Olden Benchmarks Suite

Power

Benchmark

Power Pricing Computation

Data Structures

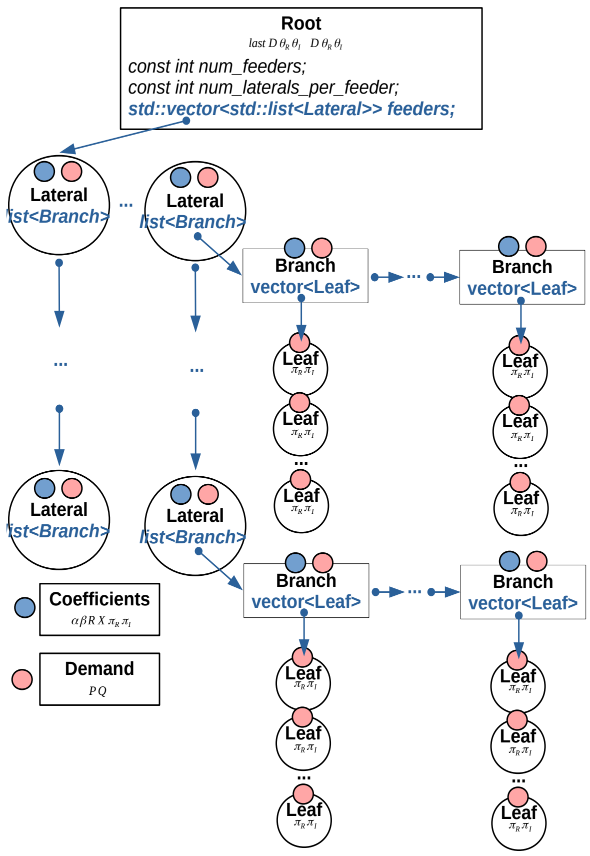
1) Composite structure with C arrays and pointer-based linked lists
2) Composite structure built with C++ std::vector and std::list

Implementation details

1) ***Standard library based implementation.***

power.h
power.cpp

This implementaton is based on a complex structure laid out in a memory as an ensemble of objects of Root, Lateral, Branch and Leaf classes. The objects are linked with the help of STL *std::vector* and *std::list* containers. Scheme below illustrates the data structure.


Root
last $D\ \theta_R\ \theta_I$ $D\ \theta_R\ \theta_I$
const int num_feeders;
const int num_laterals_per_feeder;
std::vector<std::list<Lateral>> feeders;
Lateral
list<Branch>
Branch
vector<Leaf>
Leaf
$\pi_R\ \pi_I$
...
Coefficients
$\alpha\ \beta\ R\ X\ \pi_R\ \pi_I$
Demand
$P\ Q$

Computation starts with a call to *Root::compute()*. The method accumulates power demand **Demand(P,Q)** from all lists of Laterals. Accumulation starts with the end of each list. Each lateral accumulates its power demand from all its branches and the latter in turn accumulate power demand from all their leaves. The leaves call *optimize_node()* method, which does a chunk of scientific computations (gradients, vectors, etc.) which compute P and Q.

Root::compute() is called iteratively from a while loop of *power_pricing_problem()*. Iteration continues up until P and Q error becomes less than the required epsilon.

1) ***Original tree based implementation.***

power_original.h
power_original.cpp
build.cpp
compute.cpp

Original implementation does the same computation, but uses regular C structs linked through pointer chains. The implementation calls compute() recursively through pointer links and invokes the same logic.

Feasibility study results

The C++ implementation with std::vector and std::list runs more than 3 times faster.

```
power original: 1.3899 seconds.
power: 0.419872 seconds.
```