a DAG of the best features



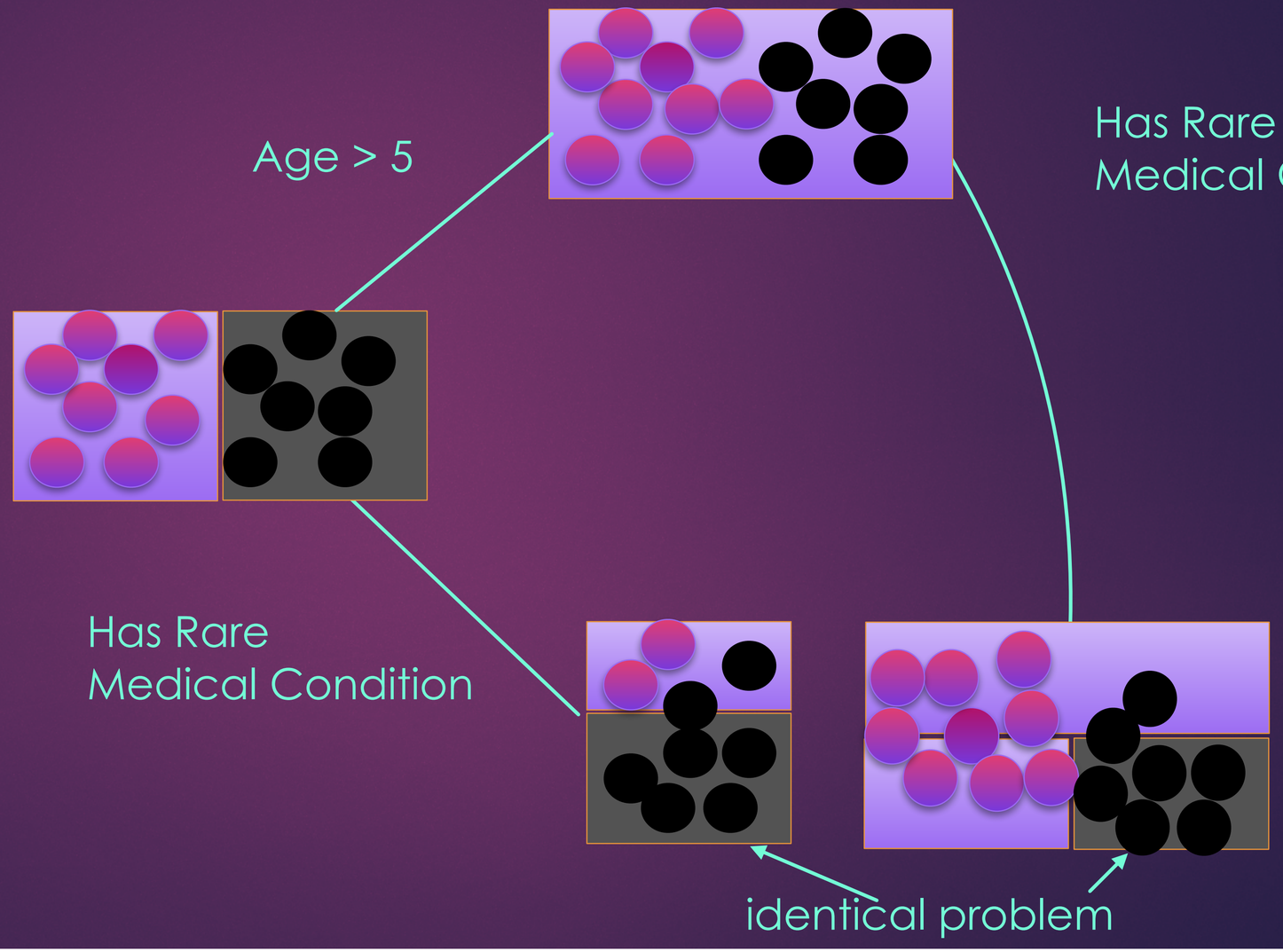
Once we find the optimal model, collapse it to get our optimal tree😄

Dynamic programming

Every dataset presents
a classification problem

Whether we filter the
data by a condition,
the resulting subset
presents a new problem.

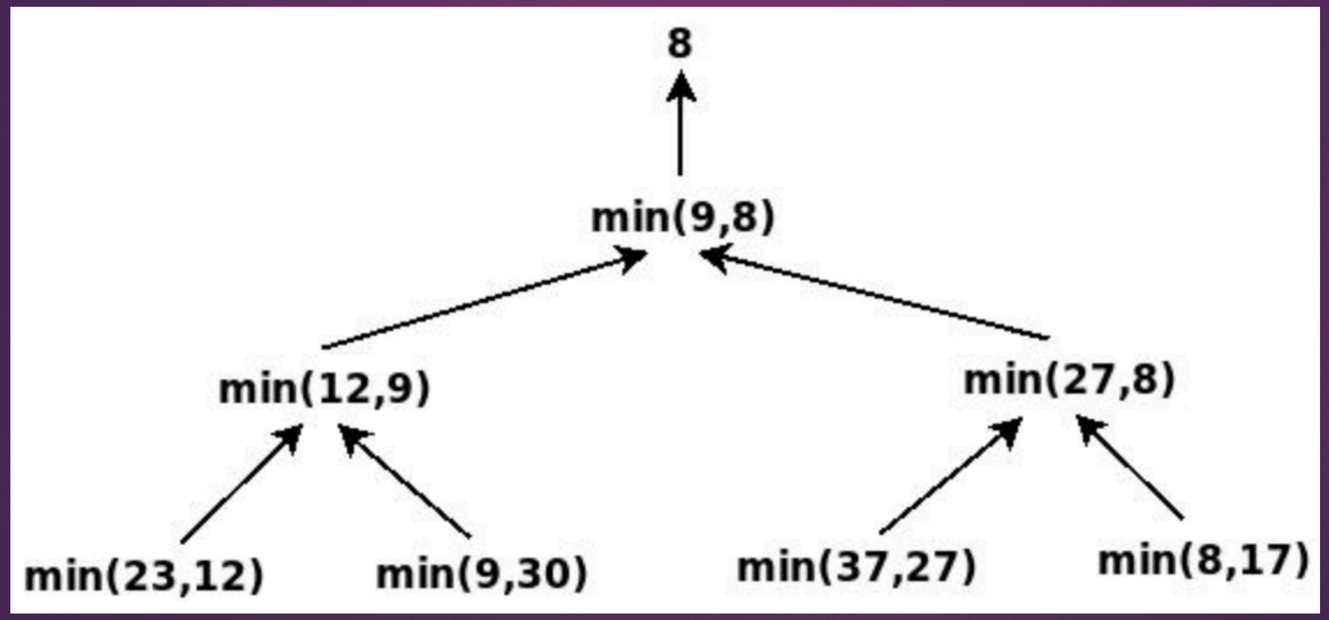
Each filtering creates
a new subproblem



Task Dependency Graph

FIND MINIMUM ELEMENT AMONG

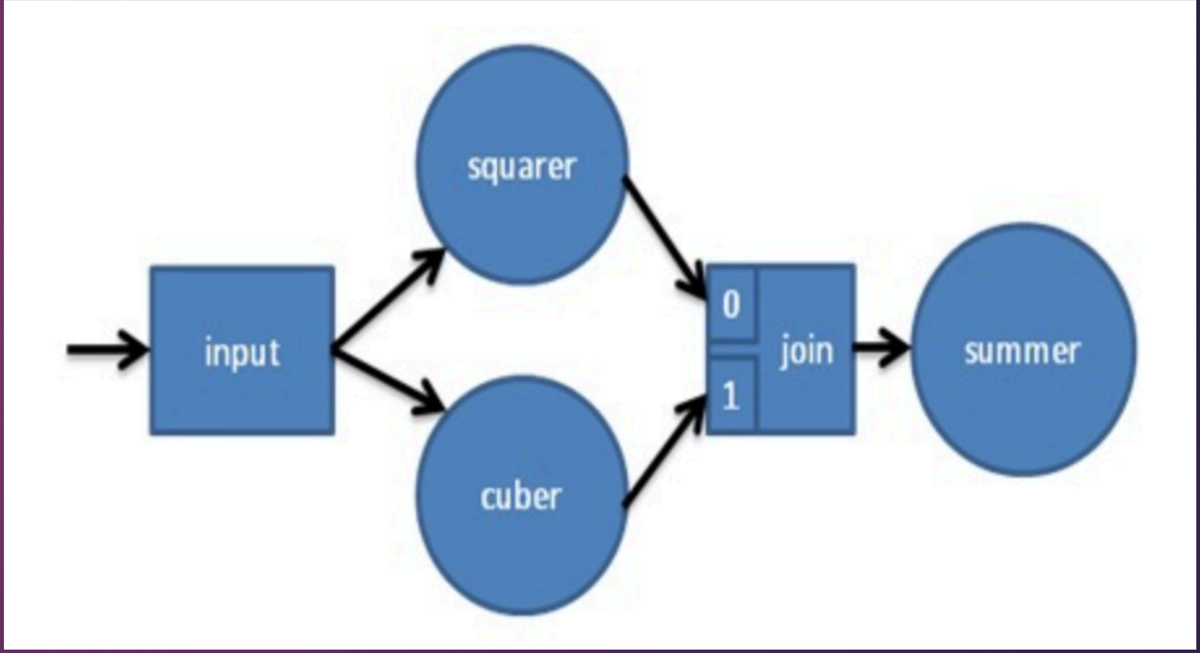
23, 12, 9, 30, 37, 27, 8, 17



Task Dependency Graph

TBB::flow

Talks
cppcon2020
Taskflow: A Parallel and Heterogeneous Task Programming System Using Modern C++ - Tsung-Wei Huang
cppcon2019
Concurrency in C++20 and Beyond - Anthony Williams - CppCon 2019



Task Dependency Graph

Message flow graph example: $\sum_{x} x^2 + x^3 \;\; \forall\, x = 1,2,\ldots 10$

Intel's TBB is a general purpose parallel programming library

```
#include <cstdio>
#include "tbb/flow_graph.h"
#include <fmt/core.h>

using namespace tbb::flow;

auto square(){
    return [](int v){ return v*v; };
}
auto cube(){
    return [](int v){ return v*v*v; };
}
class sum {
  int &my_sum;
public:
  sum( int &s ) : my_sum(s) {}
  int operator()( tuple< int, int > v ) {
    my_sum += get<0>(v) + get<1>(v);
    return my_sum;
  }
};
```

```
int main() {
  int result = 0;

  graph g;
  broadcast_node<int> input(g);
  function_node<int,int> squarer( g, unlimited, square() );
  function_node<int,int> cuber( g, unlimited, cube() );
  join_node< tuple<int,int>, queueing > join( g );
  function_node<tuple<int,int>,int>
      summer( g, serial, sum(result) );
  make_edge( input, squarer );
  make_edge( input, cuber );
  make_edge( squarer, get<0>( join.input_ports() ) );
  make_edge( cuber, get<1>( join.input_ports() ) );
  make_edge( join, summer );
  for (int i = 1; i <=10; ++i)
      input.try_put(i);
  g.wait_for_all();

  fmt::print("Final result is {}.", result);
  return 0;
}
```

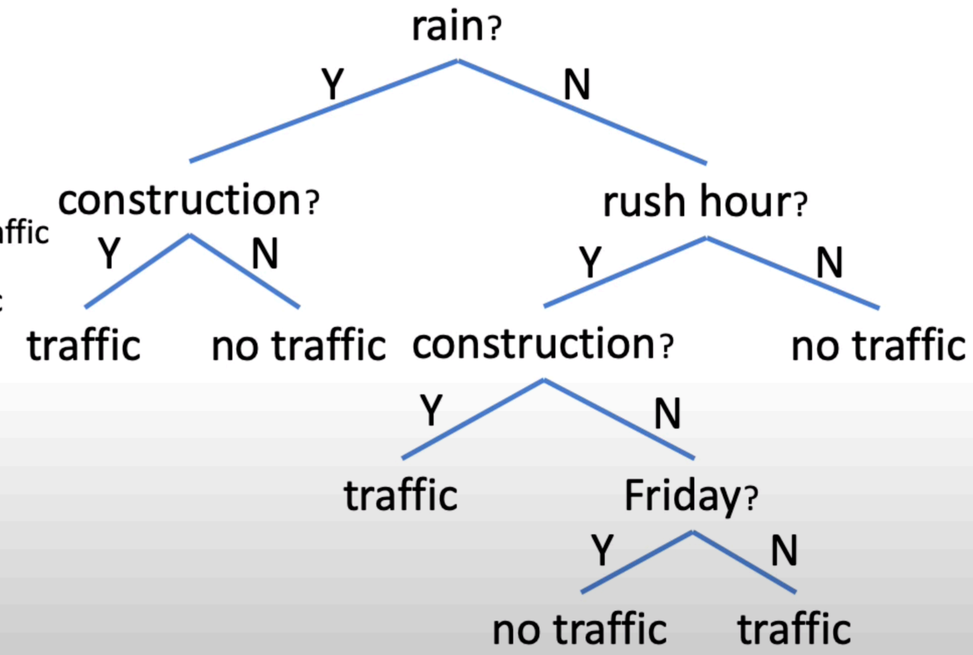
https://godbolt.org/z/E1Y6oY6qx

Task Dependency Graph

- Dependency relationship : the solution of one problem depends on another
- In mgosdt, line 25 of models.hpp, the task dependency graph is the set of optimal tree
- The tasks are the sub-problems, or subtrees
- The tasks are entered by bit sets (from the bit mask class)

Bit sets Representation of sub-problems

rain & construction & traffic
[1000010001001110000.....0]
rain & no construction & no traffic
[0110001000000000110.....1]
no rain & rush hour & construction & traffic
[0001000100000001000.....0]
no rain & rush hour & no construction & Friday and no traffic
[0000100000000000001.....0]
no rain & rush hour & no construction & Friday and traffic
[0000000010000000000.....0]
no rain & no rush hour & no traffic
[0000000000011000000.....0]

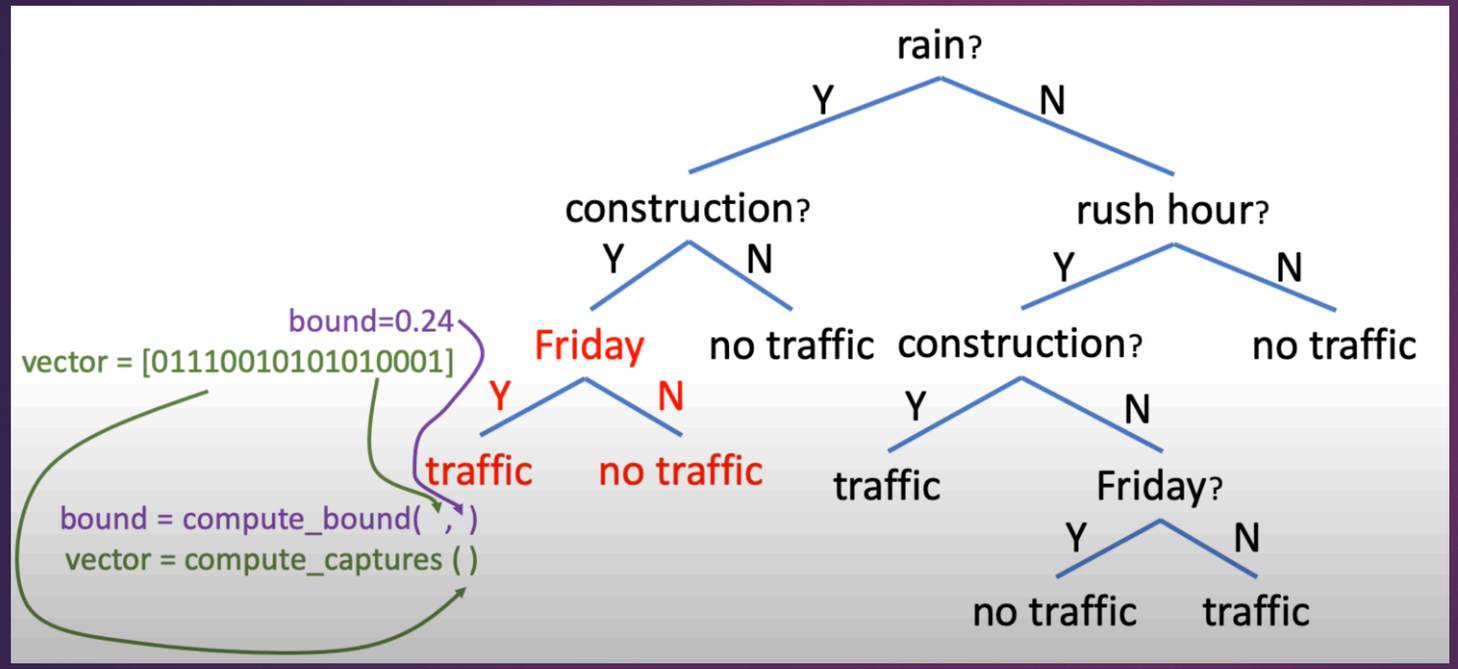


We store a bit vector, indicating which data point has features corresponding to the features described by its leaves (Bit masking. See `bitmask.cpp`, `dataset.cpp` and `encoder.cpp`)

Implementation: makeBitVector

Caching of incremental computations

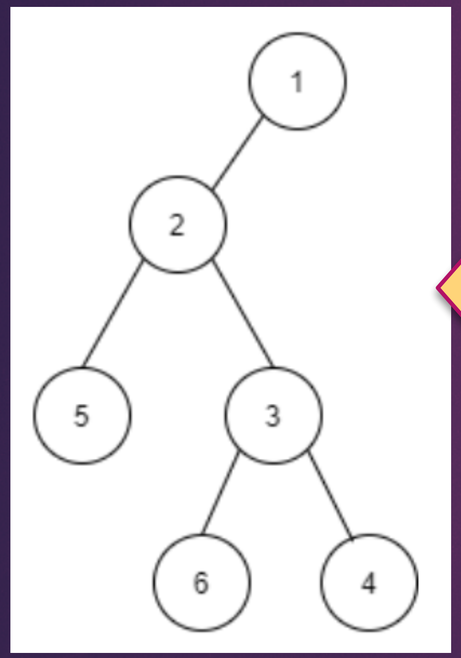
The bounds of bit vector in each leave allows us to evaluate the children of the leaves - should we split next…

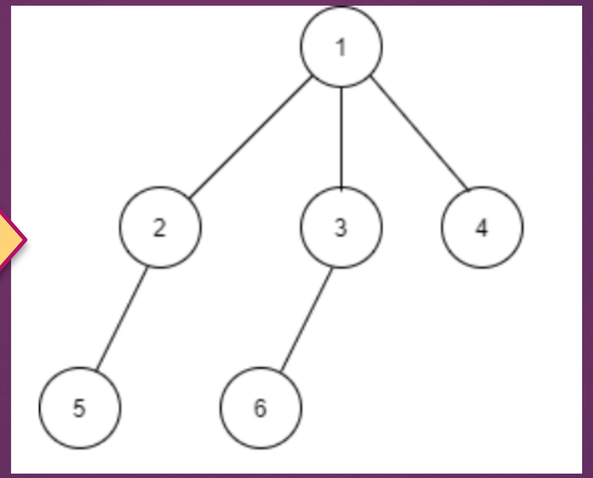


Algorithm

RECOVER N-ARY TREE FROM BINARY

MGOSDT: https://godbolt.org/z/GTzePKa9f





- Declare an array to store left and right child, initialize with -1;
- If ith node has at least 1 child, then 1st child is its left child;
- Then traverse children of ith node. If there are still children left after this node, then make the next child the right child of the current node;
- Return 'answer'

C++ implementations

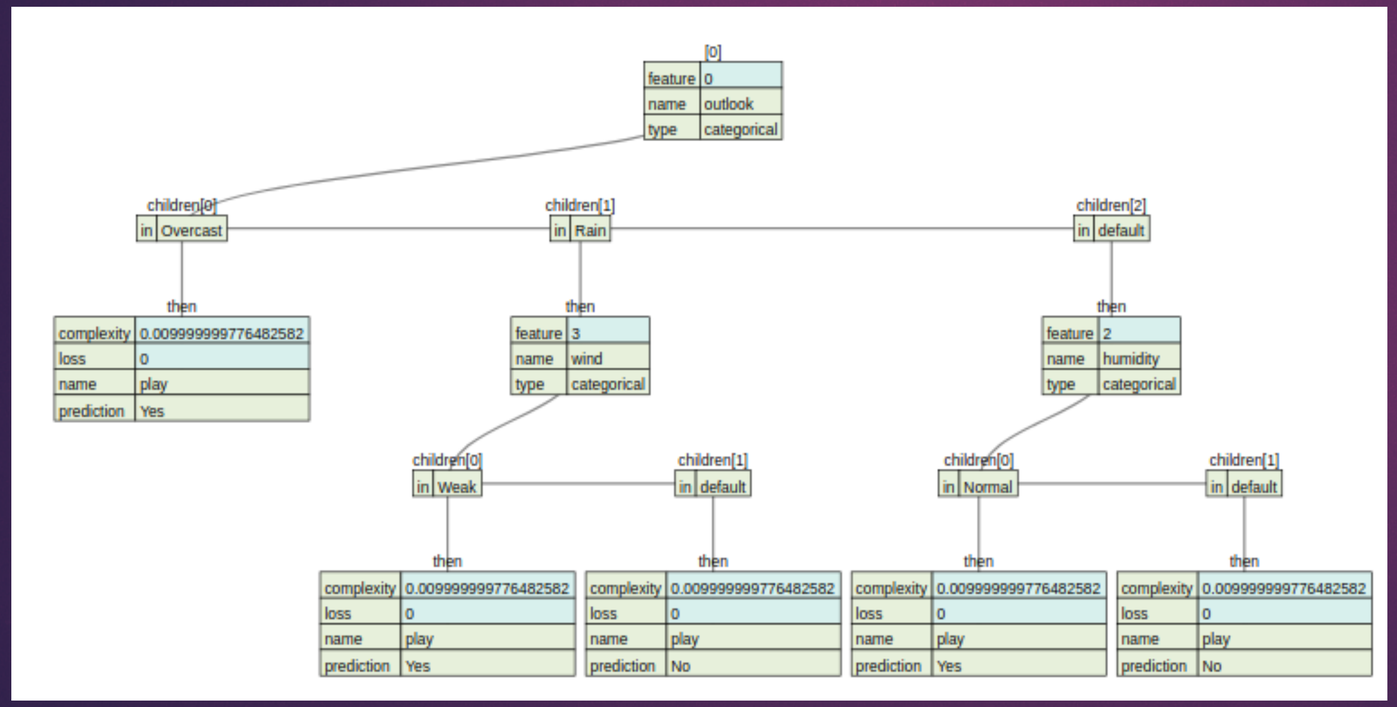
- MGOSDT : https://gitlab.com/leannejdong/mgosdt
- C++ Toolchain:
  - Linux ubuntu 20.04 LTS, 62G RAM, Intel i7,16 Core tensorbook
  - IDE: CLion
  - Compiler: Clang/gcc 10.0
  - Build tool: CMake 3.12
  - Library, framework: Boost, Intel TBB, OpenCL, GMP, nlohmann/json, Pybind11
  - Unit testing: Doctest
  - Code style: clang-format
  - CI: Gitlab Action

Results

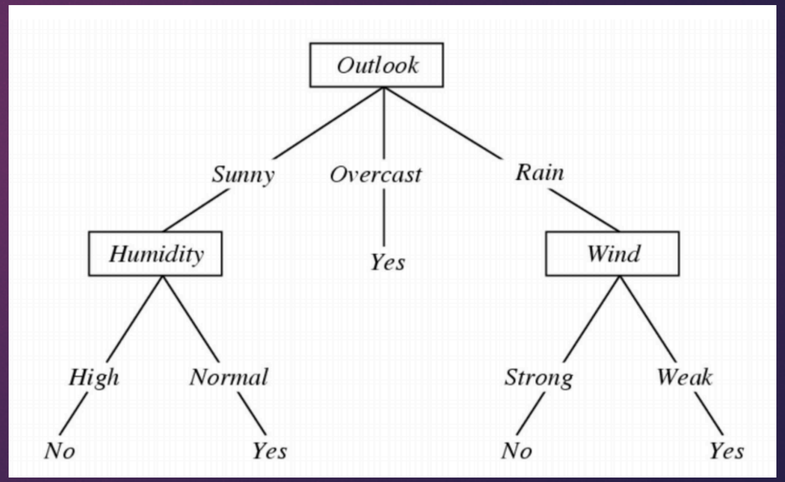
Python interface is available 😄

```
leanne@tensorbook:~/Dev/mgosdt$ time gosdt experiments/datasets/tennis/tennis.csv experiments/configurations/debug.json >doc/tennis.txt 2>&1

real    0m0.004s
user    0m0.004s
sys     0m0.000s
```



Actual tree output on GitLab



Summary

Our work presents

- A new algorithm convert binary decision tree to n-ary decision tree.
  See model::intersect and model::summarize in src/model.cpp
- Efficient memory management. With help of thread and address sanitizer, we debug our codes with address sanitizer with `fsanitize` flag. There were large number of memory leaks, data races and unitialized memory issues on the original GOSDT codebase.
- Performance optimization.
  - Completed profiling that allows us to detect which part of mgosdt are slow.
  - We came up with ideas about what speed up mgosdt and refactored.
  - Unit tests are performed to validate things actually work.

An interesting non-postdoc approach

N-ary encryption of datasets:
N-ary error correction coding scheme

Thank You!